

IP6 Algoria Worksheet

---

# Bachelor Thesis

University of Applied Sciences Northwestern Switzerland - Computer Sciences

Studierende:	Reto Frey Kevin Zogg
Betreuender Dozent:	Prof. Dr. Christoph Stamm
Experte:	Charles Zehnder
Revision:	1.0
Datum:	18. August 2011

# 1 Abstract

Algoria ist ein von der Fachhochschule Nordwestschweiz entwickeltes Programm, mit welchem Datenstrukturen gezeichnet und Algorithmen ausgeführt werden können. Algoria Worksheet ist eine eigenständige Applikation, in welcher Algoria integriert ist. Es ermöglicht den Dozierenden von Algorithmen und Datenstrukturen, ihre Aufgaben und Prüfungsvorbereitungen als interaktive Arbeitsblätter zur Verfügung zu stellen. Die Studierenden können ihrerseits diese Arbeitsblätter mit Algoria Worksheet öffnen und ihre Lösung erarbeiten. Mithilfe der integrierten Algoria Engines können Aufgaben gestellt und gelöst werden, die sich spezifisch mit dem Zeichnen, Sortieren und Analysieren von Datenstrukturen befassen. Ein Lösungsprüfer kann die Lösung der Studierenden überprüfen und Hilfestellungen zum Erreichen der Musterlösung geben. Mit Algoria Worksheet wird ein Programm entstehen, welches sich schrittweise von einem ergänzenden Hilfsmittel für Studierende bis zu einem essentiellen Bestandteil des Unterrichtes entwickeln könnte.

# Inhaltsverzeichnis

<b>1</b>	<b>Abstract</b>	<b>2</b>
<b>2</b>	<b>Einleitung</b>	<b>5</b>
<b>3</b>	<b>Disposition</b>	<b>6</b>
3.1	Ausgangslage . . . . .	6
3.2	Anforderungsbeschreibung . . . . .	6
3.2.1	Algoria Worksheet . . . . .	6
3.2.2	Arbeitsblätter . . . . .	7
3.2.3	Lösungsprüfer . . . . .	7
3.2.4	Benutzerrechte . . . . .	7
3.3	Anforderungen . . . . .	8
3.3.1	Funktionale Anforderungen . . . . .	8
3.3.2	Nicht-funktionale Anforderungen . . . . .	11
3.4	Personas . . . . .	12
3.4.1	Dozierender . . . . .	12
3.4.2	Studierender . . . . .	13
3.5	Anwendungsszenarien . . . . .	14
3.5.1	Vor und Nachbereiten des Unterrichts . . . . .	14
3.5.2	Während des Unterrichts . . . . .	14
3.5.3	Prüfungsvorbereitungen . . . . .	14
<b>4</b>	<b>Datenstruktur der Aufgabenblätter</b>	<b>16</b>
4.1	Datenstruktur der Aufgaben . . . . .	16
4.2	Versionierung . . . . .	17
4.3	Serialisierung . . . . .	17
<b>5</b>	<b>Programm</b>	<b>18</b>
5.1	Applikationsdesign . . . . .	19
5.1.1	Entwurfsmuster MVVM . . . . .	19
5.1.2	Instanzen von Algoria . . . . .	20
5.1.3	Benutzerunterscheidung . . . . .	21
5.1.4	Prism . . . . .	22
5.1.5	AvalonDock . . . . .	22
5.1.6	Klassendiagramm . . . . .	23
5.1.7	Assertions . . . . .	24
5.2	Algoria . . . . .	25
5.3	Aufgabenstellung . . . . .	26
5.3.1	RichTextBox . . . . .	26
5.3.2	FlowDocumentViewer . . . . .	27
5.4	Seitenleiste . . . . .	28
5.4.1	Aufbau . . . . .	28
5.4.2	Bildvorschau . . . . .	28

5.5	Multiple Choice . . . . .	29
5.5.1	Aufbau . . . . .	29
5.5.2	Dozierendenansicht . . . . .	29
5.5.3	Studierendenansicht . . . . .	31
5.6	Events . . . . .	32
5.6.1	Events in Algoria Worksheet . . . . .	32
5.7	Kopieren und Einfügen . . . . .	34
5.7.1	Windows Zwischenablage . . . . .	34
5.7.2	Kopieren und Einfügen in Algoria Worksheet . . . . .	34
5.8	Drag & Drop . . . . .	38
5.8.1	Kopieren und Verschieben mittels Drag & Drop . . . . .	38
5.8.2	Animation . . . . .	41
5.9	Hilfe . . . . .	43
5.9.1	Microsoft Hilfedatei . . . . .	43
5.9.2	Kontextsensitive Hilfe . . . . .	43
<b>6</b>	<b>Lösungsprüfer</b>	<b>45</b>
6.1	Subgraph-Isomorphismen und Graph matching . . . . .	46
6.2	Metrik . . . . .	47
6.3	Grobkonzept . . . . .	48
6.4	Algorithmus mit minimaler Anzahl Swaps . . . . .	49
6.4.1	Theorie . . . . .	50
6.5	Detaillkonzept . . . . .	53
<b>7</b>	<b>Reflexion</b>	<b>55</b>
7.1	Windows Presentation Foundation . . . . .	55
7.2	Lösungsprüfer . . . . .	55
7.3	Planung . . . . .	56
7.4	Ausblick . . . . .	56
<b>8</b>	<b>Ehrlichkeitserklärung</b>	<b>57</b>

## 2 Einleitung

Der Algorithmen und Datenstrukturen Unterricht ist ein grundlegender Bestandteil der Informatikausbildung an der Fachhochschule Nordwestschweiz. Ausschlaggebend für das Verständnis der besprochenen Themen ist es nicht nur, dass Dozierende eine Möglichkeit haben, Datenstrukturen und dazugehörige Algorithmen zu visualisieren, sondern auch das Vorhandensein von Übungsaufgaben, in welchen Studierende das Wissen vertiefen und anwenden können.

Mit Algoria wurde ein Grundstein für das Visualisieren von Unterrichtsinhalt aus dem Algorithmen und Datenstrukturen Kurs gelegt. Das Programm bietet sowohl einem Dozierenden als auch einem Studierenden eine interaktive Zeichenfläche, welche gezeichnete Datenstrukturen erkennen kann und Operationen auf diesen zulässt und visualisiert. Dies deckt einen Teil der Visualisierung ab. Eine Möglichkeit Aufgaben zu definieren ist in Algoria nicht vorgesehen.

Exorciser<sup>[2]</sup> ist ein Programm, welches die Möglichkeiten bietet, elektronisch definierte Aufgaben zu lösen. Dieses Programm wurde für den Unterricht in Theoretischer Informatik entwickelt und bietet dafür eine Übungsumgebung. Ein grosser Vorteil gegenüber Aufgaben auf Papier ist das unmittelbare Feedback. Mit Algoria Worksheet wird der Vorteil des sofortigen Feedbacks für den Algorithmen und Datenstrukturen Unterricht verfügbar gemacht.

In Algoria Worksheet wurden die Möglichkeiten, welche durch Algoria gegeben sind, gebunden. Dozierende sind in der Lage, Arbeitsblätter zu erstellen. Dies können zum Einen Datenstrukturen sein, auf welche die Studierenden einen, im Aufgabentext mitgelieferten, Algorithmus ausführen müssen. Die Lösung ist in einer, durch den Studierenden gezeichneten, Datenstruktur gefordert. Zum Anderen ist auch das Stellen von Multiple Choice Fragen möglich. Auch hier kann der Aufgabenstellung eine gezeichnete Datenstruktur (zum Beispiel für das Fördern des Verständnisses) beigelegt werden. Der Sinn hier ist aber, dass die Lösung durch das Markieren von richtigen Aussagen erfolgt.

Studierende können ein solches Arbeitsblatt lösen und mit einem Lösungsprüfer ihre Lösung gegen die Musterlösung testen lassen. Durch einen Lösungsprüfer, der nicht nur zwischen “Richtig und Falsch” entscheidet, sondern auch korrekte Teillösungen findet, ist ein Feedback möglich, welches einem Studierenden die Möglichkeit gibt selbstständig seine Lösung zu verbessern. Eine solche Rückmeldung ist natürlich nur in Aufgaben möglich, in welchen die Studierenden etwas zeichnen müssen. In Multiple Choice Aufgaben können pro Antwort jedoch nur richtig oder falsch markiert werden, hier ist eine Teillösung nicht möglich.

## 3 Disposition

Die Unterkapitel der Disposition liefern Informationen zur Ausgangslage und zum Anforderungskatalog dieser Arbeit. Der Anforderungskatalog wird in einem ersten Schritt in textueller Form dargestellt. In einem weiteren Kapitel werden die Funktionalen und Nicht-funktionalen Anforderungen in Tabellenform aufgelistet.

### 3.1 Ausgangslage

Im Rahmen des Projekts Algoria entwickelt das IMVS eine neuartige Tablet-PC Applikation, die den Tablet-PC des Dozierenden zum virtuellen Whiteboard werden lässt. Datenstrukturen sollen auf herkömmliche Art skizziert und entsprechende Algorithmen darauf angewendet werden können. Die Applikation Algoria erkennt die skizzierten Datenstrukturen mit Methoden der künstlichen Intelligenz, bildet sie logisch im Hauptspeicher nach und stellt sie zudem verschönert dar. Dadurch entsteht die Möglichkeit, typische Algorithmen (wie beispielsweise das Sortieren eines Arrays oder das Einfügen in einen Baum) auf die skizzierte Datenstruktur anzuwenden und den Ablauf des Algorithmus schrittweise darzustellen bzw. zu animieren. So kann sich der Dozierende auf die Erklärung konzentrieren und wird nicht durch das Skizzieren an der Wandtafel abgelenkt.<sup>[1]</sup>

### 3.2 Anforderungsbeschreibung

Im Folgenden werden die Anforderungen an die Software beschrieben, von welcher in einem ersten Schritt ein Konzept erarbeitet wurde. Anhand dieser konzeptionellen Planung wurde dann das Programm entwickelt. Die Anforderungen werden in vier Blöcke unterteilt. In einem ersten Block wird das zu erstellende Programm beschrieben. Ein zweiter Abschnitt beinhaltet Informationen über die Arbeitsblätter, welche in diesem Programm definiert und gelöst werden können. Ein weiterer Block befasst sich mit der Thematik, eine Lösung automatisch korrigieren zu können. Der letzte Block behandelt die Benutzerrechte, welche in Algoria Worksheet vorhanden sein müssen.

#### 3.2.1 Algoria Worksheet

Das Programm Algoria Worksheet steht sowohl Studierenden als auch Dozierenden zur Verfügung. Für Dozierende stehen primär Funktionen für das Erstellen von Arbeitsblättern im Vordergrund. Diese beinhalten das Eintragen der Musterlösung, des Anfangsstandes, eines Instruktions-Textes und mögliche Antworten in Multiple Choice Aufgaben. Studierende verwenden Algoria Worksheet vor allem für das Lösen eines solchen Arbeitsblattes. Für das Eintragen und Anzeigen von Studierenden-Lösungen, Musterlösungen und Anfangsstand wurde Algoria eingebunden.

Für die graphische Benutzeroberfläche wird die Windows Presentation Foundation (kurz WPF) verwendet. Die Programmiersprache ist C#. Das erstellte Programm ist dadurch nur auf Computern mit Microsoft Windows und installiertem .Net Framework ausführbar.

### 3.2.2 Arbeitsblätter

Ein Arbeitsblatt besteht aus mehreren Aufgaben. Aufgaben können in zwei Kategorien unterteilt werden. Aufgaben, welche als Lösung eine aufgezeichnete Datenstruktur fordern und solche, welche durch das Ankreuzen von Multiple Choice Antworten beantwortet werden.

Eine jede Zeichen-Aufgabe beinhaltet folgende Informationen:

- Aufgabenstellung als formatierter Text (rich text)
- Ausgangslage als gezeichnete Datenstruktur (kann leer sein)
- Musterlösung als gezeichnete Datenstruktur
- Stand der Lösung als gezeichnete Datenstruktur

Folgende Informationen sind in Multiple Choice Aufgaben enthalten:

- Aufgabenstellung als formatierter Text (rich text)
- Ausgangslage als gezeichnete Datenstruktur (kann leer sein)
- Multiple Choice Antworten

Für jede Multiple Choice Antwort wird zusätzlich die Information abgespeichert, ob die Antwort korrekt ist, und ob sie von einem Studierenden als richtig markiert wurde.

### 3.2.3 Lösungsprüfer

Das Programm verfügt über einen Lösungsprüfer. Dieser ist in der Lage, beide Kategorien von Aufgaben zu prüfen. Für Multiple Choice Aufgaben wird nur die gegebene Antwort mit der Lösung abgeglichen. Für Aufgaben, welche eine gezeichnete Datenstruktur als Lösung fordern, gelten folgende Punkte:

- Es wird eine Metrik definiert, welche die Ähnlichkeit zwischen Studierenden-Lösung und Musterlösung bestimmt.
- Der Lösungsprüfer zeigt an, wo sich ein Fehler befindet.
- Der Lösungsprüfer liefert Hilfestellungen.

### 3.2.4 Benutzerrechte

Das Programm wird von zwei verschiedenen Nutzergruppen, welche unterschiedliche Anforderungen und Berechtigungen haben, verwendet. Aus diesem Grund ist es möglich, Rechte im Programm für bestimmte Gruppen freizuschalten oder zu sperren.

### 3.3 Anforderungen

Im Folgenden werden die Anforderungen in Tabellenform, gegliedert nach funktionalen und nicht-funktionalen Aspekten, aufgeführt. Die Anforderungen wurden im Lauf der Arbeit mit dem Auftraggeber regelmässig besprochen und überarbeitet. Die Anforderungen sind priorisiert mit den Kategorien *Muss* (M) und *Kann* (K). Zusätzlich wird angegeben, welche Anforderungen *Erfüllt* (E) wurden und welche noch *Offen* (O) sind.

#### 3.3.1 Funktionale Anforderungen

ID	Abh.	Beschreibung	M/K	E/O
F1		Interaktives Arbeitsblatt		
F1.1		Userinterface		
F1.1.1		Das GUI beinhaltet eine Zeichenfläche.	M	E
F1.1.2		Das GUI beinhaltet die Aufgabenstellung.	M	E
F1.1.3		Das GUI beinhaltet eine Aufgabenauswahl.	M	E
F1.1.4		Das GUI kann die Musterlösung anzeigen.	M	E
F1.1.5		Das GUI kann eine Aufgabe darstellen.	M	E
F1.1.6		Das GUI beinhaltet eine Seitenanzeige (mit Vorschaubilder der Zeichenfläche) zur schnellen Auswahl von Aufgaben.	M	E
F1.1.7		Das GUI bietet eine Dozierenden- und eine Studierendenansicht.	M	E
F1.1.8	F1.1.1, F1.1.2, F1.1.4, F1.1.6	Die Dozierendenansicht bietet zusätzlich folgende Funktionalitäten: <ul style="list-style-type: none"> <li>• Das Aufgabenstellungstextfeld ist als Rich-Text Editor implementiert.</li> <li>• Eine Ausgangslagen-Zeichenfläche.</li> <li>• Eine Musterlösungs-Zeichenfläche.</li> <li>• Die Seitenanzeige kann mittels Drag &amp; Drop zum Sortieren von Aufgaben in einem Worksheet verwendet werden.</li> <li>• Zusätzliche Menüeinträge für das Erstellen und Editieren von Worksheets und Aufgaben.</li> </ul>	M	E
F1.1.9		Das GUI beinhaltet in der Dozierendenansicht eine Anzeige für den Lösungsablauf.	M	O
F1.1.10	F1.1.9	Die Anzeige für den Lösungsablauf bietet die Möglichkeit eine partielle Ordnung der Lösungsschritte zu definieren.	K	O
F1.1.11		Das GUI beinhaltet einen Bereich für Multiple Choice Antworten.	M	E
F1.1.12	F1.1.11	Dozierende können Multiple Choice Antworten bearbeiten.	M	E
F1.1.13		Es existiert eine kontextsensitive Hilfe für jeden Bereich des GUI.	M	E
F1.2		Worksheets		
F1.2.1		Worksheets können gespeichert werden.	M	E
F1.2.2		Worksheets können geöffnet und gelesen werden.	M	E
F1.2.3		Worksheets können verändert werden.	M	E
F1.2.4		Worksheets bestehen aus mehreren Aufgaben.	M	E
F1.2.5		Worksheets können mittels Doppelklick geöffnet werden.	K	O
F1.2.6		Worksheets können per Drag & Drop in eine Programminstanz gezogen und geöffnet werden.	K	O

F1.2.7		Worksheet Dateien werden mit einer Versionsnummer abgespeichert, um das Versionsmanagement für spätere Erweiterungen zu ermöglichen.	M	E
F1.3	Aufgaben			
F1.3.1		<p>Aufgaben beinhalten:</p> <ul style="list-style-type: none"> <li>• Aufgabenstellung (als Text)</li> <li>• Musterlösung</li> <li>• Ausgangslage (als Zeichnung)</li> <li>• Stand der Lösung</li> <li>• Änderungen am Rechtesystem</li> <li>• Voraussetzung für das Einsehen der Musterlösung</li> </ul>	M	E
F1.3.2		Die Lösungsschritte werden in einer partiellen Ordnung gespeichert.	M	O
F1.3.3		Eine Aufgabe kann aus mehreren Teilaufgaben bestehen.	K	E
F1.3.4		Die Musterlösung kann nach einem eingestellten Datum eingesehen werden.	K	O
F1.3.5		Die Musterlösung kann nach dem Erreichen eines gewissen Prozentsatzes an korrekten Eingaben eingesehen werden.	M	O
F1.3.6		Aufgaben können zwischen Programminstanzen kopiert werden (Copy & Paste / Drag & Drop).	M	E
F1.3.7		Aufgaben können innerhalb eines Worksheets kopiert werden (Copy & Paste / Drag & Drop).	M	E
F1.3.8		Der Inhalt einer Algoria-Zeichenfläche kann in die Windows Zwischenablage kopiert werden. Er wird als Algoria-Content (Datenstruktur) und als Bitmap zur Verfügung gestellt.	M	E
F2	Lösungsprüfer			
F2.1		Der Lösungsprüfer vergleicht die Musterlösung mit dem momentanen Stand der Lösung.	M	O
F2.2		Der Lösungsprüfer gibt den Übereinstimmungsgrad mit der Musterlösung an.	M	O
F2.3	F1.3.2	Der Lösungsprüfer überprüft die Richtigkeit der partiellen Ordnung.	M	O
F2.4	F1.3.2	Der Lösungsprüfer identifiziert einen falschen Lösungsschritt.	K	O
F2.5	F1.3.3	Der Lösungsprüfer gibt den Lösungsfortschritt an.	K	O
F2.6		Der Lösungsprüfer schreibt Fehlermeldungen in ein Errorlog nieder.	M	O
F3	Benutzerrechte			
F3.1		<p>Das Programm ist in zwei Versionen erzeugbar:</p> <ul style="list-style-type: none"> <li>• Dozierendenversion</li> <li>• Studierendenversion</li> </ul>	M	E

F3.2	F3.1	<p>Studierende sind im Stande folgende Funktionalitäten zu nutzen:</p> <ul style="list-style-type: none"> <li>• Worksheets laden</li> <li>• Lösungen eintragen</li> <li>• Lösungen prüfen</li> <li>• Musterlösung anzeigen</li> <li>• Worksheet mit momentanem Stand speichern</li> </ul>	M	E
F3.3	F3.1	<p>Dozierende erhalten zusätzlich zu F3.2 folgende Funktionen:</p> <ul style="list-style-type: none"> <li>• Neue Aufgaben erstellen <ul style="list-style-type: none"> <li>– Aufgabenstellung</li> <li>– Musterlösung eintragen</li> <li>– Ausgangslage eintragen</li> </ul> </li> <li>• Neue Aufgaben editieren <ul style="list-style-type: none"> <li>– Aufgabenstellung editieren</li> <li>– Musterlösung editieren</li> <li>– Ausgangslage editieren</li> </ul> </li> <li>• Worksheets verwalten <ul style="list-style-type: none"> <li>– Worksheets hinzufügen</li> <li>– Worksheets entfernen</li> <li>– Reihenfolge der Worksheets ändern</li> </ul> </li> </ul>	M	E
F3.4	F3.1	<p>Für eine jeweilige Benutzergruppe können Rechte per Worksheet gesperrt werden.</p>	M	O

Tabelle 3.1: Funktionale Anforderungen

### 3.3.2 Nicht-funktionale Anforderungen

ID	Abh.	Beschreibung	M/K	E/O
NF1		Interaktives Arbeitsblatt		
NF1.1		Userinterface		
NF1.1.1		Die Aufgabenstellung kann ausgeblendet werden.	M	E
NF1.1.2		Der Benutzer kann schnell von einer zu einer beliebigen anderen Aufgabe wechseln.	M	E
NF1.1.3		Aufgabenstellung kann aus dem Worksheetfenster heraus geschoben werden und als separates Fenster funktionieren.	K	O
NF1.1.4		Aufgabenstellung kann an mehreren vordefinierten Orten im Programm angedockt werden.	K	O
NF1.1.5		Das GUI kann auf beliebig grossen Bildschirmen dargestellt werden.	M	E
NF1.1.6		Die Musterlösung und die Ausgangslage werden übereinander in jeweils einem Layer angezeigt. Durch einen Slider lässt sich die Transparenz der Layer verändern. Bei einem Wert von unter 50% Transparenz wechselt ein Layer in den Hintergrund und verliert die Fähigkeit Eingaben anzunehmen.	K	E
NF1.2		Worksheets		
NF1.2.1		Ein Worksheet muss mindesten 20 Aufgaben beinhalten können.	M	E
NF2		Lösungsprüfer		
NF2.1		Der Lösungsprüfer liefert innerhalb von 2 Sekunden eine Antwort	M	O
NF2.2		Der Lösungsprüfer soll erst verfügbar sein, wenn der momentane Stand der Lösung nicht mehr der Ausgangslage entspricht.	K	O
NF2.3		Der Lösungsprüfer verwendet Farben für die Visualisierung von korrekten und falschen Lösungen.	K	O
NF3		Benutzerrechte		
NF3.1		Es kann per Buildflag entschieden werden ob eine Studierenden oder Dozierenden-Version erstellt wird.	M	E
NF3.2		Die Version des Programmes wird in der Titelleiste angezeigt (Dozierenden- / Studierendenversion)	M	E

Tabelle 3.2: Nicht-funktionale Anforderungen

## 3.4 Personas

Um Algoria Worksheet so gut wie möglich auf die Bedürfnisse und Fähigkeiten der Benutzer anzupassen, wurden Personas erstellt. Alle Benutzerinteraktionen wurden, mit Rücksicht auf die gegebenen Personas, umgesetzt. Ein Usabilitytest half dabei, verschiedene Punkte der Gestaltung zu verbessern. So konnte ein Grossteil aller Funktionalitäten und Möglichkeiten so integriert werden, dass sie von den Benutzern intuitiv verstanden und gefunden werden.

Da Algoria Worksheet zwei Hauptbenutzergruppen hat, namentlich die Dozierenden und die Studierenden, ist es schwierig, für alle Benutzer eine einwandfreie Benutzbarkeit zu gewährleisten. Deshalb wurden im Bezug auf die Benutzbarkeit die Studierenden priorisiert. Dies bedeutet, dass in Konfliktsituationen die Umsetzung eher auf die Studierenden ausgelegt wurde. Begründet wird dies dadurch, dass Algoria Worksheet von vielen verschiedenen Studierenden benutzt wird. Eine schlechte Benutzbarkeit würde möglicherweise dazu führen, dass die Studierenden gar nicht mit Algoria Worksheet arbeiten.

### 3.4.1 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	<ul style="list-style-type: none"> <li>• Die Software soll intuitiv bedienbar sein.</li> <li>• Die Software soll leicht verständlich sein.</li> <li>• Verschiedene Aufgabentypen sollen einfach erfasst werden können (Zeichnen, Multiple Choice, etc.).</li> <li>• Das Programm soll auch die von Studierenden erarbeiteten Lösungen anzeigen können.</li> <li>• Besitzt einen Laptop mit Tabletfunktion, d.h. er kann darauf zeichnen. Daher verwendet er diese Funktion gerne.</li> </ul>
Vorkenntnisse	Als Dozent für Algorithmen und Datenstrukturen mit einem Dokortitel und einigen Jahren Erfahrung in einer unterrichtenden Form, verfügt er über einige Vorkenntnisse im Bereich Informatik und Didaktik. Er ist es sich gewohnt mit Microsoft 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 3.3: Dozierenden Persona

### 3.4.2 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	<ul style="list-style-type: none"> <li>• Die Software soll intuitiv bedienbar sein.</li> <li>• Die Software soll leicht verständlich sein.</li> <li>• Besitzt einen Laptop mit Tabletfunktion, d.h. er kann darauf zeichnen. Daher verwendet er diese Funktion gerne.</li> </ul>
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 3.4: Studierenden Persona

## 3.5 Anwendungsszenarien

Algoria Worksheet soll in einem ersten Schritt an der Fachhochschule Nordwestschweiz im Unterricht für Algorithmen und Datenstrukturen eingesetzt werden. Das Programm dient den Studierenden zum Vor- und Nachbereiten der Kontaktstunden, sowie zur Prüfungsvorbereitung. Ausserdem würde Algoria Worksheet den Unterricht so ergänzen, dass damit Aufgaben und Übungsbesprechungen gemacht werden können. Im Folgenden werden die Szenarien detaillierter beschrieben.

### 3.5.1 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.

### 3.5.2 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.

### 3.5.3 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.

## 4 Datenstruktur der Aufgabenblätter

Die Aufgabenblätter, welche von Algoria Worksheet erstellt und gelöst werden können, werden in .aws Dateien abgespeichert. Aufgabenblätter bestehen aus zwei wichtigen Komponenten:

- Allgemeine Informationen zum Aufgabenblatt. Die wichtigste Information ist die Versionsnummer der Datei. Da vor allem in der Entwicklungs- und Anfangsphase von Algoria Worksheet mehrere Änderungen an den Datenstrukturen der Aufgaben vorgenommen werden, ist eine Versionierung unabdingbar. Zusätzliche Informationen dazu folgen im Kapitel 4.2. Neben der Versionsnummer enthält das Aufgabenblatt auch einen Namen.
- Eine Liste aller enthaltenen Aufgaben.

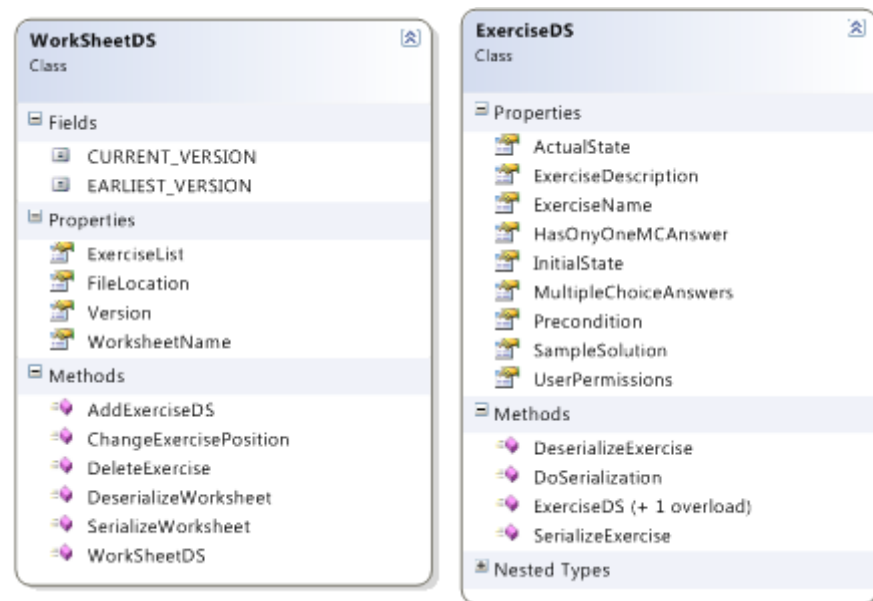


Abbildung 4.1: Datenstrukturen der Aufgabenblätter

### 4.1 Datenstruktur der Aufgaben

Ein Algoria Worksheet Aufgabenblatt enthält typischerweise mehrere Aufgaben. Jede Aufgabe besteht aus einem Namen, einer Aufgabenbeschreibung, den Algoria Zeichenflächen und einer Multiple Choice Region. Besitzt eine Aufgabe Multiple Choice Elemente, besteht die Lösung darin, die korrekten Antworten zu wählen. Im anderen Fall muss der Studierende seine Lösung in der Algoria Zeichenfläche zeichnen und sie dann vom Lösungsprüfer verifizieren lassen.

Es sind zwei weitere Eigenschaften einer Aufgabe vorgesehen, welche zur Zeit noch nicht verwendet werden. Zum einen enthält die Datenstruktur der Aufgaben eine Liste von Tupeln, die gewisse Voraussetzungen für die Verfügbarkeit des Lösungsprüfers definieren können. So könnte beispielsweise bei einer Aufgabe mit Mehrfachauswahl der Lösungsprüfer erst benutzt werden, wenn alle Antworten

entweder als falsch oder richtig markiert sind. Eine andere vorstellbare Voraussetzung wäre, dass der Lösungsprüfer für die Zeichenfläche erst nach 5 Minuten oder beim Erreichen eines bestimmten Fortschritts benutzt werden kann.

Als letzte noch nicht verwendete Eigenschaft enthält eine Aufgabe ein Byte für Benutzerrechte. Dieses Byte eröffnet Möglichkeiten, den Studierenden gewisse Aktionen zu sperren. Wie bereits in den Anwendungsszenarien beschrieben, ist es zwar unwahrscheinlich, dass Algoria Worksheet für Prüfungssituationen verwendet werden kann, dennoch kann man sich aber vorstellen, dass der Dozierende den Studierenden nicht ermöglichen will, die Musterlösung zu sehen. In einem solchen Fall wäre die Verwendung dieser Eigenschaft angemessen. Die Idee ist, dass jedes Benutzerrecht ein Bit des Benutzerrechte-Bytes beansprucht. Mit dem Bitwert kann gesteuert werden, ob ein bestimmtes Recht aktiviert oder gesperrt ist. Das Byte würde erstmal 8 verschiedene Benutzerrechte unterstützen. Falls im Verlaufe der Entwicklung von Algoria Worksheet zusätzliche benötigt werden würden, kann eine Anpassung des Datentyps vorgenommen werden.

Änderungen an der Datenstruktur der Aufgaben haben eine Erhöhung der Versionsnummer zur Folge. Im folgenden Kapitel wird das Konzept der Versionierung beschrieben.

## 4.2 Versionierung

Wie bereits erwähnt, ist die Versionierung der Datei wichtig. Sollte es Änderungen an einer der Datenstrukturen geben, können so Dateien im alten Format weiterhin gelesen und in der neuen Version wieder abgespeichert werden können. Dazu wird in der Datei als erstes die Versionsnummer abgespeichert, sodass sie beim Laden mit der zur Zeit Aktuellen verglichen werden kann.

Änderungen an den Datenstrukturen müssen in den Dateien *WorksheetDS* und *ExerciseDS* vorgenommen und zwingend dokumentiert werden. Nach einer solchen Anpassung müssen die Methoden zur Serialisierung und Deserialisierung angepasst werden. Grundsätzlich werden die Eigenschaften der neuen Versionen beim Laden einer alten Datei hinzugefügt und mit einem Standardwert versehen, welcher üblicherweise leer oder null ist. Es muss nicht möglich sein, die älteste mögliche Version zu laden, das Ziel ist lediglich, dass bei einer Aktualisierung von Algoria Worksheet die letzten Versionen der Dateien geladen werden können.

## 4.3 Serialisierung

Die Serialisierung und Deserialisierung der Daten wird von den entsprechenden Klassen selbst geregelt. Sobald das Speichern einer Datei initiiert wird, erhält *WorksheetDS* einen Formatter und einen Stream. Die Klasse kümmert sich selbst darum, ihre eigenen Variablen sowie die Versionsnummer zu serialisieren. Für jede Aufgabe im Aufgabenblatt wird die Klasse *ExerciseDS* angewiesen die Serialisierung zu übernehmen. Derselbe Ablauf wird ebenfalls für die Deserialisierung verwendet.

Durch diese Struktur wird sichergestellt, dass bei einer Anpassung der Datenstruktur jeweils nur in der entsprechenden Klasse Anpassungen gemacht werden müssen. Einzige Ausnahme bildet die Serialisierung der Algoria Zeichenflächen. Diese werden in der Klasse *SerializationHelper* behandelt, da jede in Algoria verwendete Datenstruktur ihren eigenen Serialisierer besitzt.

## 5 Programm

Algoria Worksheet ist eine Anwendung, die auf Windows Presentation Foundation, kurz WPF, basiert. Bereits bestehende WPF-Komponenten sind in Algoria Worksheet integriert, weshalb gewisse Designvoraussetzungen an die Applikation gegeben sind.

Bei der Entwicklung ist darauf geachtet worden, dass die Applikation einfach erweiterbar und wartbar ist. Dies erleichtert es Entwicklern, welche die Software erweitern und anpassen werden, ihre Arbeit zu machen. Deshalb verzichtet Algoria Worksheet so gut wie möglich auf Komponenten von Drittanbietern. Solche Quellen können zu potentiellen Problemen führen, welche nicht mit vernünftigem Aufwand zu lösen wären. Kennt der Entwickler die verwendete Komponente nicht, steigt der Arbeitsaufwand für das Lösen eines Problems zusätzlich an. Ausnahmen werden in diesem Kapitel aufgeführt und beschrieben.

In den folgenden Abschnitten werden Applikationsstruktur, Designentscheidungen, Zusammenarbeit und Funktionsweise einzelner Komponenten von Algoria Worksheet detailliert beschrieben.

## 5.1 Applikationsdesign

Dieses Kapitel beschreibt die zu Grunde liegenden Designentscheidungen, sowie deren Umsetzung. Algoria Worksheet wird von zwei Hauptbenutzergruppen verwendet, welche unterschiedliche Bedienmöglichkeiten besitzen. Das integrierte Algoria benutzt Prism in Zusammenarbeit mit MEF. Algoria Worksheet muss also damit zurechtkommen. Diese und weitere Gegebenheiten führen zu den getroffenen Designentscheidungen, welche in den folgenden Abschnitten beschrieben werden.

### 5.1.1 Entwurfsmuster MVVM

Algoria Worksheet ist nach dem Model-View-Viewmodel Muster entworfen. Die konkrete Umsetzung entspricht allerdings nicht exakt der Definition von MVVM. Eine schematische Darstellung von Algoria Worksheet sieht folgendermassen aus:

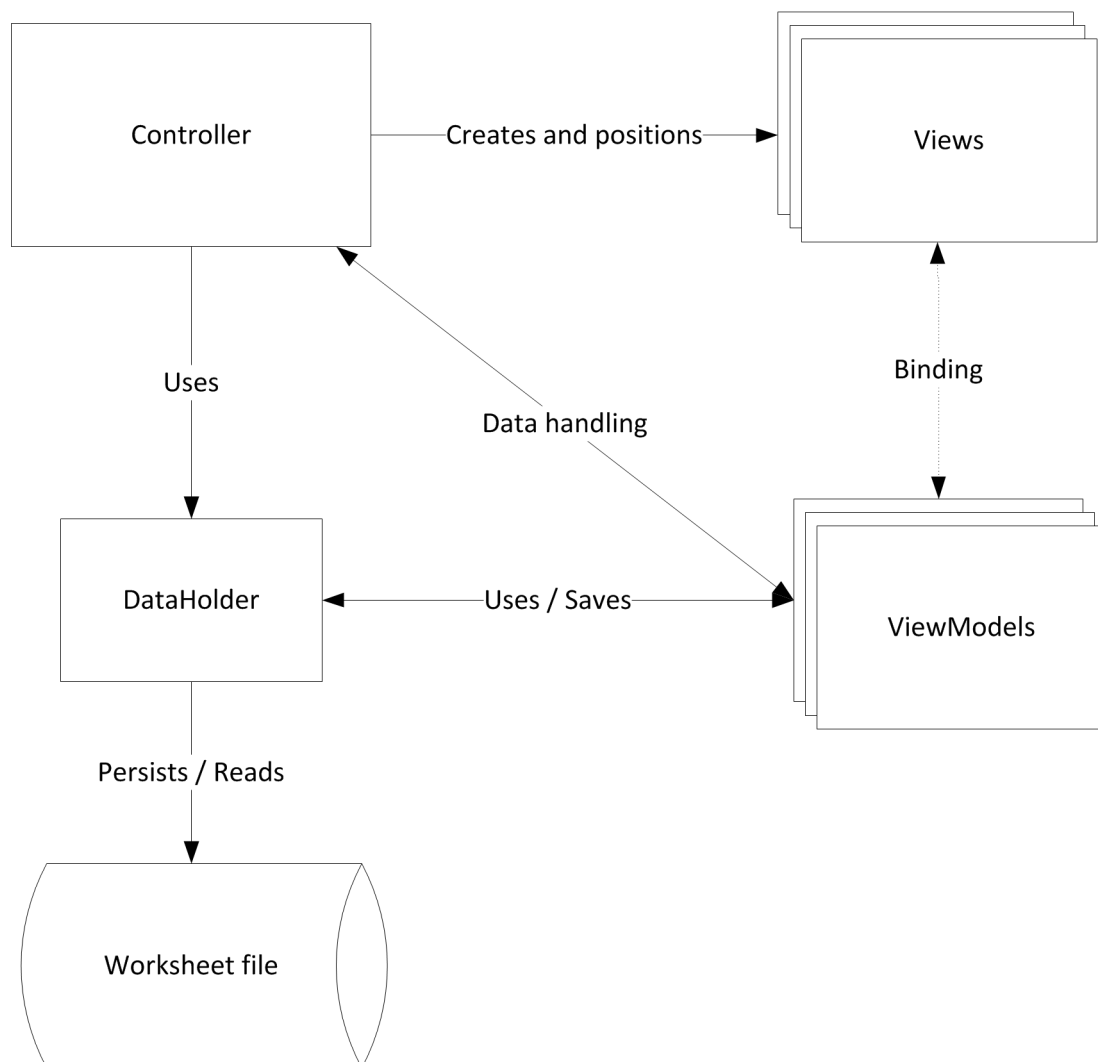


Abbildung 5.1: MVVM Design

Der *DataHolder*, welcher vom Controller benutzt wird, enthält die Daten der geöffneten Datei. Genauere Informationen zu der Datenstruktur der Worksheet-Dateien sind im Kapitel 4 zu finden. Zusätzlich zu den Daten der Aufgaben enthält er auch Referenzen auf Algoria Engines. Grund dafür ist, dass

es eine Algoria Engine pro Zeichenfläche jeder Aufgabe gibt. Eine Aufgabe enthält bis zu maximal drei Zeichenflächen. Diese Zuweisung muss auf dieser Ebene geschehen, da die Algoria Engines beim Erzeugen der Views angegeben werden müssen. Der Controller erzeugt mit den Algoria Engines die Views und positioniert diese an der richtigen Stelle, während die restlichen Daten an die ViewModels übergeben werden. Somit wird dem Controller die Aufgabe des Erzeugens, Positionierens und Entfernens von Views zugeteilt.

Es wird für jede Komponente ein ViewModel erzeugt, welches die zu darstellenden Daten enthält. Views werden auf ihr zugehöriges ViewModel gebunden, wodurch einerseits immer die Daten aus dem ViewModel dargestellt werden und andererseits das Bearbeiten von Werten ermöglicht wird. Denn wenn ein Wert vom Benutzer in der View geändert wird, wird der Wert im ViewModel aufgrund des Bindings direkt auf den neuen Wert gesetzt.

Dieser Aufbau ermöglicht es, jede beliebige View zu laden und diese auf ein ViewModel zu binden. Daher ist das Austauschen und Ändern von Views mit vergleichsweise kleinem Aufwand möglich.

### 5.1.2 Instanzen von Algoria

Im vorhergehenden Kapitel wurde erwähnt, dass es für jede Aufgabe maximal drei Algoria Instanzen gibt. Für die Studierendenversion wird nur eine Instanz benötigt, während der Dozierende eine für die Musterlösung, eine für die Ausgangslage und eine für die Studentenlösung benötigt.

Es wäre möglich, für die gesamte Applikation nur eine einzige Instanz von Algoria zu verwenden. Dazu müssten die Inhalte bei jedem Wechsel kopiert und ausgetauscht werden. Dies ist aber nur mit bereits erkannten Objekten einfach möglich, beispielsweise Arrays oder Listen. Das Austauschen von Inhalten bei denen es sich nicht um eine Datenstruktur handelt, wäre aufwändig und wesentlich schwieriger, als für jede Zeichenfläche eine eigene Instanz zu erzeugen.

In einer Vorabklärung ging es darum, ob diese Möglichkeit aufgrund der Speicherauslastung und Ladezeit überhaupt in Betracht gezogen werden kann. Es hat sich gezeigt, dass mehrere Instanzen keinen massiven Einfluss auf die Speicherauslastung haben. Folgendes Diagramm zeigt, dass die Speicherauslastung bei 30 Instanzen zwar höher ist, aber sich durchaus in einem akzeptablen Bereich befindet:

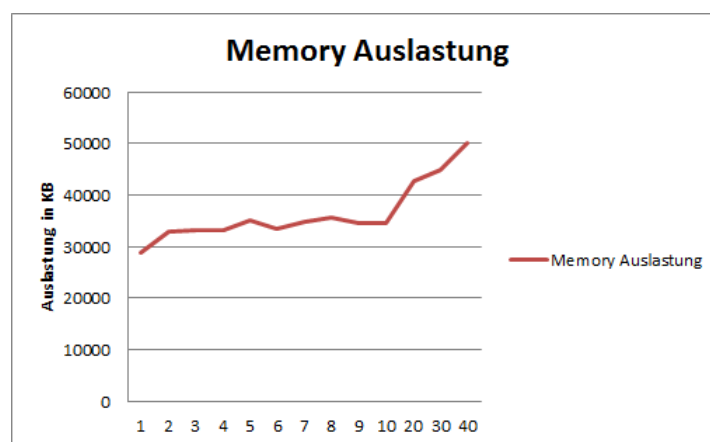


Abbildung 5.2: Speicherauslastung Algoria Instanzen

Ein Vorteil, dass für jede Zeichenfläche eine eigene Instanz verwendet wird, ist, dass zu jeder Zeit auf eine beliebige Zeichenfläche zugegriffen werden kann. Für die Seitenanzeige wurde diese Möglichkeit ausgenutzt, indem die Inhalte über einen *VisualBrush* direkt auf die Anzeigeelemente projiziert werden. Details zur Umsetzung sind in Kapitel 5.4 zu finden.

### 5.1.3 Benutzerunterscheidung

Algoria Worksheet ist für zwei unterschiedliche Benutzergruppen verfügbar und läuft unabhängig von einem Authentifizierungsserver. Aus diesem Grund werden zwei unterschiedliche Versionen des Programmes angeboten. Es ist dadurch möglich, Algoria Worksheet auch offline zu verwenden. Dies ist die einfachste Möglichkeit beiden Benutzergruppen spezifisch jene Funktionalität zu bieten, die sie benötigen, ohne dass der Benutzer durch Passwordeingabe oder Applikationseinstellungen belästigt wird.

Um diese zwei Versionen zu erzeugen, gibt es zwei verschiedene Builds. Ein Flag definiert die zu erstellende Version. Wird Algoria Worksheet kompiliert, wird abhängig von diesem Flag eine Variable gesetzt:

---

```
1 // Versionflag
2 #if ADMIN
3     public static byte VERSION = ADMIN_VERISON;
4 #else
5     public static byte VERSION = STUDENT_VERISON;
6 #endif
```

---

Codeausschnitt 5.1: Version flag

Entscheidende Stellen im Code werden dann nicht mit Präprozessordirektiven<sup>[4]</sup> unterschieden, sondern beziehen sich dann auf diese *VERSION*-Variable:

---

```
1 if (VERSION == ADMIN_VERISON)
2 {
3     return new SidepanelAdminView();
4 }
5 else if (VERSION == STUDENT_VERISON)
6 {
7     return new SidepanelView();
8 }
```

---

Codeausschnitt 5.2: Abfrage der VERSION-Variable

Damit diese essentiellen Unterscheidungen nicht in allen verschiedenen Klassen gesucht werden müssen, sind sie zusammen mit der Definition der *VERSION*-Variable in ihrer eigenen Hilfsklasse *VersionController.cs*. Wenn nun eine bestimmte Komponente angefordert wird, erhält der Aufrufer von der Hilfsklasse eine entsprechende Instanz der Komponente. Dieses Konzept hat zur Folge, dass es für die meisten Komponenten zwei verschiedene Implementationen gibt, eine für die Dozierendenversion und eine für die Studierendenversion. Da es sich dabei aber hauptsächlich um Views handelt, ist dies kein Problem, da die verschiedenen Versionen ohnehin andere Benutzeroberflächen besitzen.

Manche Views besitzen öffentliche Methoden, welche vom ViewModel aufgerufen werden. Da es mehrere Views für dieselben ViewModels gibt, muss eine Schnittstellendefinition vorhanden sein. Daher

gibt es pro ViewModel ein Interface, welches von den Views, die sich auf jenes ViewModel registrieren, implementiert werden muss.

### 5.1.4 Prism

Prism ist ein Leitfaden von Microsoft, welcher dabei helfen soll, gewisse Entwurfsmuster umzusetzen. Es bietet beispielsweise einige Vereinfachungen für die Umsetzung eines MVVM Musters. Bestehende Module wie Algoria verwenden Prism zur Umsetzung ihres Model-View-ViewModel Musters, daher wurde Prism in Algoria Worksheet ebenfalls eingebunden. Allerdings beschränkt sich der Einsatz nur auf die Zusammenarbeit mit den externen Komponenten. Algoria Worksheet verzichtet auf den Einsatz von Prism innerhalb den eigenen Komponenten, um die Komplexität und die Anzahl fremder Komponenten gering zu halten.

### 5.1.5 AvalonDock

AvalonDock ist eine Open Source Komponente eines Drittanbieters, welche es ermöglicht, dynamisch positionierbare Elemente zu erzeugen. So kann beispielsweise eine Algoria Zeichenfläche aus der Verankerung gezogen werden, indem mit der Maus der Reiter herausgezogen wird. Dadurch entsteht ein eigenes Fenster, welches dann an von AvalonDock definierte Regionen wieder andockt werden kann. Mit AvalonDock ist man in der Lage, viele der Funktionen, wie sie auch VisualStudio besitzt (automatisch ausblendende Seitenleiste), nachzubilden.

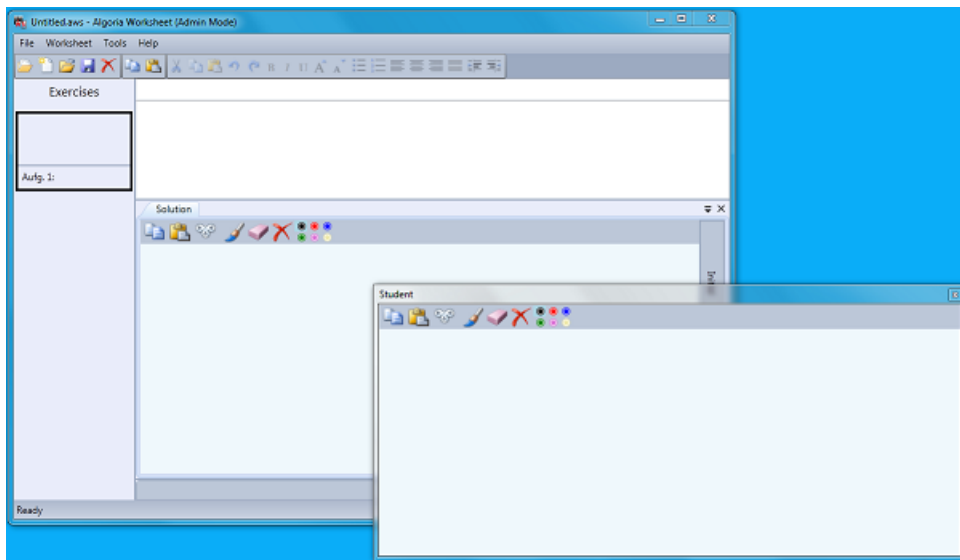


Abbildung 5.3: AvalonDock

### 5.1.6 Klassendiagramm

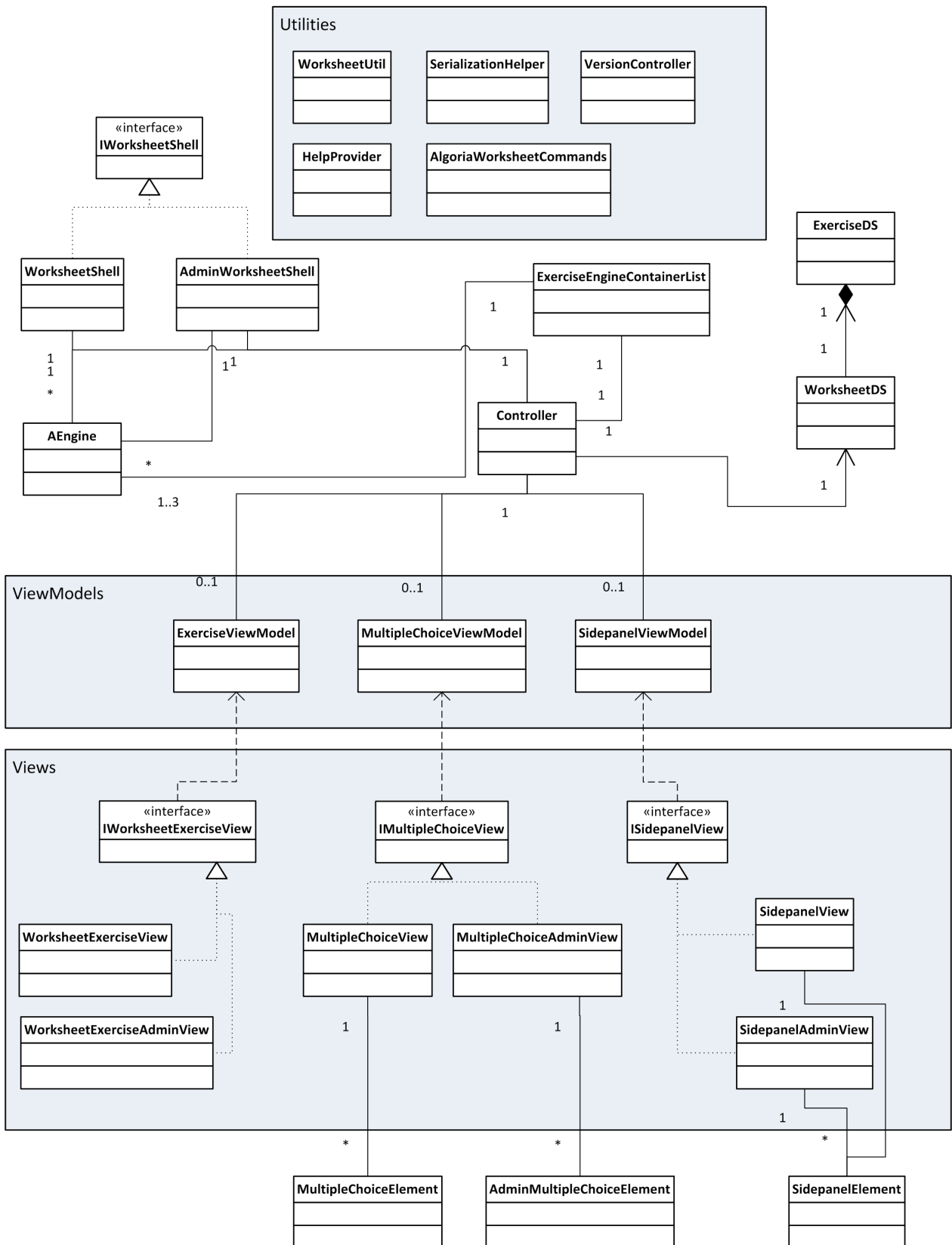


Abbildung 5.4: Klassendiagramm

### 5.1.7 Assertions

Für die Applikation gibt es zur Zeit keine Unittests. Die Funktionen wurden jeweils manuell getestet. Um aber bei einem Auftreten eines Fehlers eine möglichst schnelle Lokalisierung zu ermöglichen, enthält der Programmcode Assertions. Diese werden innerhalb von Methoden als erste Zeile ausgeführt und überprüfen die Vorbedingung für den Aufruf der Methode, sodass die Methode ein korrektes Resultat liefern kann. Wird die Vorbedingung nicht erfüllt, schlägt die Assertion fehl und es kann zurückverfolgt werden, woher diese Methode mit falschem Parameter aufgerufen wurde.

## 5.2 Algoria

Algoria übernimmt die Funktionen zum Zeichnen und Verwenden von Datenstrukturen. Diese Komponente wurde in einem vorgängigen Projekt entwickelt und ist daher nicht Teil dieser Arbeit. Algoria bietet die Möglichkeit, in einem definierten Zeichnungsbereich, Datenstrukturen zu zeichnen und deren Werte zu setzen. Erkennt das Programm die gezeichneten Formen, werden automatisch die entsprechenden Datenstrukturen und Variablen im Speicher angelegt. Über ein Kontrollelement können auf selektierten Datenstrukturen dann Algorithmen ausgeführt werden. Die Algorithmen werden mit Animationen visualisiert, damit die Studierenden ein besseres Verständnis des Ablaufes erhalten.

Algoria wurde mit diesen Funktionalitäten in Algoria Worksheet integriert. Wie bereits in Kapitel 5.1.2 beschrieben, werden mehrere Instanzen von Algoria erzeugt und verwaltet. Um das Layout möglichst gut an Algoria Worksheet anpassen zu können, wurden nicht einfach nur die Zeichnungsflächen integriert, sondern es wurde ein neues WPF-Element erstellt (*AEngine.xaml*). Dadurch kann für Algoria Worksheet eine möglichst grosse Zeichenfläche, ohne Seitenleisten oder Menüanzeigen von Algoria, garantiert werden.

Die Kontrollelemente für die Datenstrukturen wurden ebenfalls angepasst, damit sie möglichst wenig Platz wegnehmen. Dazu werden sie in der momentanen Version neben der dazugehörigen Datenstruktur platziert. Es wäre vorstellbar, diese in einer kommenden Version nur unter bestimmten Bedingungen anzuzeigen, beispielsweise beim Darüberfahren mit der Maus.

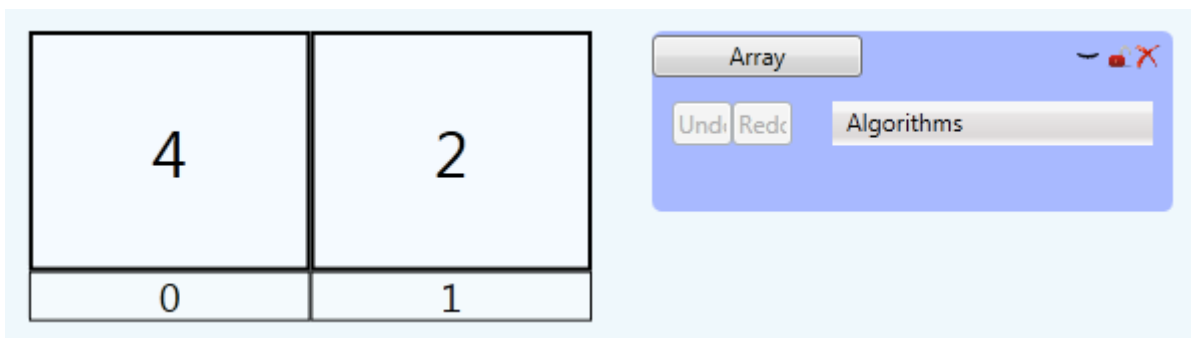


Abbildung 5.5: Algoria Datenstruktur und Kontrollelement

## 5.3 Aufgabenstellung

Die Aufgabenstellung wird immer oberhalb der Algoria Zeichenflächen dargestellt. Sie enthält, wie der Name schon sagt, die vom Dozierenden definierte Aufgabenstellung. Für die verschiedenen Versionen werden unterschiedliche WPF-UserControls verwendet. Die Dozierendenansicht verwendet eine *RichTextBox*, mit welcher der Text ähnlich wie in Microsoft Word bearbeitet werden kann. In der Studierendenversion wird ein *FlowDocumentViewer* verwendet. Dieser kann nur den formatierten Text darstellen, ihn aber nicht bearbeiten. Dieses UserControl bietet aber andere Vorteile, die im Kapitel 5.3.2 erläutert werden.

Die Aufgabenstellung ist in den *WorksheetExerciseViews* integriert. Diese Views sind an das *ExerciseViewModel* gebunden, welches nebst der Aufgabenstellung auch den Aufgabentitel enthält. Die *WorksheetExerciseAdminView* ist die View für die Dozierendenansicht, welche die *RichTextBox* verwendet.

### 5.3.1 RichTextBox

Die *RichTextBox* ist ein Eingabefeld, welches erweiterte Funktionalitäten zur Text- und Absatzformatierung bietet. So ist es möglich, mit der *RichTextBox*...

- ... die Textgröße anzupassen.
- ... die Ausrichtung eines Abschnittes zu steuern.
- ... den Textstil auf kursiv, fett und/oder unterstrichen zu ändern.
- ... eine Aufzählung mit Punkten oder Nummerierung zu erzeugen.
- ... den Einzug des Abschnittes zu steuern.
- ... einzelne Bearbeitungsschritte rückgängig zu machen.
- ... eine Auswahl zu kopieren und einzufügen.

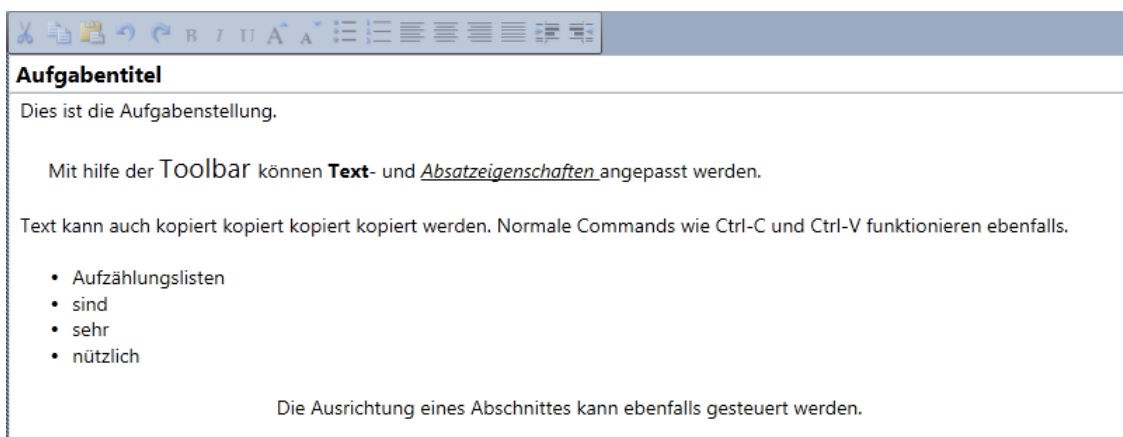


Abbildung 5.6: Beispiel *RichTextBox*

Änderungen am Inhalt werden im ViewModel gespeichert, sobald die *RichTextBox* den Fokus verliert, oder vom Controller die Anweisung an das ViewModel gegeben wurde, die aktuellsten Daten zurückzuliefern. Dies ist nötig, da unter gewissen Umständen der *LostFocus*-Event nicht ausgelöst wird (beispielsweise *Ctrl-S* um zu speichern). Die Daten werden im Property *RichTextDocument*, welches den Typ *FlowDocument* hat, gespeichert. Ein *FlowDocument* kann sowohl von einer *RichTextbox* als auch von einem *FlowDocumentViewer* geladen werden. Daher braucht es keine Anpassung, um für

die Studierendenansicht die Daten in den *FlowDocumentViewer* zu laden.

Beim Öffnen eines bestehenden Arbeitsblattes muss der gespeichert Inhalt aus dem gespeicherten Xaml-String wiederhergestellt werden. Dazu muss ein leeres *FlowDocument* erstellt werden, in welches der Text geladen werden kann.

```

1 FlowDocument doc = new FlowDocument();
2 TextRange range = new TextRange(doc.ContentStart, doc.ContentEnd);
3 range.Load(new MemoryStream(Encoding.UTF8.GetBytes(exerciseText)),
4   DataFormats.Xaml);
5 RichTextDocument = doc;

```

Codeausschnitt 5.3: RichTextBox Inhalt laden

Dieser Code erstellt ein leeres *FlowDocument* und definiert anschliessend einen Bereich (*TextRange*) um den Text hineinzuladen. *Load* ist eine Methode der Klasse *TextRange*, welche einen *MemoryStream* erwartet, sowie das Datenformat des Streams. Mit diesen Informationen deserialisiert die Methode den Stream und überschreibt den Text im definierten Bereich mit den Daten aus dem Stream. Danach wird das Dokument-Property der *RichTextBox* auf *doc* gesetzt, um die Daten in der *RichTextbox* darzustellen.

### 5.3.2 FlowDocumentViewer

Der *FlowDocumentViewer* wird zur Darstellung der Aufgabenstellung in der Studierendenversion verwendet. Genau wie die *RichTextBox* kann auch dem *FlowDocumentViewer* ein *FlowDocument* angegeben werden, welches er dann anzeigt. Die Komponente stellt gewisse Zusatzfunktionalitäten zur Verfügung, die der Benutzer nicht hätte, wenn der Text als normales *RichText*-Element dargestellt würde. Standardmässig integriert sind die folgenden Funktionen:

- Seitennavigation: Das *UserControl* unterteilt einen Text, der grösser als die Anzeigefläche ist, automatisch in Seiten.
- Automatische Spaltendarstellung
- Einstellungen zur Textgrösse
- Suche im *FlowDocument* (Ctrl-F)

Das Layout und die Controls können bei bedarf beliebig angepasst werden, indem ein eigener *Style* in einem *ResourceDictionary* erstellt wird. Weitere Informationen zum Controll sind auf der [MSDN](#) zu finden.

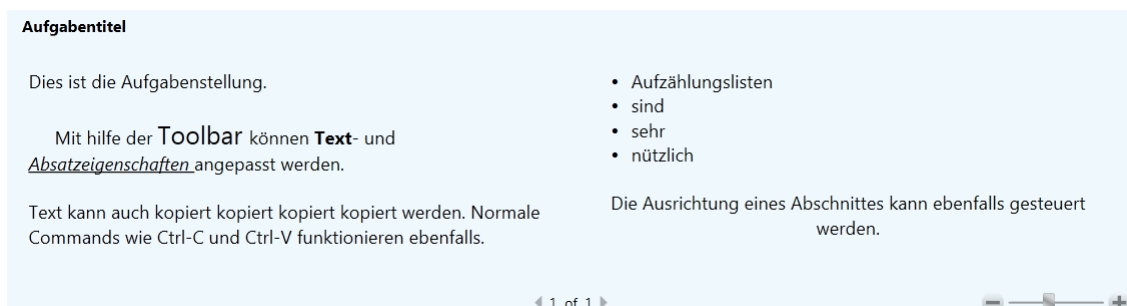


Abbildung 5.7: Beispiel *FlowDocumentViewer*

## 5.4 Seitenleiste

Die Seitenleiste ist eine Übersicht über alle Aufgaben der geöffneten Datei. In Anlehnung an Microsoft PowerPoint enthält die Seitenleiste kleine Vorsichten der Zeichenflächen, wodurch der Benutzer eine Aufgabe daran wiedererkennen kann. Um dem Benutzer mitzuteilen, dass einige Aufgaben durch Ankreuzen von richtigen Antworten zu lösen sind, erhalten diese Aufgaben ein zusätzliches Symbol, welches eine Mehrfachauswahl darstellt. Die Seitenleiste bietet nur eingeschränkte Funktionalität für die Studierenden. Dozierende haben die Möglichkeit, die Aufgabenreihenfolge in der Seitenleiste mittels Drag & Drop zu verändern. Des Weiteren können sie neue Elemente hinzufügen und Bestehende löschen. Die Seitenleiste ergänzt den Aufgabentitel mit einer automatischen Nummerierung, um dem Benutzer eine bessere Orientierung zu bieten, falls die Seitenleiste zu lange ist und daher einen Scrollbalken besitzt. Diese Nummerierung kann über die Anwendungsoptionen konfiguriert werden.

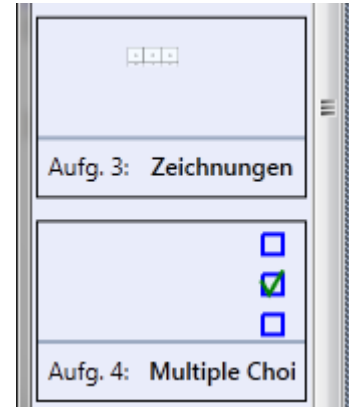


Abbildung 5.8: Vorschau in der Seitenleiste

### 5.4.1 Aufbau

Wie auch andere Komponenten von Algoria Worksheet besitzt auch die Seitenleiste zwei verschiedene Versionen für die Studierenden und die Dozierenden. Es gibt dazu eine View für die Dozierendenversion und eine zweite View für die Studierenden. Beide Views verwenden dasselbe ViewModel. Der Unterschied liegt darin, dass die View der Dozierenden ein Kontextmenu und eine Drag & Drop Funktionalität offeriert. Das Kontextmenu enthält Funktionen zum Löschen oder Hinzufügen von Aufgaben.

Zur Darstellung der Aufgaben verwenden beide Views ein gemeinsames UserControl, das *Sidepanel-Element*. Auf diese Weise wird darauf geachtet, dass so wenig unterschiedliche Implementationen wie möglich für die zwei Benutzertypen nötig sind. Details zur Drag & Drop Funktionalität können in Kapitel 5.8 gefunden werden.

### 5.4.2 Bildvorschau

Die Vorschau der Zeichenregionen wird mit einem VisualBrush dargestellt. Bei der Erzeugung der Algoria Zeichenfläche wird ein zweiter VisualBrush erzeugt, welcher den aktuellen Inhalt anzeigen kann. Dieser wird dann an die Vorschaulemente übergeben und dort als Hintergrund gesetzt.

```
1 VisualBrush cloneView = new VisualBrush(DrawingRegion);
```

Codeausschnitt 5.4: Vorschau VisualBrush

Aufgrund der Grösse der Zeichenfläche von Algoria muss ein manueller Viewport angegeben werden. Es wird von links oben ein Rechteck ausgewählt, welches dann in skaliertes Form in der Seitenleiste angezeigt wird.

## 5.5 Multiple Choice

Nebst dem Stellen von Zeichenaufgaben besteht in Algoria Worksheet auch die Möglichkeit Multiplechoicefragen zu erstellen. Eine Multiplechoicefrage kann in zwei verschiedenen Formen gestellt werden. Zum Einen kann zugelassen werden, mehrere Antworten als richtig anzusehen, zum Anderen ist es auch möglich, genau eine Antwort als die Korrekte zu definieren. Eine View für Multiplechoicefragen bietet dementsprechend einem Dozierenden die Möglichkeit zwischen diesen zwei Modi umzuschalten. Das Hinzufügen und Definieren von neuen und das Löschen von bestehenden Antworten wird ebenso angeboten, wie das Editieren von bereits bestehenden. Zu einer Antwortmöglichkeit gehört nebst dem Antworttext auch die Information, ob diese Antwort als korrekt gilt oder nicht. Es ist möglich die Reihenfolge der Antworten selbst zu definieren und nachträglich zu ändern. Die Funktionalität für Studierende beschränkt sich hingegen nur auf das Markieren von Fragen, welche als richtig angenommen werden.

### 5.5.1 Aufbau

Analog zu anderen Komponenten in Algoria Worksheet ist es auch bei Multiplechoicefragen unerlässlich, verschiedene Versionen für Studierende und Dozierende anzubieten. Es ist für jede Version eine eigenständige View vorhanden, wobei beide Views mit demselben Viewmodel arbeiten. Die Unterschiede in den Views liegt in deren Funktionalität und den Usercontrols, welche in der View für das Darstellen von Antworten verwendet werden.

Die einzelnen Views und deren Usercontrols werden im Folgenden genauer beschrieben. Im Gegensatz zum Sidepanel wurden hier für jede Version eigene Usercontrols geschrieben und nicht dieselben zweimal verwendet. Eine grundsätzliche Trennung ist nötig, da die Funktionalität hier nicht nur in der View selbst liegt, sondern auch in den Usercontrols selbst, z.B. im Verschieben einer Antwort oder dem Markieren in Dozierenden- und Studierendenansicht.

### 5.5.2 Dozierendenansicht

In der Dozierendenansicht wird eine *MultipleChoiceAdminView* verwendet. Diese View bietet nicht nur die Möglichkeit zwischen dem Aufgabentyp (eine richtige Antwort / mehrere richtige Antworten) zu wählen und eine neue Antwort hinzuzufügen, sondern beinhaltet auch die Antworten selbst. Jede einzelne Antwort wird mittels einem *AdminMultipleChoiceElement*-Usercontrol repräsentiert.

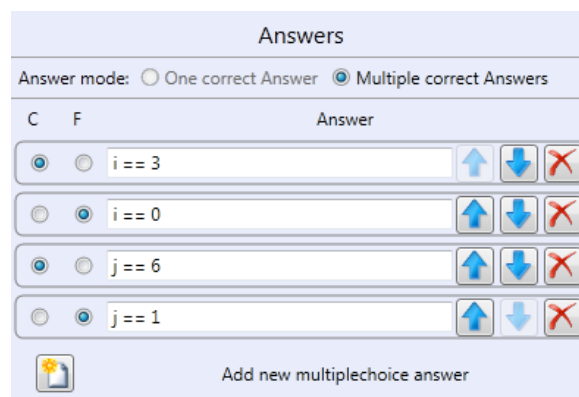


Abbildung 5.9: *MultipleChoiceAdminView* mit mehreren korrekten Antworten

In einem *AdminMultipleChoiceElement* können Dozierende den Antworttext ändern, die Frage löschen oder deren Position verändern. Auch die Angabe, welche Antworten als korrekt angesehen werden, kann in diesem UserControl getätigt werden. Ist eine Aufgabe so definiert, dass nur eine korrekte Antwort gegeben werden kann, werden beim Setzen einer Antwort als die Richtige, alle anderen Antworten automatisch als falsch gesetzt. Sollten mehrere Antworten als richtig gelten, können beliebig viele ausgewählt werden.

Damit ein Benutzer beim Verschieben eines *AdminMultipleChoiceElement* dieselbe Aktion mehrmals ausführen kann (z.B. wenn das unterste Element an die höchste Position verschoben werden soll), wird der Mauszeiger automatisch so verschoben, dass ein weiterer Klick die Aktion wiederholt. Da das Neupositionieren des Mauszeigers einen DLL-Aufruf benötigt, wurde dieser in der *MultipleChoiceAdminView* Klasse mit folgenden Code-Zeilen eingebunden.

```

1 [System.Runtime.InteropServices.DllImportAttribute("user32.dll",
   EntryPoint = "SetCursorPos")]
2 [return: System.Runtime.InteropServices.MarshalAsAttribute(System.
   Runtime.InteropServices.UnmanagedType.Bool)]
3 public static extern bool SetCursorPos(int X, int Y);

```

Codeausschnitt 5.5: Einfügen von Checkbox / Radiobutton im *MultipleChoiceElement*

Nach dem Auslösen des des *MoveUp* bzw. *MoveDown* Events wird die Mauszeigerposition neu gesetzt.

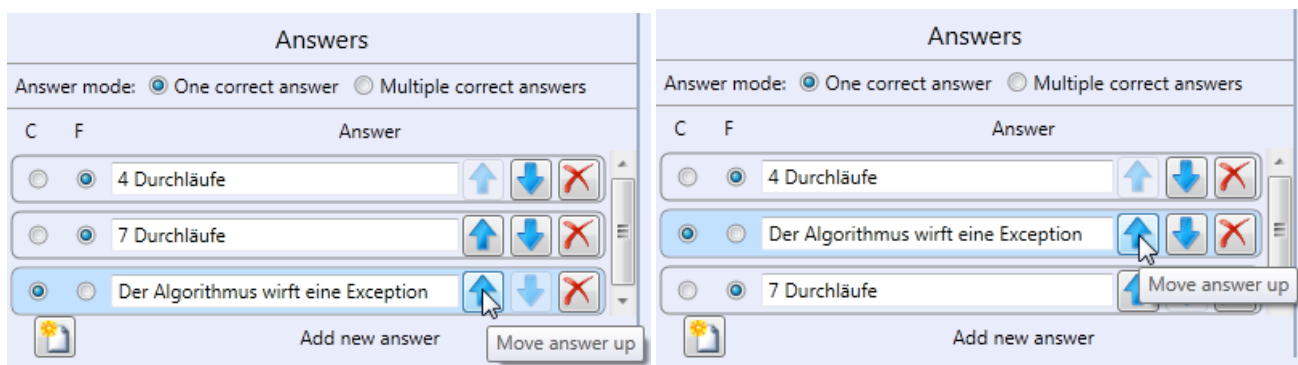


Abbildung 5.10: Mauszeigerposition vor und nach dem Klick der *MoveUp*-Schaltfläche

### 5.5.3 Studierendenansicht

Für das Darstellen von Multiplechoiceaufgaben in der Studierendenansicht wird die *MultipleChoiceView* verwendet. Funktionalitäten werden in dieser View keine angeboten, sondern sie enthält *MultipleChoiceElement*-Usercontrols, welche die einzelnen Antworten widerspiegeln.

Abbildung 5.11: *MultipleChoiceView* mit mehreren korrekten Antworten

Abbildung 5.12: *MultipleChoiceView* mit einer korrekten Antwort

Wenn in einer Multiplechoicefrage nur eine Antwort als die Korrekte markiert werden kann, enthält ein *MultipleChoiceElement* einen Radiobutton für die Auswahl der richtigen Antwort. Falls es möglich ist, mehrere Antworten als richtig anzuwählen, wird anstelle des Radiobuttons eine Checkbox verwendet. Das Usercontrol enthält ein *Stackpanel*, in welchem dann die Checkbox oder der Radiobutton platziert wird.

```

1  /*
2   * If only one correct answer is allowed this element contains a radio
3   * otherwise checkboxes will be used.
4   */
5  if (onlyOneCorrect)
6  {
7      RadioButton radioButton = new RadioButton();
8      radioButton.GroupName = "CorrectGroup";
9      ...
10     CheckBoxRadioButton.Children.Add(radioButton);
11 }
12 else
13 {
14     CheckBox box = new CheckBox();
15     ...
16     CheckBoxRadioButton.Children.Add(box);
17 }

```

Codeausschnitt 5.6: Einfügen von Checkbox / Radiobutton im *MultipleChoiceElement*

Diese Codezeilen stammen aus dem Konstruktor der *MultipleChoiceElement*-Klasse. Für Radiobuttons ist es zusätzlich notwendig, die Gruppen zu setzen. So wird sichergestellt, dass Studierende nur eine Antwort anwählen können.

## 5.6 Events

Das Konzept der *Events* in C# gibt einer Klasse die Möglichkeit, registrierte Klienten (z.B. eine zweite Klasse) zu informieren, wenn etwas Wichtiges passiert. Die häufigste Anwendung finden *Events* bei graphischen Oberflächen. Wird eine Schaltfläche betätigt, wird die dafür registrierte Methode ausgeführt. Das *PropertyChanged Event* wird für Benutzeroberflächen ebenfalls sehr häufig verwendet. Ein solches *Event* wird in dem Moment ausgelöst, sobald der Wert eines *Properties* geändert wurde. Dies kann verwendet werden, Daten im Viewmodel auf den neusten Stand zu bringen (wenn ein Benutzer in einer Eingabemaske etwas eingetippt hat) oder die graphische Oberfläche mit den neusten Daten abzufüllen (wenn sich die Daten im Hintergrund geändert haben). *Events* können aber auch in Gebieten zur Anwendung kommen, welche nichts mit Benutzeroberflächen zu tun haben. Ein Beispiel für ein solches Einsatzgebiet ist Threading. Ein Thread kann, sobald die vorgesehene Arbeit vollendet ist, mittels *Events* die Klasse informieren, welche den Thread gestartet hat.

Für das Deklarieren von *Events* werden *Delegates* verwendet. Eine Einführung in *Delegates* ist auf der MSDN-Webseite<sup>1</sup> zu finden. Die Implementierung wird im folgenden Kapitel anhand eines Beispiels aus Algoria Worksheet beschrieben.

### 5.6.1 Events in Algoria Worksheet

Für das Beschreiben von *Events* wird die *AdminMultipleChoiceElement* Klasse als Anbieter von *Events* verwendet. Auf diese Ereignisse registriert sich die Klasse *MultipleChoiceAdminView*. Wie in Abbildung 5.13 zu sehen, bietet ein *AdminMultipleChoiceElement* mehr als ein *Event* an. Die *MultipleChoiceAdminView* kann einer beliebigen Anzahl von *AdminMultipleChoiceElementen* Platz bieten. Ein solches Element bietet einem Benutzer unter anderem die Funktion, es zu löschen. Da die Elemente möglichst unabhängig von der View sein sollen, haben sie keine Referenz darauf und können ein Löschen im Viewmodel veranlassen. Das Einsetzen von *Events* bietet sich daher an. Mit *Events* kann sich die View auf solche Ereignisse in den Elementen registrieren und darauf reagieren. Im Folgenden wird hier auf das Ereignis und dessen Implementierung eingegangen, welches beim Klicken des Löschkopfs ausgelöst wird.

Wie bereits beschrieben, verwenden *Events Delegates*. Dieses *Delegate* definiert die Form der Methoden (hier muss die Methode zwingend ein *AdminMultipleChoiceElement*-Objekt annehmen), welche sich auf das Ereignis registrieren können. Danach wird der *Event* selbst deklariert.

```
1 public delegate void DeleteHandler(AdminMultipleChoiceElement sender);  
2 public event DeleteHandler DeleteEvent;
```

Codeausschnitt 5.7: Deklarieren des Lösch-*Events* und *Delegates*

Beim Drücken des Löschknopfes auf einem *AdminMultipleChoiceElement* kann dieses Ereignis ausgelöst werden. Als Argument übergibt das Element eine Referenz auf sich selbst.

```
1 DeleteHandler deleteHandler = DeleteEvent;  
2 if (null != deleteHandler)  
3 {  
4     deleteHandler(this);  
5 }
```

<sup>1</sup><http://msdn.microsoft.com/en-us/library/aa288459%28v=vs.71%29.aspx>

Codeausschnitt 5.8: Deklarieren des Lösch-Events und Delegates

Die *MultipleChoiceAdminView* registriert eine Methode mit folgenden Zeilen.

```

1 AdminMultipleChoiceElement elem = new AdminMultipleChoiceElement(...);
2 ...
3 elem.DeleteEvent += new AdminMultipleChoiceElement.DeleteHandler(
    DeleteAnswer);

```

Codeausschnitt 5.9: Registrieren der Löschmethode auf das Löschereignis

Die Methode, welche beim Eintreten des Ereignisses ausgelöst wird, kann das übergebene Element dazu verwenden, die dazugehörigen Daten aus dem ViewModel zu löschen.

```

1 private void DeleteAnswer(AdminMultipleChoiceElement sender)
2 {
3     List<ExerciseDS.MCAnswer> answers = ViewModel.
        MultipleChoiceAnswers;
4     answers.RemoveAt(sender.Index);
5     ...
6 }

```

Codeausschnitt 5.10: Löschmethode der View

Nebst dem Löschen eines Elementes aus einer Multiplechoicefrage, sind in Algoria Worksheet weitere *Events* deklariert worden. Die folgende Abbildung zeigt, welche Klassen *Events* anbieten und welche sich auf diese Registrieren.

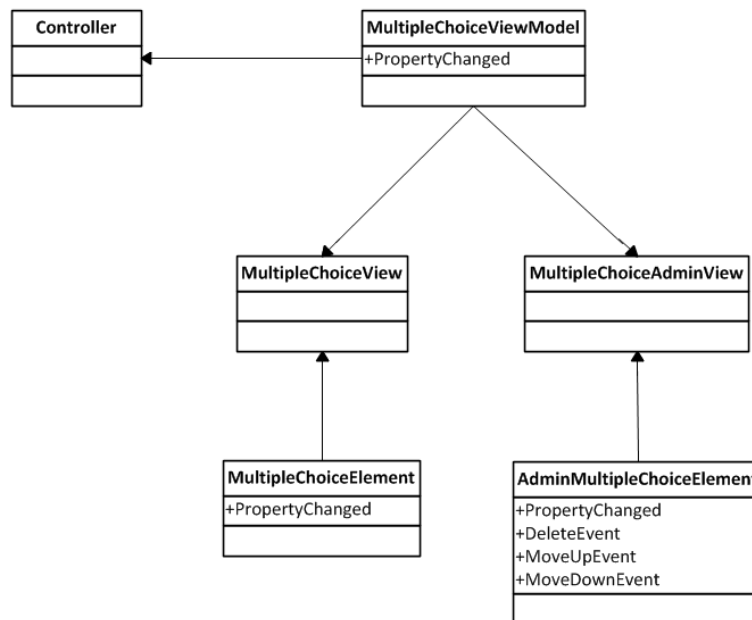


Abbildung 5.13: Diagramm mit verwendeten *Events* in Algoria Worksheet

## 5.7 Kopieren und Einfügen

Das Kopieren und Einfügen ist eine Funktionalität, welche heutzutage in praktisch jedem Programm anzutreffen ist. Auch Algoria Worksheet bietet die Möglichkeit sowohl Text (z.B. in der Aufgabenstellung) als auch Datenstrukturen, welche in Algoria gezeichnet wurden, zu kopieren und wieder einzufügen. Bevor auf die Realisierung dieser Funktion eingegangen wird, befasst sich das nächste Kapitel mit dem grundsätzlichen Konzept der Windows Zwischenablage. Die konkrete Umsetzung in Algoria Worksheet wird danach beschrieben.

### 5.7.1 Windows Zwischenablage

Unter Windows teilen sich alle Applikationen die Zwischenablage. So können Daten, welche in einem Programm kopiert und in die Zwischenablage abgelegt werden, in einem anderen Programm wieder eingefügt werden. Die Zwischenablage wird in C# durch die Klasse *Clipboard* verfügbar gemacht. Das *Clipboard* verfügt über die Möglichkeit, viele verschiedene Datenformate anzunehmen und anderen Applikationen zur Verfügung zu stellen. Jede Applikation kann die Zwischenablage durchsuchen und dabei feststellen, ob das darin gespeicherte Format verwendet werden kann.

Da die kopierende Instanz nicht weiss, in welchem Format die Daten gebraucht werden können, gibt es die Möglichkeit, dieselben Daten in verschiedenen Formaten abzulegen. Um dies zu ermöglichen, gibt es in C# die *DataObject*-Klasse. In einer Instanz dieser Klasse können mehrere Formate gespeichert werden und die Instanz dann der Zwischenablage übergeben werden.

### 5.7.2 Kopieren und Einfügen in Algoria Worksheet

In Algoria Worksheet wurde nebst dem Konzept des Kopierens und Einfügens zusätzlich die Möglichkeit realisiert, eine selektierte Auswahl auszuschneiden und an einer anderen Position wieder einzufügen. Dozierende und Studierende sind so in der Lage, Text oder in Algoria gezeichnete Datenstrukturen von einer Aufgabe zu einer anderen zu kopieren oder zu verschieben. Damit bei mehreren gezeichneten Datenstrukturen nicht automatisch alle kopiert werden, werden nur solche miteinbezogen, welche mittels den dazugehörigen Kontrollelementen selektiert wurden. Das Kopieren und Ausschneiden von Datenstrukturen wurde in der *Controller*-Klasse implementiert.

---

```
1 using (MemoryStream stream = new MemoryStream())
2 {
3     var datastructures = new HashSet<IDatastructure>();
4     if (engine.AlgoEngine.DatastructureHandles.Count > 0)
5     {
6         foreach (DatastructureHandle datastructureHandle in engine.
7             AlgoEngine.DatastructureHandles)
8         {
9             if (datastructureHandle.IsDatastructureSelected)
10            {
11                datastructures.Add(datastructureHandle.Datastructure);
12            }
13        }
14    }
```

```

14     SerializationHelper.SerializeDataStructures(datastructures, stream
        , formatter);
15     byte[] DSByteArray = stream.ToArray();
16     DataObject dataObject = new DataObject("DatastructureBA",
        DSByteArray);
17     Clipboard.SetDataObject(dataObject);
18 }

```

Codeausschnitt 5.11: Kopieren von Datenstrukturen in die Zwischenablage

Mit Hilfe der *SerializationHelper*-Klasse werden alle selektierten Datenstrukturen innerhalb der aktiven Algoria Zeichenfläche in einen *MemoryStream* gespeichert. Dieser kann direkt dem *DataObject* übergeben werden. Da die Datenstrukturen kein vordefiniertes Format besitzen, muss zusätzlich eine Kennung angegeben werden, damit beim Einfügen die richtigen Daten wieder gefunden werden können. Zum Schluss wird dann das Datenobjekt der Zwischenablage übergeben und ist fortan verfügbar.

Zusätzlich zum Kopieren ist auch das Ausschneiden eine Funktionalität, welche zum Standardreertoire vieler Applikationen gehört. Der Ablauf beim Ausschneiden ist folgendermassen: Die gewählten Datenstrukturen werden kopiert und danach aus der Algoria Zeichenfläche entfernt.

```

1 foreach (IDatastructure ds in datastructures)
2 {
3     engine.AlgoEngine.RemoveDatastructure(ds);
4 }

```

Codeausschnitt 5.12: Löschen von Datenstrukturen für das Ausschneiden

Beim Einfügen wird das Datenobjekt aus der Zwischenablage abgeholt und darin, mit Hilfe der Kennung, nach den Datenstrukturen gesucht. Sind diese vorhanden, werden sie dem *SerializationHelper* übergeben, welche die Datenstrukturen der aktiven Algoria Instanz hinzufügt.

```

1 IDataObject dataobject = Clipboard.GetDataObject();
2
3 byte[] DSByteArray = (byte[])dataobject.GetData("DatastructureBA");
4
5 if (DSByteArray != null)
6 {
7     using (MemoryStream stream = new MemoryStream(DSByteArray))
8     {
9         ...
10        var datastructures = SerializationHelper.
            DeserializeDataStructures(stream, formatter, ((AEngine)
            sender));
11        ...
12    }
13 }

```

Codeausschnitt 5.13: Einfügen von Datenstrukturen aus der Zwischenablage

Das Einfügen ist aber nicht nur in einer Algoria Worksheet Instanz möglich. Damit gezeichnete Datenstrukturen auch in anderen Programm verwendet werden können, wird beim Kopieren ein Abbild

im Bitmap Format erstellt und der Zwischenablage angefügt. Dieses Bild enthält nicht nur die selektierten, sondern alle gezeichneten Datenstrukturen der aktiven Algoria Zeichenfläche. Diese Aktion geschieht nur beim Kopieren, nicht aber beim Ausschneiden. Da andere Applikationen wie Microsoft Paint die gespeicherte Datenstruktur nicht verstehen, aber das Bitmap verwenden können, wird automatisch das Bild eingefügt.

---

```

1 RenderTargetBitmap rtb = new RenderTargetBitmap((int)WSShell.
    DockManager.ActualWidth,
2 (int)WSShell.DockManager.ActualHeight, 96d, 96d, PixelFormats.
    Default);
3 rtb.Render(engine.AlgoEngine.InkControl);
4 BmpBitmapEncoder encoder = new BmpBitmapEncoder();
5 encoder.Frames.Add(BitmapFrame.Create(rtb));
6 MemoryStream ms = new MemoryStream();
7 encoder.Save(ms);
8 System.Drawing.Image image = System.Drawing.Image.FromStream(ms);
9 dataObject.SetData(image);

```

---

Codeausschnitt 5.14: Einfügen von Datenstrukturen aus der Zwischenablage

Eine *RenderTargetBitmap* ermöglicht das Abbilden der Zeichenfläche als Bitmap. Das daraus erzeugte *System.Drawing.Image*-Bild ist in einem dem Datenobjekt bekannten Format und benötigt keine zusätzliche Kennung.

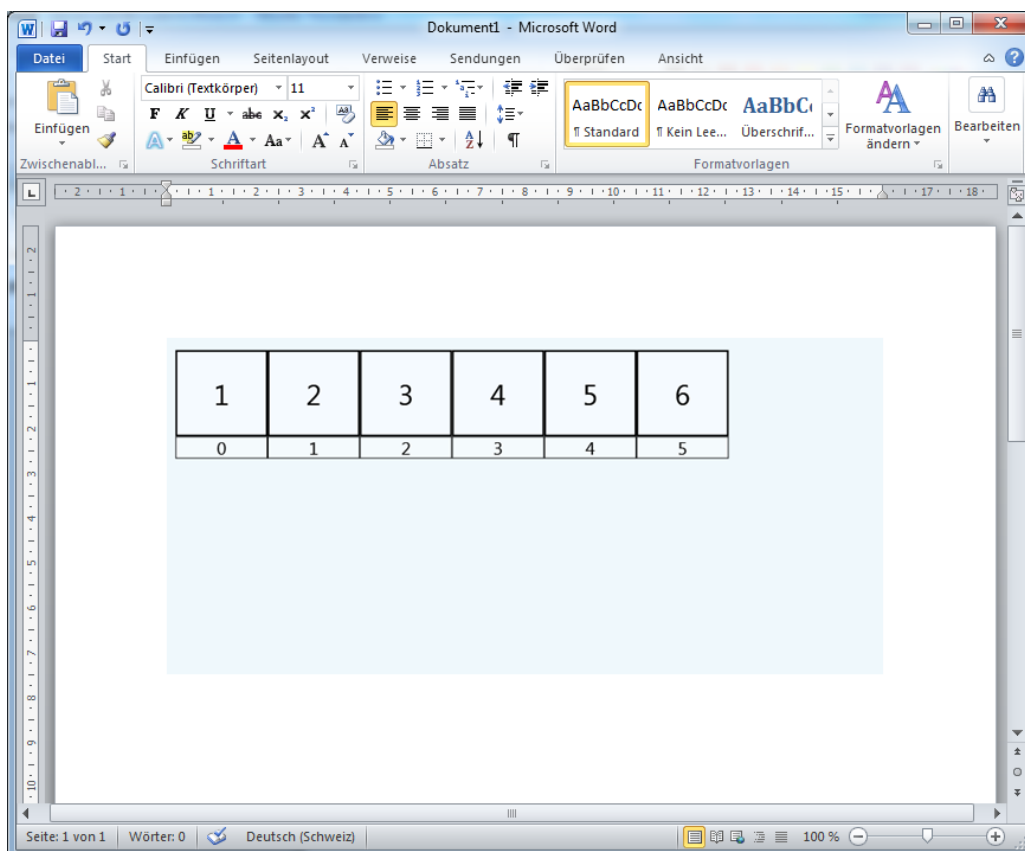


Abbildung 5.14: Eingefügte Datenstruktur in Microsoft Word

Ausschneiden, Kopieren und Einfügen sind häufig als Schaltflächen in der Symbolleiste eines Programmes zu finden. Neben dieser Möglichkeit eine der genannten Aktionen zu starten, sind Tastenkürzel eine viel verwendete Methode. Damit die standardisierten Tastenkürzel verwendet werden können, wurden *CommandBindings* verwendet. Im XAML-Dokument sieht die Definition wie folgt aus:

---

```
1 </Window.CommandBindings>
2     <CommandBinding Command="ApplicationCommands.Copy"
3         Executed="CommandBinding_ExecutedCopy"
4         CanExecute="CopyCmdCanExecute"/>
5 </Window.CommandBindings>
```

---

Codeausschnitt 5.15: Registrieren des *CommandBindings* für das Kopieren

Dieses Kommando kann nun nicht nur für Schaltflächen verwendet werden, es reagiert auch auf das Tastenkürzel CTRL-C, welches Standard für das Kopieren ist. Die für dieses Kommando registrierte Methode überprüft, ob eine Textselektion im Aufgabentext vorhanden ist. Sollte dies der Fall sein, wird der Text kopiert. Ist keine Textselektion zu finden, wird der *Controller* dazu aufgerufen, alle selektierten Datenstrukturen in der aktuellen Algoria Engine zu kopieren und der Zwischenablage zu übergeben.

## 5.8 Drag & Drop

Die Version der Seitenleiste der Dozierendenversion besitzt eine Drag & Drop Funktionalität. Dem Benutzer ist es damit möglich, die Reihenfolge der Aufgaben zu verändern. Er kann auch, wenn die Steuerungstaste gedrückt ist, eine Aufgabe an einen anderen Ort kopieren. Diese Funktionalitäten gehen über die Instanzgrenzen von Algoria Worksheet hinaus. Somit ist es möglich, eine Aufgabe aus einem Aufgabenblatt in ein anderes zu verschieben oder zu kopieren. Ebenso wie die Kopierfunktionalität, beschrieben in Kapitel 5.7, wurde dies mit einem DataObject implementiert. Eine Ergänzung, die nicht direkt mit der Funktionalität des Drag & Drop zu tun hat, ist die Animation der Seitenleistenelemente beim Verschieben einer Aufgabe. Dies wird in Kapitel 5.8.2 etwas später beschrieben.

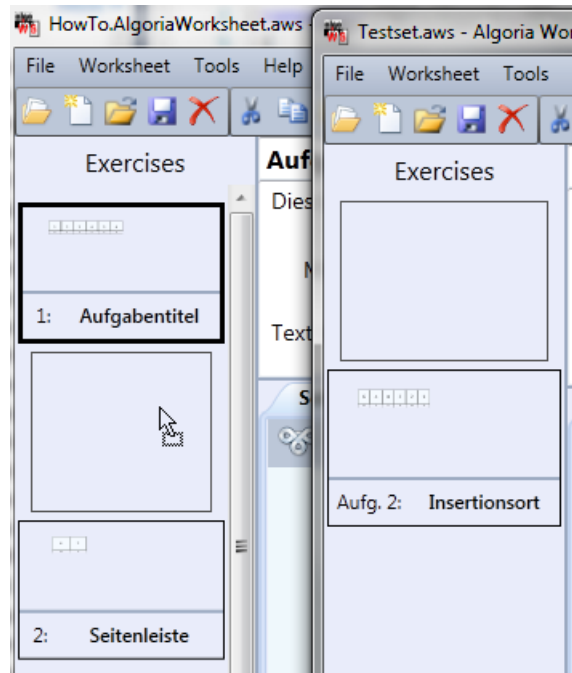


Abbildung 5.15: Drag & Drop über Instanzgrenzen hinweg

### 5.8.1 Kopieren und Verschieben mittels Drag & Drop

Sämtlicher Code, welcher für die Drag & Drop Funktionalitäten benötigt wird, ist in der Datei *SidepanelAdminView.xaml.cs* enthalten. Da nur die Dozierenden diese Bearbeitungsmöglichkeit haben, ist es nicht notwendig, die Implementation in eine andere Klasse zu verlagern. Eine Implementierung von Drag & Drop Ereignissen umfasst folgende Methoden (Auszug aus [5]):

GiveFeedback	Dieses Ereignis tritt auf, wenn ein Drag & Drop-Vorgang gestartet wird. Es ermöglicht es dem Ablageziel, Feedbackinformationen an die Ziehquelle zu senden, die damit häufig die Darstellung des Mauszeigers dynamisch anpasst, um dem Benutzer Feedback zu geben. Hierbei handelt es sich um ein Bubbling-Ereignis.
QueryContinueDrag	Dieses Ereignis tritt auf, wenn sich während eines Drag & Drop-Vorgangs der Zustand von Tastatur oder Maustasten ändert. Hierdurch kann die Ablagequelle den Drag & Drop-Vorgang abhängig vom Zustand der Tastatur bzw. Maustasten abbrechen. Hierbei handelt es sich um ein Bubbling-Ereignis.

DragEnter	Dieses Ereignis tritt auf, wenn ein Objekt in die Begrenzung des Ablageziels gezogen wird. Hierbei handelt es sich um ein Bubbling-Ereignis.
DragLeave	Dieses Ereignis tritt auf, wenn ein Objekt aus der Begrenzung des Ablageziels gezogen wird. Hierbei handelt es sich um ein Bubbling-Ereignis.
DragOver	Dieses Ereignis tritt auf, wenn ein Objekt innerhalb der Begrenzung des Ablageziels gezogen (bewegt) wird. Hierbei handelt es sich um ein Bubbling-Ereignis.
Drop	Dieses Ereignis tritt auf, wenn ein Objekt auf dem Ablageziel abgelegt wird. Hierbei handelt es sich um ein Bubbling-Ereignis.

Tabelle 5.1: Methoden für Drag &amp; Drop Ereignisse aus [5]

In Algoria Worksheet wurden die Methoden *GiveFeedback*, *Drop*, *DragEnter* und *DragOver* überschrieben, wobei es sich bei *DragEnter* und *DragOver* um dieselbe Methode handelt. Um den Drag & Drop Modus zu starten, gibt es einen MouseMove-Listener, welcher alle Mausbewegungen über der Seitenleiste erfasst. Sobald dieser Listener erkennt, dass zusätzlich zu einer Mausbewegung auch die Maustaste gedrückt ist, initiiert er den Drag & Drop Modus.

---

```

1 SidepanelElement panelElement = WorksheetUtil.FindAncestor\
2 <SidepanelElement>((DependencyObject)e.OriginalSource);
3 isdragging = true;
4 DroppedInThisInstance = false;
5 DraggedInThisInstance = true;
6 DataObject dragData = new DataObject(DataObjectName, <<
    SerializedExercise>>);
7 DragDrop.AddGiveFeedbackHandler(this, DropList_GiveFeedback);
8 DragDrop.DoDragDrop(this, dragData, DragDropEffects.Move |
    DragDropEffects.Copy);

```

---

Codeausschnitt 5.16: Starten von Drag &amp; Drop

Als Erstes muss festgestellt werden, welches Element das Ziel des Drag & Drop ist. Dazu wird die Methode *FindAncestor* aufgerufen. Sie gibt das erste Väterelement im Visualtree vom Typ *SidepanelElement* zurück. *Isdragging* wird gesetzt, damit kein zweites Drag & Drop initiiert werden kann, während dem schon eines am Laufen ist. Die beiden weiteren Variablen werden benötigt, um beim Kopieren oder Verschieben über Instanzgrenzen hinweg die richtigen Aktionen durchführen zu können. Die serialisierten Aufgabendaten werden in ein *DataObject* gespeichert, damit sie in einer anderen laufenden Instanz von Algoria Worksheet deserialisiert werden können. Weitere Informationen zum Thema *DataObject* sind im Kapitel 5.7 zu finden. Beim Aufruf von *DoDragDrop* beginnt der Drag & Drop Modus. Mit der Abarbeitung des Codes wird an dieser Stelle gewartet, bis der Benutzer die Maustaste, und damit das Element, loslässt. Beim Eintreten in diesen Zustand wird der Event *DragEnter* aufgerufen. Danach wird beim Bewegen der Maus der Event *DragOver* ausgelöst, sofern der Benutzer mit der Maus über den entsprechend definierten Bereich fährt. Lässt der Benutzer die Maustaste los, wird der Event *Drop* ausgelöst. Wichtig ist, dass diese Events immer in der entsprechenden Instanz von Algoria Worksheet ausgelöst werden. Das bedeutet, dass beispielsweise bei der Methode *Drop* abgefragt werden muss, ob es sich um dieselbe Instanz handelt, in der auch Drag & Drop gestartet wurde.

Die Methode *DragEnterOver* behandelt die Events *DragEnter* und *DragOver*. Es wird geprüft, ob der Benutzer mit der Maus über einem gültigen Ablagebereich ist. Falls dies zutrifft, wird ausserdem anhand der Steuerungstaste entschieden, ob es sich um eine Kopier- oder Verschiebeoperation handelt. Dies wird mittels *DragDropEffects.Copy* oder *DragDropEffects.Move* dem Benutzer durch eine Visualisierung mitgeteilt.

```
1 if (CurrentDragOverElement != null)
2 {
3     if (e.KeyStates.HasFlag(DragDropKeyStates.ControlKey))
4     {
5         e.Effects = DragDropEffects.Copy;
6     }
7     else
8     {
9         e.Effects = DragDropEffects.Move;
10    }
11 }
```

---

Codeausschnitt 5.17: Drag & Drop Effekte

Die Methode *DropList\_GiveFeedback* wird nur benötigt, um die Visualisierung der zur Zeit aktiven Aktion (Kopieren oder Verschieben) zu erhalten. Normalerweise funktioniert dies auch ohne eine eigene Implementation von *GiveFeedback*. Da aber Drag & Drop nicht standardmässig auch über die Instanzgrenzen der Anwendung funktioniert, muss hier eine eigene Implementation geliefert werden. Diese sorgt nur dafür, dass die Aktionen aus der Instanz, welche den Drag & Drop Modus gestartet hat, auf die Zielinstanz übertragen werden.

```
1 private void DropList_GiveFeedback(object sender,
2     GiveFeedbackEventArgs e)
3 {
4     ActiveEffects = e.Effects;
5 }
```

---

Codeausschnitt 5.18: Aktive Effekte über Instanzgrenzen übertragen

Die Variable *ActiveEffects* wird nach der Drag & Drop Operation abgefragt. War die ausgeführte Operation ein Verschieben von einer Algoria Worksheet Instanz in eine Andere, dann muss das Element aus dem alten Aufgabenblatt entfernt werden.

```
1 if (!DroppedInThisInstance && ActiveEffects.HasFlag(DragDropEffects.
2     Move))
3 {
4     ViewModel.MainController.DeleteExercise(SelectedElementForDragging
5     );
6 }
```

---

Codeausschnitt 5.19: Entfernen nach Verschieben über Instanzgrenzen

Bei einem Drop-Event wird überprüft, ob es sich um eine Kopier- oder Verschiebeoperation handelt. Handelt es sich um eine Kopieroperation wird das Element einfach an der entsprechenden Stelle eingefügt, indem eine neue Aufgabe aus den serialisierten Daten erzeugt wird. Dies kann instanzunabhängig gemacht werden, da die serialisierten Daten vom *DataObject* sowieso vorhanden sind.

Handelt es sich allerdings um eine Verschiebeoperation, muss zuerst überprüft werden, ob das Element aus derselben Instanz von Algoria Worksheet stammt. Ist dies nicht der Fall, wird eine neue Aufgabe erzeugt, welche wie beim Kopieren aus den serialisierten Daten rekonstruiert wird. Das Löschen aus der alten Instanz übernimmt die *MouseMove*-Methode, welche nach dem Drop weiter ausgeführt wird. Denn wie bereits erwähnt, wird der Ablauf beim DoDragDrop (Listing 5.16) angehalten, bis ein Drop-Event auftaucht. Nun wird der Code aus dem Listing 5.19 ausgeführt.

## 5.8.2 Animation

Währenddessen Drag & Drop aktiv ist, werden Animationen benutzt, um dem Benutzer einen besseren Anhaltspunkt über den Zielort des Elementes zu geben. Ein Zwischenraum wird ausgefahren und eine Linie wird gezeichnet, um den Ablageort hervorzuheben. Alle anderen Zwischenräume sind oder werden eingefahren. Dazu wird in der Methode *DragEnterOver* geprüft, ob sich der Mauszeiger über einem Seitenleistenelement befindet. Ist dies der Fall, dann wird abhängig von der Position der vorhergehende oder nachfolgende Zwischenraum des Elements animiert. Dazu werden die beiden Methoden *AnimatePrevRectangle* und *AnimateNextRectangle* verwendet.

Innerhalb von *AnimatePrevRectangle* wird zuerst der entsprechende Zwischenraum gesucht, für welchen ein Rectangle zur Darstellung verwendet wurde. Sobald dieser gefunden wurde, wird eine Animation darauf registriert, die den Zwischenraum vergrößert und eine Linie zeichnet. Innerhalb derselben Methode wird eine weitere Animation auf alle Zwischenräume registriert, welche zum Zeitpunkt des Aufrufes nicht eingefahren sind.

---

```
1 int Index = Int16.Parse(PreviousRectangle.Name.Substring(2));
2 if (!AnimationInProgressList[Index])
3 {
4     AnimationInProgressList[Index] = true;
5     // create and start animation
6 }
```

---

Codeausschnitt 5.20: Animation der Seitenleiste starten

Der Index wird aus dem Namen des Rectangle-Elements bestimmt. Da diese Elemente dynamisch aus dem Code erzeugt werden, kann sichergestellt werden, dass es sich beim Parsen immer um einen Zahlenwert handelt. Der Index wird dann verwendet, um auf ein Element der Liste *AnimationInProgressList* zuzugreifen. Dies ist eine Liste, welche für jeden Zwischenraum einen booleschen Wert besitzt, der angibt, ob eine Animation auf dem zugehörigen Zwischenraum läuft. Falls dies der Fall ist, darf keine neue Animation darauf registriert werden, sondern die Laufende muss zuerst abgeschlossen werden.

Ist eine Animation abgeschlossen, wird automatisch das Event *Completed* ausgelöst. Auf dieses ist folgende Methode registriert, welche die *AnimationInProgressList* an der entsprechenden Stelle zurücksetzt.

```
1 void RectangleStoryboard_Completed(Object sender, EventArgs e)
2 {
3     Storyboard Board = (Storyboard)((ClockGroup)sender).Timeline;
4     Rectangle TargetRect = (Rectangle)Storyboard.GetTarget(Board);
5
6     int Index = Int16.Parse(TargetRect.Name.Substring(2));
7     AnimationInProgressList[Index] = false;
8 }
```

---

## Codeausschnitt 5.21: Animation beendet

Das *Storyboard* ist ein Container für eine Timeline mit Zielelement. Es enthält Informationen zur enthaltenen Animation, wie beispielsweise den Wert und die Zieleigenschaft. Mit *GetTarget()* kann das Animationsziel abgefragt werden. Dies wird hier benötigt, um den Index des animierten Zwischenraumes zu rekonstruieren. Dadurch kann in der *AnimationInProgressList* der Wert an der entsprechenden Stelle neu gesetzt werden. Nach der Ausführung dieses Codeabschnittes kann eine neue Animation auf dem Zwischenraum gestartet werden.

## 5.9 Hilfe

Wie die meisten Programme heutzutage verfügt Algoria Worksheet über eine Hilfe im Microsoft Windows Stil. Dieses Kapitel befasst sich mit Microsoft Hilfedateien, welche für Algoria Worksheet erstellt worden sind, und der Implementation von kontextsensitivem Verhalten der Hilfe, beim Klicken der F1 Taste. Nebst dem kontextsensitiven Benutzen ist es zudem möglich, die Hilfe über das Menü zu öffnen. Für das Darstellen der Hilfedateien wurde die *Help*-Klasse verwendet. Die Klasse verfügt über eine *ShowHelp* Methode, welche den Pfad zu einer Hilfedatei entgegennimmt und die Hilfe im bekannten Windows Stil darstellt.

Die gesamte Hilfe-Logik ist in der statischen Klasse *HelpProvider* zu finden. Zur Startseite der Hilfe kommt man mit einer einzigen Zeile. Anzumerken ist, dass für die Dozierenden- als auch für die Studierendenversion eine eigene Hilfedatei geschrieben und kompiliert wurde. Die *VersionController*-Klasse wird für das Abfragen des Namens der Hilfedatei verwendet. So ist keine Versionsunterscheidung der *HelpProvider*-Klasse nötig.

---

```

1 String helpFile = @"Help/" + VersionController.GetHelpFileName();
2 Help.ShowHelp(null, helpFile);

```

---

Codeausschnitt 5.22: Öffnen der Startseite der Hilfe

### 5.9.1 Microsoft Hilfedatei

Eine Microsoft Hilfedatei ist ein kompiliertes Archiv aus HTML-Dateien (.CHM-Datei), welche die eigentlichen Hilfetexte enthalten. Für das Erstellen einer solchen .CHM-Datei wurde das Microsoft Programm *HTML Help Workshop*[3] verwendet. Für jeden der Bereiche in Algoria Worksheet (beispielsweise die Aufgabenstellung oder die Seitenleiste) wurden HTML Dateien erstellt, welche die Handhabung erklären. Durch das Kompilieren des Projektes im *HTML Help Workshop* wird eine .CHM-Datei erstellt. Diese kann nun verwendet werden.

### 5.9.2 Kontextsensitive Hilfe

Zusätzlich zum statischen Öffnen der Hilfe, wurde die Verwendung der F1 Taste ermöglicht. Diese reagiert kontextsensitiv und öffnet die Hilfe auf der HTML-Seite, welche die momentan aktive Region behandelt. Damit die *HelpProvider*-Klasse die momentan aktive Region erkennen kann, ist das Einfügen einer Kennung für jede Region notwendig, welche durch ein Kapitel in der Hilfe beschrieben wird. Für jede Region wurde die Kennung so gewählt, dass sie mit dem Namen der dazugehörigen HTML-Datei übereinstimmt.

Die folgenden Zeilen sind aus der *WorksheetShell* XAML-Datei.

---

```

1 xmlns:h="clr-namespace:IMVS.AlgoriaWorksheet"
2 h:HelpProvider.HelpString='AlgoriaWorksheet'

```

---

Codeausschnitt 5.23: Einfügen der *HelpString*-Variable

Diese Kennung kann für jedes *Grid* oder jeden *ScrollViewer* gesetzt werden.

---

```
1 <Grid Name="ExerciseRegion" ... h:HelpProvider.HelpString='
    ExerciseRegion' />
```

---

Codeausschnitt 5.24: Deklarieren der *HelpString*-Kennung

Damit auf den Wert der *HelpString*-Variable zugegriffen werden kann, muss ein *DependencyProperty* für diese Variable definiert werden, welche sich auf den Namen der Variablen registriert.

---

```
1 public static readonly DependencyProperty HelpStringProperty =
    DependencyProperty.RegisterAttached("HelpString", typeof(string),
    typeof(HelpProvider));
```

---

Codeausschnitt 5.25: Öffnen der Startseite der Hilfe

Die *HelpProvider*-Klasse registriert sich auf das Ereignis, welches ausgelöst wird, sobald der Benutzer die F1 Taste betätigt. Die dafür vorgesehene Methode wird nur dann aufgerufen, falls der Sender eine, zur Identifizierung notwendige, Definition der *HelpString*-Variable enthält. In diesem Falle wird wieder die *Help*-Klasse für das Anzeigen verwendet.

---

```
1 String helpTopic;
2 if ("FBEngine".Equals(HelpProvider.GetHelpString(sender as
    FrameworkElement)))
3 {
4     helpTopic = ForegroundEngineName + ".htm";
5 }
6 else
7 {
8     helpTopic = HelpProvider.GetHelpString(sender as FrameworkElement)
        + ".htm";
9 }
10 String helpFile = @"Help/" + VersionController.GetHelpFileName();
11 Help.ShowHelp(null, helpFile, HelpNavigator.Topic, helpTopic);
```

---

Codeausschnitt 5.26: Öffnen der Startseite der Hilfe

Im Gegensatz zum einfachen Öffnen der Startseite wird eine überlagerte Version der *ShowHelp*-Methode verwendet. Diese ermöglicht es, direkt auf ein Kapitel zuzugreifen. Der dafür notwendigen HTML-Dateinamen ist durch die Kennung des Senders bekannt.

In der Dozierendenansicht sind sowohl *InitialStateEngine* als auch die *SampleSolutionEngine* in derselben Region enthalten. Die *HelpProvider*-Klasse verfügt über eine Variable, in welcher der Name der Engine, welche im Moment im Vordergrund ist, gespeichert wird.

## 6 Lösungsprüfer

Algoria Worksheet enthält einen Lösungsprüfer, der die Lösung der Studierenden mit der Musterlösung vergleicht. Die Überprüfung der Multiple Choice Antworten ist simpel, diese können direkt mit der Musterlösung verglichen werden. Der Vergleich zweier oder mehrerer Datenstrukturen ist allerdings nicht so einfach. Daher wird in den folgenden Kapiteln die Theorie und Strategie für die Behandlung dieses Problems beschrieben.

Der Lösungsprüfer überprüft nicht einfach nur die Datenstrukturen auf Gleichheit, sondern er kann den Studierenden auch hilfreiche Hinweise geben. Die Studierenden werden damit, falls ihre Lösung nicht schon korrekt ist, schrittweise auf die Lösung hingeführt. Eine Überprüfung auf Gleichheit mit der Antwort 'gleich' oder 'nicht gleich' führt schnell zu Verzweiflung. Lautet die Aufgabe beispielsweise nur, ein sortiertes Array zu zeichnen, wäre die Lösung von Abbildung 6.1 eigentlich korrekt, eine normale Gleichheitsüberprüfung würde es aber als falsch ansehen, bis auch die Zahlen identisch sind.

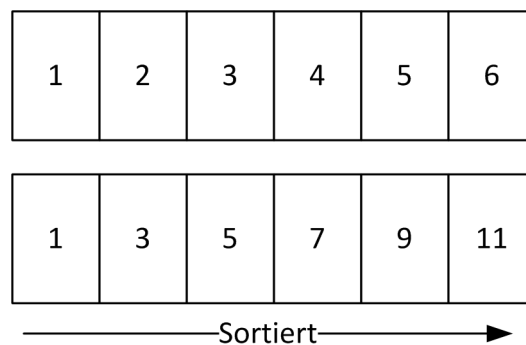


Abbildung 6.1: Vergleich von sortierten Arrays

Das Ziel des Lösungsprüfers ist es, korrekte Teile in der Lösung der Studierenden zu finden, und dann anzuzeigen, welche Komponenten noch fehlen oder falsch sind. Auch dies ist kein einfaches Problem. Im nächsten Kapitel wird das Problem im Allgemeinen beschrieben.

## 6.1 Subgraph-Isomorphismen und Graph matching

Wie findet der Lösungsprüfer einen korrekten Teil in der gezeichneten Datenstruktur der Studierenden? Als Ausgangslage sind zwei Datenstrukturen gegeben; eine Musterlösung und eine Studierendenlösung. Da die beiden Datenstrukturen nicht dieselbe Grösse haben müssen, gibt es folgende möglichen Fälle:

- Die Musterlösung ist grösser als die Studierendenlösung. Ist der Studierendengraph bisher korrekt, wäre er identisch zu einem Teil der Musterlösung. Es ist aber auch möglich, dass nur ein Teil der Studierendenlösung mit einem Teil der Musterlösung übereinstimmt.
- Beide Datenstrukturen haben dieselbe Grösse. Sie könnten daher entweder identisch sein, oder ein Teil der Studierendenlösung stimmt mit einem Teil der Musterlösung überein.
- Letztlich gibt es noch den Fall, in welchem die Musterlösung kleiner ist als die Studierendenlösung. In dieser Situation könnte ein Teil der Studierendenlösung mit dem kompletten Musterlösungsgraph identisch sein. Die andere Möglichkeit ist, dass wiederum nur jeweils ein Teil übereinstimmt, wie in den beiden anderen Fällen auch.

Auf das Allgemeine bezogen, handelt es sich hierbei um das sogenannte *Subgraph-Isomorphismus Problem*<sup>1</sup>. Dieses Problem ist NP-Vollständig, und daher vermutlich nicht effizient lösbar. Dieses Problem wird vor allem im Zusammenhang mit der Mustererkennung in Chemie, Biologie und der Informatik erforscht. Es gibt einige Ansätze und Algorithmen, welche zwar funktionieren, aber deren Laufzeit sehr schlecht ist. Einer der berühmteren Algorithmen wurde von Julian R. Ullmann 1976 [6] entwickelt. Dieser Algorithmus funktioniert mittels Backtracking und arbeitet mit den Adjazenzmatrizen der Graphen. Mit Operationen auf einer Permutationsmatrix versucht der Algorithmus aus dem einen Graphen einen Isomorphen Graphen zu finden, welcher mit dem Vergleichsgraphen übereinstimmt.

Weitere Ansätze und Informationen sind in den Veröffentlichungen von Battiti/Mascia[7] und Bunke/Shearer[8] zu finden. Aus den gegebenen Informationen hat sich gezeigt, dass es für den Lösungsprüfer in Algoria Worksheet mehr Sinn macht, die Probleme auf die einzelnen Datenstrukturen einzuengen. Dadurch kommen spezifische Eigenschaften und Gegebenheiten hinzu, die der Lösungsprüfer ausnutzen kann, um die Datenstrukturen zu vergleichen. So hat beispielsweise ein Array oder eine einfach verkettete Liste niemals mehrere Folgeknoten. Ein Baum ist dagegen schon komplexer, besitzt aber keine Zyklen und jedes Objekt hat nur einen Vorfahren. Im Umfang dieser Arbeit konnte eine umfassende Theorie für die Behandlung der Arrays entwickelt werden, welche voraussichtlich auch die Basis für den Algorithmus der einfach verketteten Liste bietet. Im Folgenden wird diese beschrieben.

---

<sup>1</sup>[http://en.wikipedia.org/wiki/Subgraph\\_isomorphism\\_problem](http://en.wikipedia.org/wiki/Subgraph_isomorphism_problem)

## 6.2 Metrik

Der Lösungsprüfer in Algoria Worksheet arbeitet anstelle von Gleichheiten mit Ähnlichkeiten. Eine bekannte Methode für die Bestimmung der Ähnlichkeit zweier Strings ist die Levenshtein-Distanz<sup>2</sup>. Die Levenshtein-Distanz liefert eine Metrik, welche die Operationen, welche notwendig sind um den ersten String in den zweiten umzuwandeln, wiedergibt. Levenshtein definierte drei Operationen (Einfügen, Löschen und Ersetzen eines Zeichens). Je weniger Operationen notwendig sind, desto ähnlicher sind die zwei Strings.

Analog zur Levenshtein-Distanz wurde auch für die Messung der Ähnlichkeit von zwei Arrays eine Metrik erstellt. Die Metrik enthält drei Elementar Operationen. Im Vergleich zu den Operationen der Levenshtein-Distanz haben im Algoria Worksheet Lösungsprüfer nicht alle Operationen dasselbe Gewicht. Im Folgenden werden die definierten Operationen beschrieben:

Operation	Beschreibung
Set Value Status 1	Ein Wert im Lösungsarray wird durch einen beliebigen ersetzt.
Set Value Status 2	Ein Wert im Lösungsarray wird durch einen anderen Wert im Lösungsarray ersetzt.
Add Field	Am Ende des Arrays wird ein leeres Feld hinzugefügt.
Delete Field	Ein beliebiges Feld wird gelöscht.

Tabelle 6.1: Operationen der Ähnlichkeits-Metrik

Die Operationen wurden so gewählt, dass sie orthogonal zueinander sind. Dies bedeutet, dass keine Operation durch eine Kombination der restlichen Operationen ersetzt werden kann, und der gleiche Effekt erzielt wird. Die Operation *Set Number* besitzt zwei Gewichtungen, da sie in zwei verschiedenen Stadien des Vergleichsalgorithmus verwendet wird. Im ersten Status (*Grösse anpassen*) ist das Gewicht einer *Set Value*-Operation sehr viel grösser, als im darauffolgenden *Swaps berechnen*-Status. Der Grund liegt darin, dass im ersten Status Werte gesetzt werden können, welche im Studienenden-Array gar nicht vorkommen, sondern nur durch einen aus dem Lösungs-Array ersetzt wird. Dies ist eine wesentlich grössere Anpassung, als das Austauschen (swap) von zwei Werten, welches durch zweimaliges *Set Value* modelliert wird.

Während der Algorithmus durchläuft, wird das Vorkommen der einzelnen Operationen gezählt. Durch das Multiplizieren der Anzahl einer Operation mit der Gewichtung gibt der Algorithmus eine Zahl aus, welche analog zur Levenshtein-Distanz verwendet werden kann. Soll heissen: Je kleiner die Zahl, welche der Algorithmus liefert, desto ähnlicher sind sich die zwei Arrays.

Die Gewichtung der Operationen sind dem Dozierenden überlassen. Je nach Aufgabenstellung ist es schwerwiegender, wenn die Lösung zu wenige oder zu viele Elemente hat. In diesem Falle wären *Add Field* und *Delete Field* sehr viel höher gewichtet als *Set Value*. Andere Aufgaben hängen nicht so stark von der Anzahl Arrays, sondern von den darin enthaltenen Werten ab und darin, dass nur Werte aus der Musterlösung vorkommen. Hier werden die Gewichtungen von *Add Field* und *Delete Field* gemindert und die für den *Set Value* im Status 1 sehr stark erhöht.

<sup>2</sup>[http://en.wikipedia.org/wiki/Levenshtein\\_distance](http://en.wikipedia.org/wiki/Levenshtein_distance)

## 6.3 Grobkonzept

Der Lösungsprüfer von Algoria Worksheet kann nicht selbst entscheiden, ob eine Lösung richtig oder falsch ist, da er die Aufgabe nicht kennt. Um das zu verdeutlichen sind hier zwei Beispiele, welche unterschiedliche Aufgabentypen umfassen:

- Die Lösung enthält nur ein sortiertes Array, unabhängig von seinem Inhalt. Das bedeutet, die Zahlenwerte sind nur für die Ordnung relevant.
- Die Lösung muss exakt dieselben Zahlen an den exakt selben Positionen enthalten. Die Arrays müssen dieselbe Grösse haben.

Weiterhin ist davon auszugehen, dass gewisse Aufgabentypen einige Operationen der Metrik schwerer gewichten als andere. Vorstellbar ist hier eine Aufgabe, bei welcher die Anzahl Elemente und derer Werte schon vorgegeben ist, die Studierenden müssen daher nur noch die Elemente an die richtigen Positionen verschieben. Eine solche Aufgabe würde eine falsch positionierte Zahl mit mehr Fehlerpunkten gewichten, als eine Aufgabe, bei welcher ein Array erst noch aufgebaut werden müsste.

Diese Vielfalt von Aufgabentypen führt dazu, dass der Lösungsprüfer in gewissen Punkten parametrisierbar sein muss. Deshalb erhalten die Dozierenden folgende Möglichkeiten bei der Erstellung einer Aufgabe:

- Es kann definiert werden, dass die Lösung die exakten Zahlen enthalten muss. Das bedeutet, der Lösungsprüfer berechnet die benötigten Operationen, bis beide Arrays identisch sind.
- Es kann definiert werden, dass die Lösung nur eine bestimmte Ordnung erfüllen muss. Dies kann aufsteigend oder absteigend sein.
- Es kann jeweils eine Gewichtung für die Elementaroperationen *add*, *remove* und *setValue* gesetzt werden. Diese Werte werden bei der Berechnung der Metrik verwendet um die Fehlerpunkte, beziehungsweise das Ähnlichkeitsmass zu erhalten.

Die zwei Fälle, exakte Zahlen oder Ordnung, müssen separat behandelt werden. Der Vergleich mit einer Lösung, welche nur eine Ordnung einhalten muss, ist relativ einfach. In einem ersten Schritt werden berechnet, wie viele Elemente fehlen oder zu viel sind. Danach kann ein Sortieralgorithmus darauf angewendet werden, welche die minimale Anzahl Swaps verwendet.

Der Vergleich zwischen zwei Arrays mit Berücksichtigung der korrekten Zahlenwerte ist etwas komplexer. Die Grundidee ist, den Ablauf in zwei separate Teile zu unterteilen. Im ersten Teil werden die Arrays gleich gross gemacht, wobei dazu einfach deren Schnittmenge verwendet wird. Hierbei werden die benötigten Operationen zur Metrik hinzugefügt. In Teil zwei werden die übrig gebliebenen Arrays mit Swaps verändert, bis sie identisch sind. Die Anzahl Swaps wird dann in der Metrik mit der Operation *setValue* miteinberechnet. Im folgenden Kapitel wird der dafür vorgesehene Algorithmus beschrieben.

## 6.4 Algorithmus mit minimaler Anzahl Swaps

Die Ausgangslage für diesen Algorithmus sind zwei Arrays derselben Grösse, welche dieselben Zahlen beinhalten. Ziel ist es, die geringste Anzahl an Vertauschoperationen zu berechnen, damit beide Arrays identisch sind.

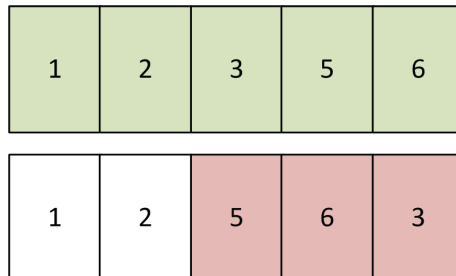


Abbildung 6.2: Ausgangslage des Algorithmus

Abbildung 6.2 zeigt ein Beispiel, in welchem die ersten Elemente übereinstimmen, die letzten drei allerdings vertauscht sind. Der Algorithmus verwendet eines der Arrays als Referenzarray ( $M$ ) und arbeitet dann auf dem Eingabearray ( $E$ ).

---

```

1 swaps = 0
2 visited = []
3 for i = 0 to |E| {
4     if i not in visited {
5         swapgroup = []
6         visited.add(i)
7         v = E[i]
8
9         while v not in swapgroup {
10            swapgroup.add(v)
11            p = suche v innerhalb M und liefere die Position
12            visited.add(p)
13            v = E[p]
14        }
15
16        swaps += (swapgroup.count - 1)
17    }
18 }
```

---

Codeausschnitt 6.1: Algorithmus in Pseudocode

Der Algorithmus startet beim ersten Element des Eingabearrays. Er erzeugt sogenannte *Swapgruppen* und berechnet dann anhand dieser die benötigten Swaps. Die besuchten Positionen im Eingabearray werden in einer Liste *visited* gespeichert. Jedes Element des Eingabearrays muss nur einmal besucht werden, somit kann bei den folgenden Durchgängen geprüft werden, ob eine Position schon einmal besucht wurde. Der Wert des Elementes an dieser Position wird nun in die Swapgruppe eingefügt. Dann wird dieser Wert im Referenzarray gesucht und dessen Position  $p$  zurückgegeben. Nun wird diese Position in die Liste *visited* gespeichert. Es wird geprüft, ob der Wert des Elementes an dieser Position bereits in der Swapgruppe ist. Falls dies nicht der Fall ist, geht es weiter, indem wieder dieser

Wert im Referenzarray gesucht wird. Dies geht so lange weiter, bis ein besuchter Wert bereits in der Swapgruppe enthalten ist. Somit ist die Swapgruppe vollständig, und deren um eins verringerte Grösse wird zu den benötigten Swaps hinzugerechnet. Für jede noch nicht besuchte Position wird dieser Ablauf zur Erzeugung einer Swapgruppe durchgeführt. Der Algorithmus weist somit jedes Element einer Swapgruppe zu und berechnet dann anhand Swapgruppen und Gruppengrösse die benötigten Swaps.

Das Auffinden aller Swapgruppen ist leider ein quadratischer Aufwand. Es wird zwar jedes Element in  $E$  nur einmal besucht ( $|E|$ ), aber für jedes von diesen, muss der dazugehörige Wert in  $M$  gefunden werden. Dies braucht im schnitt  $\frac{|M|}{2}$  Vergleiche. Da  $|M|$  und  $|E|$  gleich sind, und der Faktor  $\frac{1}{2}$  vernachlässigbar ist, kann die Laufzeit des Algorithmus auf  $O(n^2)$  festgelegt werden.

Der Algorithmus ist nicht so komplex, wie er im ersten Moment erscheinen mag. Im folgenden Kapitel werden Swapgruppen genauer definiert, damit der Algorithmus klarer wird. Ausserdem wird bewiesen, dass der Algorithmus tatsächlich die minimal benötigte Anzahl Swaps ausrechnet.

### 6.4.1 Theorie

**Satz:** Der Algorithmus findet die minimale Anzahl Swaps:

$$r = \sum_{i=1}^n (|G_i| - 1) \tag{Eq. 1}$$

**Definition 1:** Als Input sind zwei Arrays gegeben, ein Musterarray  $M$  und ein Eingabearray  $E$ . Der Algorithmus errechnet die Anzahl benötigter Swaps um vom Eingabearray auf das Musterarray zu kommen. Die Elemente des Musterarrays werden mit  $m_i$ , die Elemente des Eingabearrays mit  $e_i$  adressiert.

**Definition 2:** Eine Swapgruppe ist eine Menge von Elementen  $e_i$  welche miteinander vertauscht werden müssen, damit der Wert von  $m_i$  mit  $e_i$  übereinstimmt. Eine Swapgruppe wird konstruiert, indem mit einem Element  $e_i$  gestartet wird, davon der Wert  $V(e_i)$  genommen wird und dieser in  $M$  gesucht wird. Die gefundene Position  $j$  führt zum nächsten Element  $e_j$  der Swapgruppe. Dieser Ablauf wird wiederholt, bis ein Wert  $V(e_j)$  zum zweiten Mal gefunden wurde. Dann nämlich ist der Zyklus geschlossen und die Suche ist innerhalb einer Gruppe einmal komplett im Kreis gelaufen.

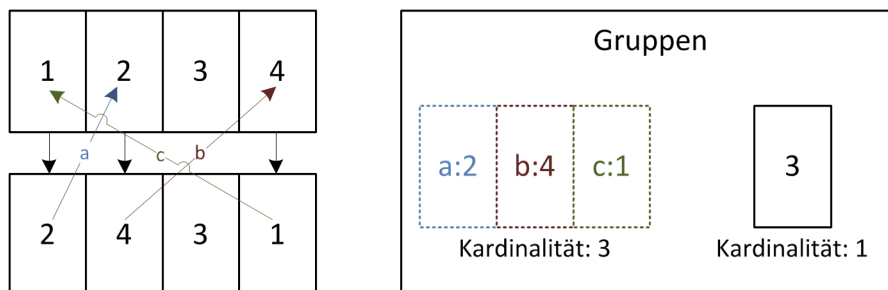


Abbildung 6.3: Beispiel: Swapgruppen

Jedes Element  $e_i$  ist in genau einer Swapgruppe enthalten. Ein Element an der korrekten Position bildet eine eigene Swapgruppe der Grösse 1. Die Anzahl enthaltene Elemente entspricht der Gruppenkardinalität  $|G|$ .

**Definition 3:** Ein Swap ist eine Operation, welche zwei Zahlen miteinander vertauscht. Für nachfolgende Erläuterungen wird die folgende Notation verwendet:  $\text{Swap}(0-0)$ ,  $\text{Swap}(1-0)$ ,  $\text{Swap}(0-1)$  und  $\text{Swap}(1-1)$ . Der Inhalt der Klammern zeigt hierbei an, ob und welche Zahlen nach der Operation an der korrekten Stelle sind. Ein  $\text{Swap}(1-1)$  beispielsweise führt dazu, dass beide Zahlen nach dem Vertauschen an ihrer richtigen Position sind und daher nicht noch in einer weiteren Operation vorkommen werden.

**Lemma 1:** Eine Gruppe kann nicht mit weniger als  $r = |G| - 1$  Swaps sortiert werden.

**Beweis:**

*Fall 1 ( $|G| = 1$ ):* Falls die Gruppe eine Kardinalität von 1 hat, ist das Element bereits an der korrekten Position und muss daher nicht vertauscht werden. Die Anzahl benötigter Swaps ist daher  $r = 1 - 1 = 0$ .

*Fall 2 ( $|G| > 1$ ):* In diesem Fall ist jedes Element der Gruppe an einer falschen Position und muss daher mindestens einmal mit einem Swap verschoben werden. Wäre ein Element zu Beginn bereits richtig positioniert, wäre es gemäss Definition 2 nicht in der Gruppe enthalten.

Definition 3 sagt aus, dass maximal zwei Zahlen mit einem Swap richtig positioniert werden können. Der Algorithmus verwendet ausschliesslich  $\text{Swap}(1-0)$  und einen  $\text{Swap}(1-1)$ , daher benötigt er für jede Swapgruppe exakt  $|G| - 2$   $\text{Swap}(1-0)$  und einen  $\text{Swap}(1-1)$  um alle Elemente von  $G$  zu positionieren. Um also ein besseres Resultat zu erhalten, müssten mehrere  $\text{Swaps}(1-1)$  verwendet werden, ohne dass zum Ausgleich ein  $\text{Swap}(0-0)$  benötigt wird. Im folgenden Abschnitt soll bewiesen werden, dass mehrere  $\text{Swaps}(1-1)$  nicht innerhalb einer Swapgruppe möglich sind.

Einen  $\text{Swap}(1-1)$  kann es nur geben, wenn beide Zahlen vorher an der falschen Position waren und nach dem Vertauschen an der korrekten Position sind. In einer Swapgruppe mit mehr als zwei Elementen muss also eine der beiden Zahlen an eine falsche Position gebracht werden um einen  $\text{Swap}(1-1)$  zu ermöglichen. Auf der anderen Seite der Vertauschoperation ergibt sich aber eine Zahl an der korrekten Position, es sei denn, ein  $\text{Swap}(0-0)$  wurde verwendet. Da ein  $\text{Swap}(0-0)$  aber keine Zahl an die korrekte Position bringt, erhöht er die minimale Anzahl Vertauschungen um eins. Folgende Illustration soll verdeutlichen, dass mehrere  $\text{Swaps}(1-1)$  innerhalb einer Gruppe immer durch einen  $\text{Swap}(0-0)$  ausgeglichen werden müssen.

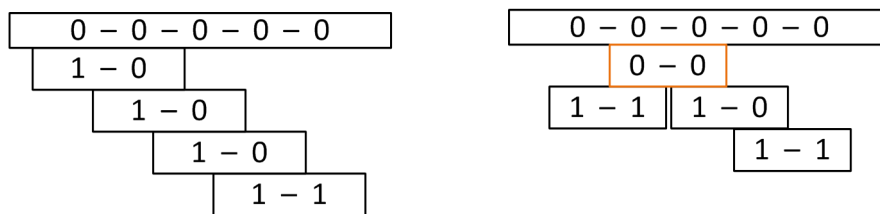


Abbildung 6.4: Schematische Darstellung möglicher Swaps

Dieser benötigte  $\text{Swap}(0-0)$  hat zur Folge, dass sich die Swapgruppe in zwei eigenständige Gruppen aufteilt. Eine Swapgruppe ist gemäss Definition 2 ein geschlossener Zyklus und kann daher als Ringliste visualisiert werden (Abbildung 6.5). Dabei zeigt die Referenz aber nicht auf das nächste Element, sondern auf das Element an der Position  $x$ , wobei  $x$  der Wert des aktuellen Elements ist. Diese Tatsache führt dazu, dass jeder  $\text{Swap}(0-0)$  innerhalb einer Gruppe eine Aufteilung der Gruppe in zwei Eigene zur Folge hat. Die folgende Abbildung zeigt ein Beispiel dazu. Die grün markierten Elemente

werden mit einem Swap(0-0) vertauscht, wodurch die Referenzen neu zwei abgetrennte Swapgruppen ergeben.

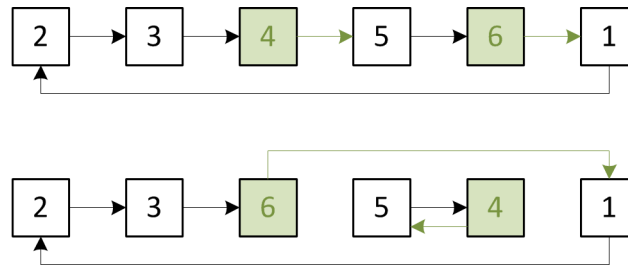


Abbildung 6.5: Swapgruppe als Ringliste

**Lemma 2:** Ein Array kann auch mit Swaps(0-0) nicht in weniger als  $r = \sum_{i=1}^n (|G_i| - 1)$  Schritten geordnet werden, wobei  $n$  der Anzahl Swapgruppen des Arrays entspricht.

**Beweis:**

Ein Swap(0-0) teilt eine Gruppe in zwei Gruppen auf, oder vereinigt zwei Gruppen zu einer Gemeinsamen.

*Fall 1 (teilt):* Angenommen ein Array besitzt zwei Swapgruppen der Grössen 4 und 6. Ein Swap(0-0) teilt nun die Gruppe mit der Kardinalität 6 in zwei 3er-Gruppen auf. Mit der Formel aus Lemma 1 lässt sich folgende Anzahl Swaps für die Ausgangslage errechnen:  $r = (4 - 1) + (6 - 1) = 8$   
 Nach der Aufteilung der grösseren Gruppe ergeben sich immernoch 8 Swaps (Swap(0-0) muss miteinbezogen werden):  $r = (4 - 1) + (3 - 1) + (3 - 1) + 1 = 8$   
 Es lässt sich die allgemeine Formel ableiten, welche beweist, dass die minimale Swapanzahl dieselbe bleibt, falls ein Swap(0-0) eine Gruppe aufteilt:

$$(|G| - 1) = (|G_1| - 1) + (|G_2| - 1) + 1 \qquad G = G_1 \dot{\cup} G_2 \qquad (\text{Eq. 2})$$

*Fall 2 (vereinigt):* Ausgehend von derselben Konstellation würde nun der Swap(0-0) die beiden Gruppen mit den Kardinalitäten von 4 und 6 vereinigen. Es ergibt sich eine neue Swapgruppe der Grösse 10. Die Anzahl Vertauschungen vor dem Swap(0-0) wäre ebenfalls 8. Nach der Vereinigung wäre sie allerdings 10, wenn der Swap(0-0) mitberechnet wird:  $r = (10 - 1) + 1 = 10$   
 Dieser Fall stellt daher sogar eine Verschlechterung dar, und kann daher Lemma 2 nicht widerlegen.

Der Fall, dass ein Swap(0-0) weder eine Gruppe vereinigt, noch eine Gruppe aufteilt, existiert nicht. Dieses Szenario würde bedeuten, dass ein Swap(0-0) innerhalb einer Gruppe die Gruppe unverändert lassen würde. Im Beweis zu Lemma 1 wird gezeigt, dass ein Swap(0-0) innerhalb einer Gruppe zwingend eine Aufteilung in zwei eigene Gruppen zur Folge hat.

Es ist damit bewiesen, dass selbst Swaps(0-0) keine Reduktion der benötigten Vertauschungen zur Folge haben, sondern nur die Anzahl Gruppen erhöhen kann, welche dann aber zusammen mit den Swaps(0-0) wieder in derselben Anzahl Vertauschungen zu lösen sind. Da der Algorithmus die Swaps mithilfe von Swapgruppen berechnet, erhält er die minimale Anzahl. Der Algorithmus verwendet pro Gruppe genau  $n - 2$  Swaps(1-0) und einen Swap(1-1). Aus diesem Grund bewegt er sich nicht über Gruppengrenzen hinweg, und löst jede Gruppe in exakt  $n - 1$  Schritten. Der Satz ist damit bewiesen.

### 6.5 Detailkonzept

Wie bereits im Grobkonzept beschrieben, kann für jede Aufgabe eingestellt werden, ob die Lösung nur eine Ordnung ist, oder die korrekten Zahlenwerte besitzen muss. Diese Unterscheidung macht der Lösungsprüfer direkt zu Beginn. Abbildung 6.6 zeigt den Ablauf des Lösungsprüfers.

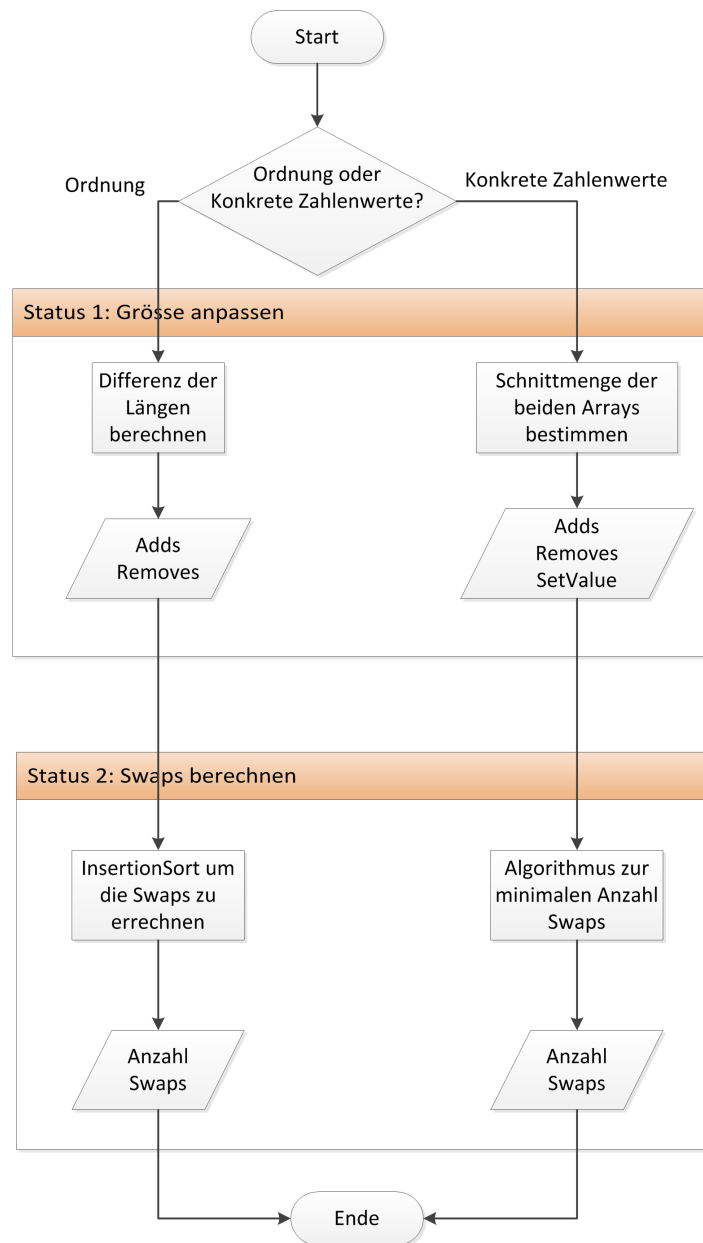


Abbildung 6.6: Ablauf des Lösungsprüfers

Der gesamte Ablauf ist in zwei Status unterteilt. In der ersten Phase werden die beiden Arrays auf dieselbe Grösse gebracht. Falls der Lösungsprüfer nur auf Ordnung überprüfen soll, kann einfach die Differenz zwischen den Arraygrößen verwendet werden. Diese wird dann als *add* oder *remove* in der Metrik eingerechnet.

Muss der Lösungsprüfer auf korrekte Zahlenwerte prüfen, wird in dieser ersten Phase die Schnittmenge beider Arrays berechnet. Dazu wird einfach geschaut, ob eine Zahl in beiden Arrays vorkommt und

beide Arrays werden nur auf diese Zahlen reduziert. Daraus entstehen *adds* und *removes*. Diese werden aber nicht direkt in die Metrik eingetragen, sondern die Differenz der *adds* und *removes* werden als *setValues* eingetragen. Die übrigen *adds* oder *removes* werden dann in die Metrik einbezogen. Dadurch werden falsche Zahlen, die in der Musterlösung nicht vorkommen, mit einem *setValue* einfach ersetzt. Ein wichtiger Punkt ist, dass die Operation *setValue* in der ersten Phase schwerer bewertet wird, als in der zweiten Phase.

In der zweiten Phase werden auf den übrig gebliebenen Arrays die minimalen Anzahl Swaps berechnet, um diese dann als jeweils zwei *setValues* einzuberechnen. Ist für die korrekte Lösung bloss eine Ordnung gefordert, kann ein Sortieralgorithmus verwendet werden, welcher das Problem in minimaler Anzahl Swaps löst. Dazu kann auch der zuvor beschriebene Algorithmus verwendet werden, welcher auf den Swapgruppen basiert.

Für das Überprüfen auf die konkreten Zahlenwerte wird in dieser Phase der in Kapitel 6.4 definierte Algorithmus verwendet, um die minimale Anzahl Swaps zu erhalten. In beiden Fällen werden dann die Anzahl benötigten Swaps mit zwei multipliziert, um sie dann als *setValue* in der Metrik miteinberechnen zu können.

Dieser Ablauf wird für jede Kombination von Musterarray und Eingabearray durchgeführt. So ergibt sich für jedes Paar einen Zahlenwert, der die Ähnlichkeit bestimmt. Je kleiner dieser Wert ist, desto ähnlicher sind sich zwei Arrays. So kann auch eine Zuordnung stattfinden, falls dies benötigt wird.

# 7 Reflexion

## 7.1 Windows Presentation Foundation

Das Design und die Erstellung der Benutzeroberfläche ist mit WPF sehr einfach. Sehr viele benötigte Elemente sind standardmässig enthalten, so ist beispielsweise der *FlowDocumentViewer* eine dieser Standardkomponenten, welche sehr hilfreich ist.

Das Konzept der *Bindings* erleichtert die Darstellung und das Aktualisieren der Daten im Model. Mit diesem Konzept lässt sich das MVVM-Entwurfsmuster sehr gut und einfach umsetzen. Es ist beeindruckend einfach, eine neue View zu erstellen und diese auf ein bestehendes Model zu binden, ohne dass dabei viele Änderungen nötig sind.

Die selbst erstellten Komponenten können beliebig aufgebaut sein, und in jeder XAML-Datei verwendet werden. Das ermöglicht es, eigene *UserControls* zu erzeugen, die auf die persönlichen Anforderungen zugeschnitten sind. Für das einheitliche Look and Feel zusammen mit den Standardkomponenten, gibt es die Möglichkeit, bestehende Styles zu überschreiben. Mit einem *ResourceDictionary* können Standardstyles für Elemente definiert werden, die dann automatisch auf alle Elemente eines bestimmten Typs angewendet werden. Dies ist zwar etwas mühsam, da viele Zeilen Code geschrieben werden müssen, um eventuell nur eine kleine Änderung zu machen, aber die Möglichkeiten sind dafür fast unbeschränkt.

Ein weiterer, zu erwähnender, Aspekt sind die *ApplicationCommands*. Diese bieten eine Liste von vordefinierten Aktionen für ein Programm. Davon werden beispielsweise *Copy* und *Cut* in Algoria Worksheet verwendet. Ein Vorteil davon ist, dass die Tastenkürzel automatisch registriert werden, wenn man *ApplicationCommands* verwendet. Es ist eine geringfügige Verringerung des Aufwandes, da alle diese Standardaktionen schon vordefiniert sind. Für ein Programm werden sie auf jeden Fall benötigt, daher ist dies eine Erleichterung.

Bei der Entwicklung mit WPF und dem Verarbeiten der Benutzerinteraktionen muss vorsichtig vorgegangen werden. Falls viele Events verwendet werden, kann schnell die Übersicht verloren gehen. Das ist ein Problem, weil die Events sich möglicherweise selber auslösen. Dies führt zu ungewollten Seiteneffekten. In Algoria Worksheet gab es Probleme mit inkonsistenten Daten zwischen View und Model. Das Problem wurde zwar lokalisiert und behoben, aber es hat sich dadurch gezeigt, dass die Nebeneffekte von vielen Event-Aufrufen etwas unterschätzt wurde. Daraufhin wurde ein Eventdiagramm erstellt, um eine Übersicht über alle Events zu bieten. Das Diagramm zeigt auf, von wo welche Events stammen.

## 7.2 Lösungsprüfer

Die Arbeit am Lösungsprüfer war hauptsächlich eine Theoretische. Es musste ein durchdachtes Konzept erarbeitet werden, welches jeden erdenklichen Fall berücksichtigt. Der Aufwand war insgesamt grösser als erwartet. Der Grund dafür liegt einerseits darin, dass eine komplette Metrik für den Vergleich erstellt werden musste. Andererseits stellte sich der erste Ansatz, die Datenstrukturen allgemein

gleich zu behandeln und als Graphen darzustellen, als zu schwieriges, nicht effizient lösbares, Problem heraus. Dann hätte der Lösungsprüfer zwar für alle Datenstrukturen funktioniert, aber das zu lösende Problem, das Subgraph-Isomorphismus-Problem, ist zu komplex.

Die Aufgabe war aber sehr interessant und herausfordernd, denn zum ersten Mal mussten eigenständig Algorithmen entwickelt werden. Die spezielle Herausforderung dabei war, die Theorie zu den Algorithmen zu entwickeln und vor allem zu beweisen. Es ist schade, dass keine Implementierung mehr gemacht werden konnte, da die Zeit zu knapp wurde. Dennoch ist der Lernerfolg gross und die zu Stande gekommene Theorie ist stichhaltig.

### 7.3 Planung

Am Ende konnten leider nicht alle Anforderungen erfüllt werden. Auch sind einige geplante Komponenten noch nicht implementiert. Dies ist darauf zurückzuführen, dass die Anforderungen während der Projektlaufzeit laufend angepasst wurden. Dadurch dass die Anforderungen priorisiert wurden, konnten die wichtigsten dennoch umgesetzt werden. Der Aufwand für den Lösungsprüfer wurde unterschätzt, weshalb es am Ende nicht mehr für eine Implementation gereicht hat. Für zukünftige Aufträge derselben Art muss daher mehr Zeit für einen solchen theoretischen Teil eingerechnet werden.

Es wurde ein Usability-Test etwa in der Hälfte der Zeit durchgeführt. Dieser Test war zwar sehr hilfreich bei der Gestaltung des Benutzerinterfaces, wurde allerdings zu spät durchgeführt. Man hätte die Benutzbarkeit bereits in der ersten Version des GUI-Prototypen testen sollen.

Der Grobplan konnte im Grossen und Ganzen eingehalten werden. Es wurde mehr Zeit für das Programm selbst verwendet, weil einige neue Anforderungen während der Entwicklung hinzukamen. Da dies aber mit dem Auftraggeber jeweils abgesprochen wurde, ist die Verzögerung absehbar gewesen. Die Grobplanung war angemessen, mit einer guten Menge an Restzeit gegen Ende, die schlussendlich auch benötigt wurde.

### 7.4 Ausblick

Die nächsten Schritte in der Entwicklung von Algoria Worksheet wären folgende:

- Die Implementierung des vorgestellten Ablaufs des Lösungsprüfers.
- Der Lösungsprüfer in Algoria einbinden, sodass korrekte Teile grün visualisiert werden, und auf fehlende Komponenten hingewiesen wird.
- Die Entwicklung von Konzepten für weitere Datenstrukturen.
- Ein Interface für die Dozierenden, um Einstellungen für den Lösungsprüfer eintragen zu können.
- Die Datenstruktur für Aufgaben enthält im Moment eine Variable für die Benutzerrechte. Diese werden zur Zeit noch nicht verwendet. Es ist vorstellbar, dass die Dozierenden dort einstellen können, ob die Studierenden die Lösung ansehen dürfen. Weitere Benutzerrechte sind denkbar.

## 8 Ehrlichkeitserklärung

Hiermit bestätigen die Autoren, diese Arbeit ohne fremde Hilfe und unter Einhaltung der gebotenen Regeln erstellt zu haben.

**Reto Frey**

---

Ort, Datum  
Unterschrift

**Kevin Zogg**

---

Ort, Datum  
Unterschrift

# Literaturverzeichnis

- [1] Aufgabenstellung P6 Algoria Worksheet  
*Christoph Stamm (2011)*
- [2] Exorciser: Automatisches Generieren und Korrigieren von strukturierten Übungen zur theoretischen Informatik [online]  
<http://www.swisseduc.ch/informatik/exorciser/> 2011.
- [3] HTML Help Workshop and Documentation [online]  
<http://www.microsoft.com/download/en/details.aspx?displaylang=en&id=21138>  
2011.
- [4] C#-Präprozessordirektiven [online]  
<http://msdn.microsoft.com/de-de/library/ed8yd1ha%28v=vs.80%29.aspx> 2011.
- [5] Übersicht über Drag & Drop [online]  
<http://msdn.microsoft.com/de-de/library/ms742859.aspx> 2011.
- [6] An algorithm for subgraph isomorphism  
*Julian R. Ullman (1976); Journal of the ACM 23: 31–42*
- [7] An Algorithm Portfolio for the Sub-Graph Isomorphism Problem  
*Roberto Battiti, Franco Mascia (2007); SLS'07 Proceedings of the 2007 international conference on Engineering stochastic local search algorithms: designing, implementing and analyzing effective heuristics: 106-120*
- [8] A graph distance metric based on the maximal common subgraph  
*Horst Bunke, Kim Shearer (1998); Pattern Recognition Letters Volume 19 Issue 3-4, March 1998*

# Abbildungsverzeichnis

4.1	Datenstrukturen der Aufgabenblätter . . . . .	16
5.1	MVVM Design . . . . .	19
5.2	Speicherauslastung Algoria Instanzen . . . . .	20
5.3	AvalonDock . . . . .	22
5.4	Klassendiagramm . . . . .	23
5.5	Algoria Datenstruktur und Kontrollelement . . . . .	25
5.6	Beispiel <i>RichTextBox</i> . . . . .	26
5.7	Beispiel <i>FlowDocumentViewer</i> . . . . .	27
5.8	Vorschau in der Seitenleiste . . . . .	28
5.9	<i>MultipleChoiceAdminView</i> mit mehreren korrekten Antworten . . . . .	29
5.10	Mauszeigerposition vor und nach dem Klick der <i>MoveUp</i> -Schaltfläche . . . . .	30
5.11	<i>MultipleChoiceView</i> mit mehreren korrekten Antworten . . . . .	31
5.12	<i>MultipleChoiceView</i> mit einer korrekten Antwort . . . . .	31
5.13	Diagramm mit verwendeten <i>Events</i> in Algoria Worksheet . . . . .	33
5.14	Eingefügte Datenstruktur in Microsoft Word . . . . .	36
5.15	Drag & Drop über Instanzgrenzen hinweg . . . . .	38
6.1	Vergleich von sortierten Arrays . . . . .	45
6.2	Ausgangslage des Algorithmus . . . . .	49
6.3	Beispiel: Swapgruppen . . . . .	50
6.4	Schematische Darstellung möglicher Swaps . . . . .	51
6.5	Swapgruppe als Ringliste . . . . .	52
6.6	Ablauf des Lösungsprüfers . . . . .	53

# Tabellenverzeichnis

3.1	Funktionale Anforderungen . . . . .	10
3.2	Nicht-funktionale Anforderungen . . . . .	11
3.3	Dozierenden Persona . . . . .	12
3.4	Studierenden Persona . . . . .	13
5.1	Methoden für Drag & Drop Ereignisse aus [5] . . . . .	39
6.1	Operationen der Ähnlichkeits-Metrik . . . . .	47

# Codeausschnitte

5.1	Version flag . . . . .	21
5.2	Abfrage der VERSION-Variable . . . . .	21
5.3	RichTextBox Inhalt laden . . . . .	27
5.4	Vorschau VisualBrush . . . . .	28
5.5	Einfügen von Checkbox / Radiobutton im <i>MultipleChoiceElement</i> . . . . .	30
5.6	Einfügen von Checkbox / Radiobutton im <i>MultipleChoiceElement</i> . . . . .	31
5.7	Deklarieren des Lösch-Events und <i>Delegates</i> . . . . .	32
5.8	Deklarieren des Lösch-Events und <i>Delegates</i> . . . . .	32
5.9	Registrieren der Löschmethode auf das Löschereignis . . . . .	33
5.10	Löschmethode der View . . . . .	33
5.11	Kopieren von Datenstrukturen in die Zwischenablage . . . . .	34
5.12	Löschen von Datenstrukturen für das Ausschneiden . . . . .	35
5.13	Einfügen von Datenstrukturen aus der Zwischenablage . . . . .	35
5.14	Einfügen von Datenstrukturen aus der Zwischenablage . . . . .	36
5.15	Registrieren des <i>CommandBindings</i> für das Kopieren . . . . .	37
5.16	Starten von Drag & Drop . . . . .	39
5.17	Drag & Drop Effekte . . . . .	40
5.18	Aktive Effekte über Instanzgrenzen übertragen . . . . .	40
5.19	Entfernen nach Verschieben über Instanzgrenzen . . . . .	40
5.20	Animation der Seitenleiste starten . . . . .	41
5.21	Animation beendet . . . . .	42
5.22	Öffnen der Startseite der Hilfe . . . . .	43
5.23	Einfügen der <i>HelpString</i> -Variable . . . . .	43
5.24	Deklarieren der <i>HelpString</i> -Kennung . . . . .	44
5.25	Öffnen der Startseite der Hilfe . . . . .	44
5.26	Öffnen der Startseite der Hilfe . . . . .	44
6.1	Algorithmus in Pseudocode . . . . .	49