Programm zur Visualisierung von benutzerdefinierten Algorithmen auf klassischen Datenstrukturen

---

Manuel Selenka

20.12.2021

Version: 1.0

Johannes Gutenberg-Universität

FB 08 - Informatik
Institut für Informatik
Clean Thesis Group (CTG)

Bachelorarbeit

Programm zur Visualisierung von benutzerdefinierten Algorithmen auf klassischen Datenstrukturen

Manuel Selenka

1. Rezensent Dr. Stefan Endler
FB 08 - Informatik
Johannes Gutenberg-Universität

2. Rezensent Dr. Frank Fischer
FB 08 - Informatik
Johannes Gutenberg-Universität

Betreuer Dr. Stefan Endler

20.12.2021

Manuel Selenka

Programm zur Visualisierung von benutzerdefinierten Algorithmen auf klassischen Datenstrukturen

Bachelorarbeit, 20.12.2021
Rezensenten: Dr. Stefan Endler und Dr. Frank Fischer
Betreuer: Dr. Stefan Endler

Johannes Gutenberg-Universität
Clean Thesis Group (CTG)
Institut für Informatik
FB 08 - Informatik
Saarstraße 21
55122 Mainz

Abstrakt

Die vorliegende Arbeit befasst sich mit der Erstellung eines Programmes zur Visualisierung von Datenstrukturen. Diese Anwendung mit dem Namen Älgorithm Visual Studioërmöglicht die Erstellung von benutzerdefinierten Algorithmen auf klassischen Datenstrukturen wie Variablen, Arrays und verketteten Listen. Der Benutzer bindet eigene Algorithmen dynamisch in das Programm ein. Durch eine ausgearbeitete Struktur kann die Operationsliste und zugehörige Visualisierung eingesehen werden. Dabei werden vorgefertigte Methoden benutzt, die im Hintergrund die Darstellung und Manipulation der Daten übernehmen.
Um die zuvor genannten Aspekte in einer Software zu verwirklichen, wurden zunächst das Anforderungsdokument und eine allgemeine graphische Oberfläche in Form von Mockups erstellt. Aus diesen beiden bildete sich eine detaillierte Strukturierung und ein Aufbau der Software durch UML-Diagramme. Zusammen mit den theoretischen Grundlagen der verwendeten Datenstrukturen dienten alle Komponenten als Vorlage und Hilfsmittel für die nachfolgende Implementierung.
Das Programm ist durch seine Struktur darauf ausgelegt, zukünftige Erweiterungen und neue Datenstrukturen aufzunehmen. Die erfolgreiche Umsetzung des erarbeiteten Konzepts wird in den kommenden Kapiteln beschrieben.

Inhaltsverzeichnis

1 Einleitung

In der nachfolgenden Ausarbeitung wurde auf Grund besserer Lesbarkeit auf das Gendern verzichtet.

1.1 Motivation

Ein wichtiger Bestandteil während eines Studiums oder einer Ausbildung im IT-Bereich ist das Programmieren. Dabei beginnt man meist, kleine Algorithmen auf klassischen Datenstrukturen wie Arrays oder Bäumen zu entwerfen. Um seinen Programmentwurf zu testen, muss er in einer passenden Umgebung ausgeführt werden. In den meisten Fällen wird auf eine integrierte Entwicklungsumgebung zurückgegriffen. Innerhalb dieser können über die Konsole dem Benutzer Informationen ausgegeben werden. Ist das Ergebnis des Algorithmus falsch beginnt die Fehlersuche. Damit man die fehlerhaften Stellen findet, müssen alle Anweisungen des Algorithmus durchlaufen und überprüft werden. Denn die letztendliche Ausgabe des Algorithmus wird meist durch Hilfsstrukturen und -variablen bedingt und erzeugt. Damit der Programmierer über die Daten in der Ausführung des Algorithmus den Überblick behält, wird oft eine einfache Konsolenausgabe verwendet. Hier ist meist eine zusätzliche Beschriftung der Ausgabe notwendig, falls die Menge innerhalb der Konsole die Übersicht hemmt. Neben der Nutzung der Konsole kann über eine integrierte Entwicklungsumgebung in der Regel auf einen Debugger zurückgegriffen werden. Dieser zeigt alle Variablen, Methoden, Klassen und Instanzen an, unabhängig ihrer Wichtigkeit.
Im Gegensatz dazu kann eine Visualisierung aller genutzten Objekte eines Algorithmus logische Fehler schneller aufzeigen. So kann leichter der Überblick behalten werden über alle Hilfsstrukturen und -variablen und deren Veränderungen. Will man sich diese Datenstrukturen visualisieren lassen, sind externe Programme notwendig. Abseits davon können durch zusätzliche Bibliotheken visuelle Objekte erzeugt werden, um die einzelnen Elemente des Algorithmus abstrakt darstellen. Eine Erzeugung dieser Anzeige ist dadurch aufwendig und bedarf zusätzlicher Arbeit und Platz in Form von Programmcode. Da die Visualisierung aber besonders für Programmieranfänger hilfreich sein kann, ist die Erstellung der Anzeige für sie oft zu kompliziert.
Um diese Nachteile zu vermeiden, ist ein Programm notwendig, welches die Veränderungen des Algorithmus und alle für den Programmierer wichtigen Datenstrukturen

anzeigt. Weiterhin sollen die einzelnen Operationen und deren Umsetzung auf der Datenstruktur in verständlicher Form abgebildet werden. Dies erleichtert dem Benutzer eine klare Übersicht darüber, wie sein Algorithmus arbeitet und die Daten verändert. Eine minimale und deutliche Anzeige der Datenstrukturen ist hierfür unabdingbar.
Die nachfolgende Ausarbeitung behandelt ein neues, eigens erstelltes Programm mit dem Namen Älgorithm Visual Studio". Diese Anwendung zielt darauf ab, dem Programmierer eine einfache und übersichtliche Anwendung zu bieten, um seine Arbeit zu kontrollieren und zu testen. Die Einbindung des Algorithmus soll dabei leicht und möglichst intuitiv sein, ohne dabei Abstriche in der Funktionalität zu machen. Hierbei muss der Benutzer die bereitgestellten Bibliotheken verwenden, damit eine Visualisierung der genutzten Funktionen vollzogen werden kann. Das bedeutet, dass der Anwender die Funktionen, welche die Datenstrukturen manipulieren, benutzen muss, um einen Algorithmus zu schreiben. Im Hintergrund sollen diese manipulierenden Methoden eine Anzeige der Struktur und deren Veränderungen erzeugen. Programmierfehler sind bei dem Programmentwurf selbstverständlich. Nur selten kann ein Entwickler einen fertigen Algorithmus direkt fehlerfrei vorzeigen. Ein Abfangen der Fehler und eine möglichst genaue Darstellung dieser ist erforderlich, damit der Programmierer den Fehler in seinem Entwurf ausbessern kann.

1.2 Problemstellung und theoretischer Hintergrund

Eine weit verbreitete Vorlesung im Informatik-Studium behandelt das Thema der klassischen Datenstrukturen. Dazu gehören beispielsweise Arrays, verkettete Listen, Hash-Tabellen, Bäume, Graphen, Stapelspeicher oder der Heap. Ein allgemeines Verständnis und korrekter Umgang mit diesen Strukturen ist dabei wichtig, da sie immer wieder benutzt und gebraucht werden. Sie verkörpern einen fundamentalen Teil der Informatik, zu dem ebenso Algorithmen gehören, die auf den Datenstrukturen ausgeführt werden.
Die wohl bekanntesten Operationen stellen die Sortieralgorithmen dar, wie zum Beispiel Bubblesort, Mergesort und Heapsort. Sie werden meist auf Arrays, verketteten Listen oder Heap-Speichern angewandt. Damit aus Theorie Praxis wird, ist es von Vorteil, Algorithmen auf zugehörigen Datenstrukturen anzuwenden, um ein tieferes Verständnis der Thematik zu erhalten.
Einem Programmieranfänger stehen zunächst zwei Arten der Kontrolle eines Algorithmus zur Verfügung. In erster Linien helfen integrierte Entwicklungsumgebungen (Englisch: Integrated Development Environment kurz ÏDE") den Nutzern beim Erstellen und Testen der Verfahren. Die Schwierigkeit besteht darin, die sich ändernden Elemente und deren wechselnde Positionen innerhalb der Datenstruktur zu über-

wachen. Gleichzeitig sind die Variablen des Verfahrens genauso wichtig, da diese mit den Iterationen und lokalen Hilfsmethoden die letztendliche Veränderung der Datenstruktur bewirken. Um während der Ausführung der Befehle einen Überblick darüber zu behalten, welche Veränderungen vorgenommen werden, muss man entweder diese aufwendig über die Konsole ausgeben lassen oder den Debugger benutzen. Die Konsole ist dabei ein zu oft genutztes Hilfsmittel, obwohl die Arbeit dafür viel höher ist als mit einem Debugger. Dieser zeigt alle Variablen, Instanzen, etc. des Programmprojektes an. Der Nachteil hierbei liegt an der Darstellung der Daten. Zwar sind dort im Endeffekt alle Hinweise zu finden, doch die Suche nach den relevanten Informationen ist hier der entscheidende Faktor. Es wird alles in der gleichen Form angezeigt, unabhängig davon, wie wichtig die Information für den Benutzer ist. Es können hierfür Daten dargestellt werden, die nicht zu dem Algorithmus in Bezug stehen und damit unnötigen Platzfürs Auge beanspruchen. Die Datenstruktur wird außerdem nicht in dem Maße verdeutlicht angezeigt wird, wie es für den Programmierer notwendig wäre. Benutzt der Anwender einen binären Baum, ist Kontrolle der Daten auf ihre Korrektheit aufwendig. Diese graphischen Informationen könnten beispielsweise in einem Array gespeichert werden, was eine Kontrolle der Daten erschwert. Dies kann demnach für unterschiedliche Datensätze und Datenstrukturen besonders mühsam und ungenau sein. Die meisten Debugger können zudem nicht die Veränderungsschritte der Datenstruktur umkehren, was eine zusätzliche Hilfe bedeutet. So könnten Fehler besser erkannt und letztendlich Zeit gespart werden.
So stoßen diese beiden Möglichkeiten schnell an ihre Grenzen und verhindern ein effektives Programmieren, da viel zu lange nach Fehlern gesucht werden muss. Für Listen und Arrays kann der Debugger eine gute Hilfe sein, jedoch bei komplexeren Datenstrukturen wird das Vorstellungsvermögen schnell überfordert, wie beispielsweise bei Graphen und Bäumen. Dazu sind externe Visualisierungsprogramme oder zusätzliche Bibliotheken notwendig, die sich auf die Darstellung dieser Daten spezialisieren.

1.3 Vergleich von bestehenden ähnlichen Programmen

Diese Passage dient dem Vergleich mit bereits existierenden Programmen, die sich mit der Visualisierung klassischer Datenstrukturen und deren Operationen beschäftigen. Ist eine neue Anwendung überhaupt erforderlich oder gibt es schon eine entsprechende Bibliothek für die Visualisierung von Datenstrukturen? Diese Frage soll beantwortet werden und einen Ausblick geben, worauf die Anwendung dieser Arbeit eine Lösung geben kann.
Die Website "visualgo.net"hat sich darauf spezialisiert, vorgefertigte fundamentale

Algorithmen von Datenstrukturen zu visualisieren. Dabei wählt man eine Struktur und einen zugehörigen Algorithmus aus. Die Daten bzw. Zahlen werden als Balken repräsentiert. Beim Durchlauf des Algorithmus erscheint der Pseudocode und verdeutlicht, an welcher Stelle sich die Animation im Code befindet. Weiterhin verfügt die Visualisierung über eine manuelle oder eine zufällig vorgefertigte Eingabe der Anzeigedaten. Zusätzlich kann ein Training für jede Datenstruktur vollzogen werden, welches sich dem Lernen der Thematik widmet. Hier ist eine Vielzahl an Strukturen implementiert wie Arrays, verkettete Listen, Binäre Heaps, Bäume, Graphen und Hash-Tabellen. Auf diesen können Algorithmen wie minimaler Spannbaum, Insertionsort oder Problem des Handlungsreisenden angezeigt werden. Eine Visualisierung eines eigenen Algorithmus ist dabei nicht möglich, sondern beschränkt sich auf die Darstellung dieser klassischen Methoden und deren Funktionsweise. Bei verketteten Listen ist zwar eine manuelle Methodenausführung möglich, jedoch keine Konzipierung eines kompletten Algorithmus. Grund dafür ist, dass immer nur der momentane Zustand der Liste dargestellt wird und die Methoden nicht gespeichert werden. Damit können nur atomare Operationen genutzt werden, die rein darauf abzielen, Konzept und Funktion der Datenstruktur und Methoden zu verstehen.[5]
Einen ähnlichen Schwerpunkt legt die Internetseite von David Galles, bei der wieder typische Methoden der bekanntesten Datenstrukturen dargestellt werden. Hierbei gibt es mehr Anpassungsmöglichkeiten der Visualisierung und eine generell größere Auswahl an Algorithmen als bei "visualgo.net". Ebenso stehen mehr Datenstrukturen wie z.B. geometrische Algorithmen zur Verfügung.[4] Die Vor- und Nachteile teilen sich die beiden Webanwendungen. Zu dieser Kategorie zählt auch das Github-Verzeichnis Älgomation", das genau wie die vorherigen Webseiten die Algorithmen visualisiert. Das Verzeichnis ist dagegen mit Algorithmen weniger bestückt und weist keinerlei neue Features im Vergleich zu den anderen vor. Zugleich kann kein eigener Datensatz für die Visualisierung verwendet werden. Außerdem erfordert das Github-Verzeichnis eine aufwendigere Installation.[9]
Einen ganz anderen Ansatz verfolgt dagegen die Internetseite Älgorithm-Visualizer".[1] Dieses Projekt, welches auch als Github-Verzeichnis downloadbar ist, verbindet als erstes die Visualisierung und das Programmieren eines Algorithmus. Auf der Internetseite lassen sich, wie bei den vorher genannten Anwendungen, vorgefertigte Algorithmen auf fundamentalen Datenstrukturen anzeigen. Daneben wird der Code angezeigt, der diese Visualisierung erzeugt. Außerdem wird eine Befehlsliste erstellt, welche die Methoden auf der Datenstruktur zusammenfasst. Die Auswahl der Algorithmen ist dabei ausreichend. Das Programm punktet vor allem damit, dass hier in einem Editor codiert werden kann. Dazu müssen die vorgefertigten Bibliotheken benutzt werden, um eine Visualisierung der Daten zu erzeugen. Um die Funktionen allerdings richtig zu gebrauchen, muss sich der Benutzer die Dokumentation der Visualisierungsklasse aneignen. Ein Nachteil ist, dass man jede Animation selbst aufrufen muss und sie nicht automatisch im Hintergrund abläuft. Hat man die Funktionalität und Vorgehensweise der Anwendung durchschaut, kann dies ein mächtiges

Werkzeug sein. Der eigentliche Code des Algorithmus, den sich der Benutzer erstellen will, wird durch die zu benutzenden Visualisierungsfunktionen sehr umfangreich und verbirgt die eigentliche Thematik. Allerdings ist eine direkte Umsetzung und Codierung eines Algorithmus möglich. Die Programmiersprachen, in denen der Algorithmus geschrieben werden kann, sind: Java, Javascript und C++. Diese Vielfalt ist ein großer Gewinn und hilft den meisten Nutzern auf eine bereits bekannte Sprache zurückzugreifen. Der Code muss in einer gewissen Form vorliegen, damit das Programm diese verarbeiten kann. Um Fehler zu beheben, werden diese nur kurz auf der Internetseite angezeigt. Die kurze Meldung erschwert jedoch die Ursache des Fehlers zu erkennen, was besonders für Programmieranfänger problematisch ist. Daher eignet sich dieses Werkzeug möglicherweise vermehrt für fortgeschrittene und erfahrenere Programmierer.
Um die Nachteile eines zusätzlichen Umgangs mit Visualisierungselementen zu vermeiden, soll das hier beschriebene Programm eine Antwort bilden. Eine abstrakte Datenstruktur soll dem Nutzer zur Verfügung stehen, die im Hintergrund die Aktionen hervorruft und eine Darstellung der Strukturen ermöglicht. Der Anwender soll nur mit den atomaren Operationen einer Datenstruktur umgehen und sich diese durch das Programm visualisieren lassen. Zudem sollen Fehler deutlicher und verständlicher dem Benutzer vermittelt werden, damit dieser bei dem Erstellungsprozess seines Algorithmus effektiv vorankommt. Dafür ist das Programm eingeschränkt auf eine Programmiersprache: Java. Diese stellt einen fundamentalen Teil der Programmiersprachen dar. Besonders für Architektur- und Designmuster ist diese Sprache von großer Bedeutung, welche gleichermaßen in vielen Systemen, Webseiten oder Apps verwendet wird. Eine Erweiterung auf andere Sprachen kann jedoch für die Zukunft offen gehalten werden.

1.4 Zielsetzung

Die Anwendung Älgorithm Visual Studio"dient in erster Linie der Visualisierung von Datenstrukturen. Sie eignet sich besonders für Neulinge, aber auch fortgeschrittene Entwickler, die auf klassischen Datenstrukturen arbeiten und dort Algorithmen entwickeln wollen. Dabei sollten neben den Funktionen der vorgefertigten Klassen die abstrakten Datenstrukturen und integrierten Methoden verwendet werden. Somit wird eine direkte Visualisierung der Befehle ermöglicht, ohne diese explizit zu erstellen. Wird bei einem Algorithmus keine bereitgestellte Datenstruktur verwendet, ist keine Visualisierung der Daten durchführbar. Bereits vorgefertigte klassische Datenstrukturen von Java werden demnach nicht visualisiert und sind bei der Benutzung dieses Programmes nicht zu empfehlen. Hier muss beispielsweise die eigens implementierte Klasse Array statt der Listen von Java verwendet werden. Diese abstrakten Datenstrukturen des Programmes beinhalten alle zugehörigen Funktionen

und Informationen, sodass keine weitere Struktur notwendig ist.
Die Software, die in dieser Arbeit beschrieben wird, bietet eine Lösung für die Probleme des Absatzes Problemstellung und theoretischer Hintergrund". Der Benutzer hat die Möglichkeit, mit geringem Aufwand einen eigenen Algorithmus in das System dynamisch einzubinden. Dabei muss er in einer entsprechenden Form vorliegen wie bei dem Programm Älgorithm-Visualizer". Dieser wird innerhalb der Anwendung Älgorithm Visual Studioäusgeführt und auf den zugehörigen Datenstrukturen visualisiert. Der Benutzer hat neben der Auswahl des Algorithmus und entsprechenden Datenstrukturen die Möglichkeit, den Programmcode einmal komplett oder Schritt für Schritt durchlaufen zu lassen. Letzteres bedeutet, dass nach jeder Veränderung auf der Datenstruktur der Algorithmus pausiert und auf die manuelle Fortsetzung durch den Nutzer wartet. Zudem soll es möglich sein, die Anpassungen der Struktur umzukehren und die getätigten Funktionen schrittweise zurückzulaufen. Die Visualisierung geschieht über ein separates Fenster, groß genug um alle nötigen Daten anzuzeigen. Gleichzeitig werden dem Benutzer alle Methoden aufgelistet, die er in seinem Algorithmus ausgewählt hat. Für eine Fehlerbehandlung sollen möglichst alle Fehlerarten abgefangen und visualisiert werden. Eine zusätzliche Beschreibung der Fehler ist angestrebt.
Das Programm testet die Korrektheit von benutzerdefinierten Algorithmen und hilft bei deren Erstellungsprozess. Nach der Kompilierung und Kontrolle des Algorithmus durch eine übergeordnete Komponente wird dieser in dem Fenster angezeigt. Durch die Visualisierung sollen so schneller logische Programmfehler entdeckt werden.

1.5 Strukturierung des Inhalts der Arbeit

Die nachfolgende Arbeit beginnt mit der Beschreibung der Konzepte von Algorithmen und den verwendeten Datenstrukturen. Darauf folgt die Auflistung der angestrebten Anforderungen an das Programm Älgorithm Visual Studio". Das Kapitel richtet sich nach dem typischen Anforderungsdokument aus dem Softwareentwicklungsprozess. Anschließend widmet sich das Kapitel 4 der Architektur dieser Software. Hier werden die verwendeten Entwurfsmuster und ihre tatsächliche Anwendung in der Software beschrieben, unterstützt von UML- und Sequenzdiagrammen. Im darauffolgendem Kapitel wird auf die Implementierung eingegangen. Zentrales Thema ist die Strukturierung und die Herausforderungen des Erstellungsprozesses. Es werden u.a. die Fragen beantwortet, welche Probleme aufgetreten sind und wie eine Erweiterung der Anwendung ermöglicht wurde. Den Abschluss bildet die Zusammenfassung mit einem Fazit und Ausblick auf vielleicht kommende Arbeiten an diesem Programm.

2 Theoretische Grundlagen

Datenstrukturen und Algorithmen sind eng miteinander verbunden. Zum einen werden Algorithmen auf Datenstrukturen ausgeführt, zum anderen können Datenstrukturen Algorithmen als Teile in sich enthalten, um zum Beispiel Daten zu verarbeiten.[8](S.1) Beide Komponenten sind Teil dieser Arbeit. Die Datenstrukturen werden durch Algorithmen des Benutzers visualisiert. Um diese Verbindung und die Problematik beim Erstellungsprozess der Software besser zu verstehen, werden beide Elemente in diesem Kapitel erläutert. Zudem werden im Folgenden die Grundlagen zu Datenstrukturen und den letztendlich verbauten Strukturen des Programmes auf theoretischer Ebene definiert.

2.1 Algorithmen

Algorithmen sind systematische Ausführungen von Anweisungen und elementaren Aktionen.[6](S.75) Diese führen ihre Operationen auf Datenstrukturen aus. Die Datenstrukturen wiederum enthalten Algorithmen, um ihre Daten zu verarbeiten. Daher sind beide eng miteinander gekoppelt und bedingen sich einander.[8](S.1) Um das Konzept von Datenstrukturen zu verstehen, hilft es ebenso Algorithmen näher zu betrachten. Die Operationen auf den Datenstrukturen sind wiederum Teil eines Algorithmus.
Als Einstieg in die Thematik dient die Mathematik. Diese beschreibt auf einer abstrakten Ebene eine spezifizierte Funktion.[8](S.1) Eine solche Funktion kann die Berechnung des größten gemeinsamen Teilers zweier Zahlen sein. Das Beispiel (2.1) definiert auf der mathematisch abstrakten Ebene, ob eine Liste leer ist. Diese Funktion kann wiederum von einem algorithmischen Verfahren (2.2) aus Anweisungen beschrieben werden. Diese sagen, mit welchen Instruktionen das Beispiel (2.1) gelöst werden kann. Mit der Darstellung dieses Codes ist keine genaue Sprache gemeint, sondern eine übergeordnete abstrakte Pseudocodesprache. Diese ist unabhängig von Programmiersprachen und kann in fast jeder Sprache implementiert werden. Genauso dient sie der strukturierten und einfachen Darstellung von Befehlen. So soll jeder verstehen können, wie das Problem durch den Algorithmus gelöst wird. Diese algorithmische Spezifikation wird nun in der untersten Ebene, der Programmiersprache (2.3), implementiert.[8](S.1-2) Dies geschieht u.a. mit Prozeduren, Anweisungen und Schleifen. Diese drei Abstraktionsebenen werden bei der Algo-

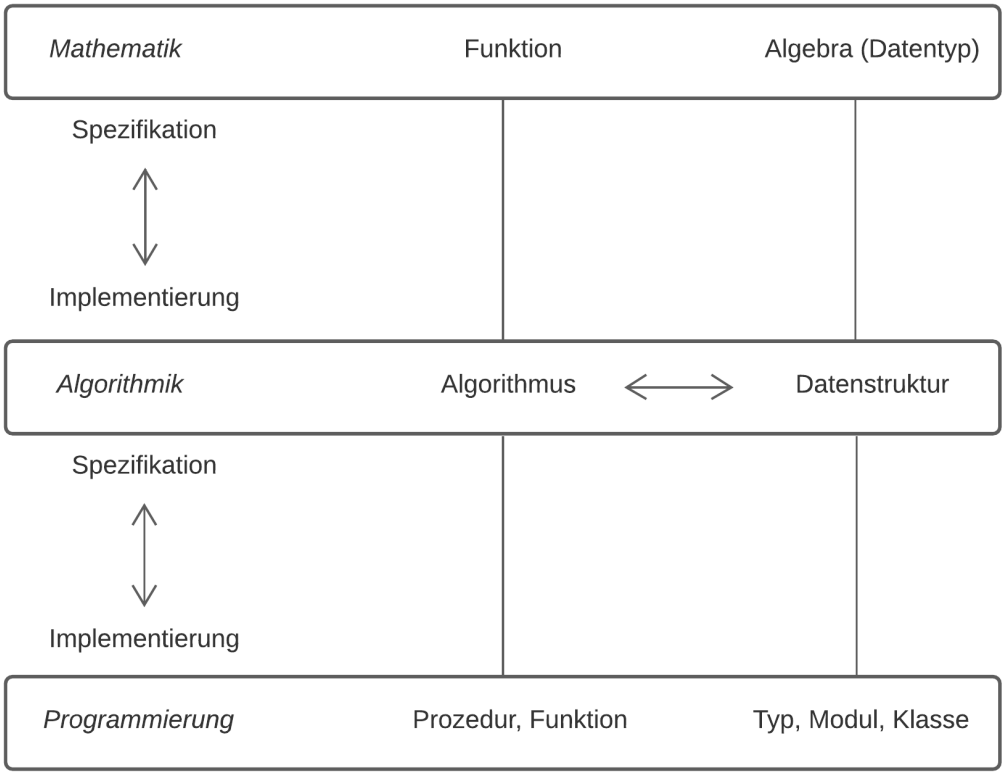

Abbildung 2.1: Schaubild aus [8] S.1 Abbildung 1.1: Abstraktionsebenen von Algorithmen und Datenstrukturen

rithmuserstellung durchlaufen. Die oberste Ebene der Mathematik wird aber oft übersprungen und direkt mit dem Pseudocode begonnen. Diese oberste Schicht stellt formal fest, was berechnet wird. Das kann jedoch gleichermaßen in textlicher bzw. umgangssprachlicher Form bereits definiert worden sein. Die Algorithmik-Ebene beschreibt, wie das Problem gelöst werden kann. Die Programmiersprache ist dann die letztendliche Umsetzung des Problems (siehe Abb. 2.1.).[8](S.3) Welche Probleme durch eine Algorithmik bewältigt werden, kann die Häufigkeit, in der ein Problem auftritt, entscheiden.[11](S.5) Ähnlich zu der Modularisierung wird ein immer wieder auftretendes Problem in einer eigenen Funktion gekapselt, um Redundanz und Ressourcen zu sparen. Das nachfolgende Beispiel ist eine Umsetzung der Funktion, die testet, ob ein Liste leer ist. Zuerst ist die algebraische Darstellung zu sehen. Danach folgt der Pseudocode. Hierbei ist es wichtig, dass nicht Elemente verwendet werden, die speziell in einer Programmiersprache existieren. Unter (2.3) ist die Umsetzung in der Programmiersprache Java dargestellt.

$$leereListe : F(\mathbb{Z}) \rightarrow BOOL$$
$$leereListe(S) = \{ \begin{array}{ll} true & \text{falls Länge(S)} = 0 \\ false & \text{falls Länge(S)} > 0 \end{array} \} \tag{2.1}$$

```
Algorithmus leereListe(S)
{Eingabe ist eine Liste mit Integerwerten der Länge n}
var ergebnis : bool;
ergebnis := true;
if S.length > 0 then ergebnis := false end if
return ergebnis;
```
(2.2)

```
public boolean leereListe(int[] s){
    boolean b = true;
    if (s.length > 0){
    b = false;}
    return b;}
```
(2.3)

Ein Algorithmus hat wie in dem vorherigen Beispiel eine Eingabe, die eine Ausgabe durch Instruktionen bewirkt. Neben der Eingabe von Daten, auf denen operiert wird, können es ebenso Parameter sein, die den Algorithmus genauer einstellen. Für eine Funktion, die überprüft, ob ein Element in einer gegebenen Menge enthalten ist, kann dies eine weitere ganze Zahl sein, welche die minimale Auftrittsanzahl des

gesuchten Integers definiert.
Allgemein gibt es unendlich viele Arten, wie man ein Problem auf allen Ebenen löst. Besonders beim Programmieren ist die Art der Realisierung wichtig. Auf der einen Seite sollte ein Algorithmus zwar klar und verständlich definiert sein, auf der anderen Seite ist dies jedoch nicht immer die effizienteste Umsetzung des Problems. Die Ressourcen, also Speicherplatz und Laufzeit, sind hierfür wichtige Faktoren. Nicht immer ist der einfachste Algorithmus der effektivste, weswegen ein Abwägen der jeweiligen Seiten entscheidend ist.[8](S.5) Hinzu kommt, dass der Leser des Algorithmus ihn schnell verstehen soll. Dieser Aspekt muss gleichermaßen bei einem Algorithmusentwurf berücksichtigt werden. Aus diesem Grund ist eine gute Beschreibung mit Kommentaren und selbstsprechenden Variablen, Funktionen und Klassen unabdinglich. Sie hilft nicht nur dem Kontrolleur, das Konzept des Algorithmus zu verstehen, sondern auch zukünftigen Personen, die daran arbeiten.
Wenn ein Problem algorithmisch gelöst werden soll, müssen alle möglichen Arten der Ausgabe berücksichtigt werden. Dazu muss der Definitions- und Wertebereich einer Funktion klar definiert werden. Gemeint sind alle möglichen Eingabe- und Ausgabearten und deren Werte. Falls manche Teile des Definitionsbereichs nicht unterstützt werden, müssen dieses dokumentiert oder algorithmisch vermittelt werden. Ein kleiner Algorithmus kann dadurch schnell umfangreich werden, obwohl das eigentliche Problem in wenigen Zeilen berechnet wäre. Da es unendlich viele Möglichkeiten der Problembewältigung gibt, ist das allgemeine Design und die Struktur des Algorithmus zu beachten. Dabei ist wichtig zu wissen, welche Anweisungen und Instruktionen besonders ressourcenintensiv sind. Eine Reduzierung von Variablen, die nicht verwendet werden, ist trivial. Die Reduktion der Laufzeit dagegen stellt eine größere und effektivere Herausforderung dar.
Die Laufzeit eines Algorithmus wird generell in drei verschiedenen Arten berechnet. Der Best Case ist der minimale Bearbeitungsaufwand einer Funktion. Der Worst Case bildet den maximalen, der Average Case den durchschnittlichen Aufwand. Diese werden nicht zeitlich gemessen, da diese abhängig der Hardware und Software eines Computers sind. Stattdessen wird die Eingabegröße eines Programmes in Beziehung zur Anzahl der Instruktionen gesetzt, die auf ihr ausgeführt werden. Für das Suchen eines Elementes in einer Menge wäre der Best Case, wenn sich das gesuchte Element an der ersten Position der Menge befindet und direkt zurückgegeben werden kann. Der Worst Case wäre, wenn das Element nicht zu finden ist oder das letzte Element das Gesuchte ist. Die durchschnittliche Laufzeit dagegen lässt sich schwerer berechnen. Dabei ist nicht die Mitte aus den beiden anderen Laufzeiten gemeint, sondern wie häufig eine gewisse Eingabe im Durchschnitt auftritt. Das bedeutet, jede Eingabe muss statistisch gewichtet und in Beziehung zu deren Laufzeit gebracht werden. Die einfache Gleichverteilung ist dabei nicht immer die exakte Lösung des Problems, weswegen diese Laufzeit besonders aufwendig zu kalkulieren ist.[8](S.10) Das hat zur Folge, dass der Fokus meist auf der Worst Case-Laufzeit liegt. Damit kann man den maximalen Aufwand eines Algorithmus abschätzen. Diese Laufzeit-Notation ist

in (2.4) dargestellt und besagt, dass eine Funktion f maximal so schnell wächst wie eine Funktion g. f ist die Laufzeitfunktion des Algorithmus und $O(g)$ eine Menge von Funktionen. Diese ist wie folgt definiert:

$$\begin{aligned} &\text{O-Notation: Seien f} : \mathbb{N} \to \mathbb{R}^+,\ \text{g} : \mathbb{N} \to \mathbb{R}^+. \\ \text{f = O(g)} : \Leftrightarrow &\exists\, n_0 \in \mathbb{N}, \text{c} \in \mathbb{R}, \text{c} > 0 : \forall\, n \geq n_0\ f(n) \leq c * g(n) \text{ [8](S.11)} \end{aligned} \tag{2.4}$$

Die zugehörige Funktionsmenge $O(g)$ ist dabei festgelegt als:

$$\text{O(g)} = \{\ \text{f}:\mathbb{N} \to \mathbb{R}^+ \mid \exists\, n_0 \in \mathbb{N}, c \in \mathbb{R},\ \text{c} > 0 : \forall\, n \geq n_0\ f(n) \leq c * g(n)\} \text{ [8](S.12)} \tag{2.5}$$

Die wohl teuersten Instruktionen eines Algorithmus sind Schleifen über die Eingabegröße. Besonders sollten die Ineinanderverschachtelung dieser Schleifenart vermieden werden. Eine for-Schleife über die Eingabegröße n würde eine Worst Case-Laufzeit von $O(n)$ bewirken. Zwei ineinandergeschachtelte Schleifen kosten $O(n^2)$ und drei demnach $O(n^3)$. So würde bei einer Eingabelänge von 100 für die drei Varianten ein Aufwand von 100,$O(n^2)$ mit 10 000 und $O(n^3)$ mit 1 000 000 anfallen. Sonstige einfache Instruktionen sind im nachfolgenden Absatz mit $O(1)$ berechnet und daher für die dominierende Größe der Schleife, also n, zu vernachlässigen. Es gilt weiter:

$$\begin{aligned} T_1(n) &= n + 4 = O(n) \\ T_2(n) &= n^2 + n = O(n^2) \\ T_3(n) &= 10000 * n = O(n) \text{ [8](S.12)} \end{aligned} \tag{2.6}$$

Damit beeinflussen konstante Faktoren nicht die Landau-Notation, genauso wenig wie additive Konstanten. Der dominierendste Faktor, wie in Beispiel T_2, ergibt die maximale Laufzeit.[8](S.12)
In der Anwendung dieser Arbeit schreibt der Nutzer Algorithmen in der Programmiersprache Java. Zwar muss die Syntax von Java eingehalten werden, dennoch soll eine Art des pseudosprachenähnlichen Umgangs simuliert werden. Eine weitere Abstraktionsebene mit vorgefertigten Funktionen wird ihm vorgegeben. Jede Operation wird vom übergeordneten Programm so schnell wie möglich erledigt. Die Simulation der Kosten einer Operation kann jedoch für die Visualisierung offen gehalten werden.

2.2 Datenstrukturen

Der Begriff Datenstruktur beschreibt bereits sehr treffend seine Bedeutung: Zum einen beinhaltet er das Wort Daten, welches ein allgemeiner Ausdruck jeglicher Informationen ist. Zum anderen weist der Wortteil Struktur auf eine Organisation

dieser Daten hin. Die Daten können unterschiedlichen Typs und Größe sein. Um einen effizienten und schnellen Zugriff auf die Daten zu ermöglichen, braucht es eine Organisation dieser Daten. Daher wird den Informationen eine Struktur zugewiesen, die für den Anwendungsfall besonders passend sein soll.
Ebenso wie Algorithmen kann man auch Datenstrukturen in einzelne Abstraktionsebenen einteilen. Eine Datenstruktur besteht auf mathematischer Ebene aus einem algebraischen oder abstrakten Datentyp. Diese Spezifikation einer Datenstruktur wird wiederum in einer Programmiersprache implementiert.[8](S.1)

2.2.1 Variablen

In der Mathematik werden wie in der Informatik viele Datenstrukturen verwendet. Eine Variable bildet dabei eine der kleinsten Datenstrukturen. Sie besitzt Kapazität für eine einzige Information. Doch bevor sie verwendet werden kann, muss sie, wie jede andere Datenstruktur auch, erst initialisiert werden. Ein Computer schafft bei der Initialisierung Raum für eine entsprechende Struktur eines bestimmten Typs. Eine boolesche Variable wird beispielsweise im Speicher mit 1 Bit repräsentiert, da sie nur den Wert wahr oder falsch annehmen kann. Ihr Definitionsbereich ist also $\mathbb{D} = \{true, false\}$. Neben einer booleschen Variablen gibt es noch viele weitere Datentypen wie z.B. Strings für Zeichen, Integer für ganze Zahlen, float für Gleitzahlen. Die Anzahl der Datentypen wird von der Softwaresprache determiniert, in der das Programm geschrieben wird. Jeder Datentyp kann an eine Variable angehängt werden und beschreibt deren Datenformat. In dynamisch typisierten Sprachen wie Python muss einer Variable kein Typus hinzugefügt werden; bei statisch typisierten Sprachen wie Java jedoch schon. Das letztendliche Setzen des Wertes der Variablen geschieht im darauf folgenden Schritt. Der Wert muss mit dem Definitionsbereich der Variablen übereinstimmen. Wird so der Wert einer booleschen Variablen auf den Wert "Hallo"gesetzt, führt dies zu einem Programmfehler. Weiterhin ist es möglich, eine Variable mit dem Wert einer anderen gleichen Typs zu setzen oder sich diesen Wert auch ausgeben zu lassen. Die Variable wird durch die Operation löschen wieder entfernt und ihr Speicher freigegeben. Grundsätzlich ist eine Variable eine 1-elementige Datenstruktur mit den allgemeinen Operatoren: erstellen, setzen, lesen und löschen des Wertes und der Variablen. Die maximale Laufzeit der Operationen in Landau-Notation ist immer $O(1)$. Neben diesen übergeordneten Methoden der Variablen gibt es einige weitere Funktionen, um die Werte der Variablen zu manipulieren. Dabei sind mathematische Operatoren wie $+$, $-$, $*$, $/$, $Modulo$ gemeint. Diese Operatoren werden zwischen zwei Zahlen gesetzt und ergeben dadurch einen neuen Wert. Der daraus resultierende Wert kann einen neuen Typ erzeugen wie in Beispiel (2.7.2). In den Gleichungen sind Beispiele für weitere Operatoren auf Inte-

ger, Strings und booleschen Werten aufgeführt. Diese Operatoren sind in gängigen Programmiersprachen integriert, so auch in Java.

1. $integer \times integer \rightarrow integer$ Operatoren: +, -, *, /, Modulo
2. $integer \times integer \rightarrow boolean$ Operatoren: $=, \leq, \geq, <, >, \neq$
3. $boolean \times boolean \rightarrow boolean$ Operatoren: and, or
4. $boolean \rightarrow boolean$ Operator: not
5. $string \times string \rightarrow boolean$ Operatoren: $=, \leq, \geq, <, >, \neq$

Gleichungen entnommen aus [8](S.41) (2.7)

Diese Datenstruktur ist mit den genannten Operatoren in dem Programm verbaut. Der einzige Nachteil einer statisch typisierten Variablen: Der Typ und ihre Größe lassen sich nicht anpassen. Hierfür ist die Erstellung einer neuen Variablen notwendig.

2.2.2 Array

Eine weitere typische und täglich verwendete Datenstruktur ist eine Liste. Praktische Beispiele einer Anwendung wären das Erstellen einer Einkaufsliste oder die Mitschrift einer Vorlesung. Alles wird, meist untereinander, nach und nach niedergeschrieben. Wenn sich die Frage stellt, welches Produkt an dritter Stelle steht, weiß jeder genau, dass der dritte Punkt auf der Einkaufsliste gemeint ist. In der Informatik gibt es zu dieser analogen Liste gleichermaßen ähnliche Datenstrukturen: verkettete Listen und Arrays. Im Prinzip ist ein Array oder eine Liste eine Aneinanderreihung von Variablen gleichen Typs.[2](S.99) Zu Beginn führt man eine Initialisierung der Liste durch, indem der Typ der Elemente sowie der Name des Arrays deklariert wird. In Java wird die Länge der Liste dynamisch angepasst und kann nicht bei der Initialisierung gesetzt werden. Das hier beschriebene Programm verwendet Arrays fester Größe. Der Programmierer kann diese nicht verkürzen oder erweitern. Um das Array zu erweitern, muss ein neues angelegt werden. Die Kosten zur Erstellung des Arrays liegen für n Elemente bei $O(n)$. Jedes Element kann wie die Variable gelesen, gesetzt und gelöscht werden. Zugegriffen wird auf die Elemente über den Index eines Arrays. Begonnen wird dabei üblicherweise mit dem ersten Element eines Arrays A mit $A[0]$.[8](S.42) Die Operation suchen besitzt die Kosten $O(n)$. Dabei müssen maximal alle Elemente durchlaufen werden. Das Lesen geschieht in $O(1)$, nachdem das gelesene Element gefunden wurde. Gleiches gilt für das Löschen eines Elementes des Arrays. Es besteht aus dem Suchen mit $O(n)$ und dem letztendlichen Löschen mit $O(1)$ Kosten. Zudem können zwei Elemente eines Arrays getauscht werden. Diese Operation setzt voraus, dass beide Instanzen in $O(n)$ ermittelt werden. Für das Tauschen muss eine weitere Variable erstellt werden, um

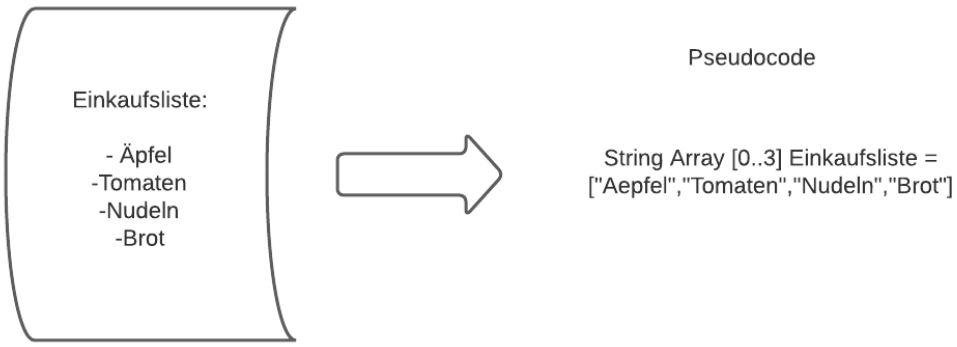

Abbildung 2.2: Umwandlung einer analogen Einkaufsliste in ein digitales Array

einen der beiden Werte zwischenzuspeichern. Zuerst wird der Inhalt des ersten Arrayelements auf eine neue Variable geschrieben, dann das erste Arrayelement auf den Wert des zweiten Elementes gesetzt und abschließend das zweite Arrayelement auf den Wert der neuen Variable. Allgemein geschieht dies wieder in $O(n)$ $(suchen\ des\ Arrayelemt1\ +\ suchen\ des\ Arrayelemt2\ +\ Variable\ erstellen\ +\ Variable\ setzen\ +\ Arrayelement\ 1\ setzen + Arrayelement\ 2\ setzen = O(n) + O(n) + O(1) + O(1) + O(1) + O(1) = O(n))$.(Nach [2] S.138) Zuletzt sollte das Array wieder vollständig gelöscht werden, wodurch die Laufzeit von $O(n)$ anfällt (suchen + löschen). Summa summarum sind dies alle grundlegenden Befehle eines Arrays. Alle weiteren möglichen Funktionen, wie zum Beispiel ërstes Element eines Arrays ausgeben", lassen sich aus den genannten Operationen zusammensetzen. Die Abbildung 2.2 zeigt die Umsetzung einer alltäglichen Einkaufsliste in Pseudocodesprache.
Abschließend wird in (2.8) eine Implementierung der Operation ßuche Element in einem Arrayïn Pseudocode gezeigt. Der Index der ersten Zahl k wird zurückgegeben, falls eine in dem Array vorkommt. Sind mehrere vorhanden, wird der Index des ersten k's wiedergegeben. Falls keine Zahl k enthalten ist, wird eine -1 zurückgegeben. Im Worst Case werden in der Funktion alle Elemente durchlaufen und am Ende die Variable $ergebnis$ mit -1 ausgegeben.

Algorithmus *sucheElement(A,n,k)*

$\{Eingabe\ ist\ ein\ Array\ mit\ Integerwerten\ der\ Länge\ n\ und\ gesucht\ wird\ Wert\ k\}$

var $ergebnis\ :\ int;$

$ergebnis := -1;$

for each i **in** A :

  if $A[i]\ ==\ k$ **then** $ergebnis := i;$

  return $ergebnis;$

return $ergebnis;$

(2.8)

Der Nachteil eines klassischen Arrays ist die Beschränkung aller Elemente auf einen Typ und eine statische Länge. Der Vorteil ist der schnelle Zugriff auf die Elemente über einen Index.

2.2.3 Verkettete Liste

Die dritte Datenstruktur wurde bereits im vorherigen Absatz erwähnt: die einfach verkettete Liste. Diese ähnelt dem Array, ist aber eine einfache und minimale Variante davon. Arrays kann man sich vorstellen als verkette Listen, bei der jeder Index in $0(1)$ erreichbar ist.[8](S.39) Dies gilt nicht für die verketteten Listen - diese können nur nacheinander durchlaufen werden. Jedes Element besitzt einen Wert und einen Zeigerwert, der sich auf das nächste Element richtet. Damit kennt jedes Element nur seinen Nachfolger, anders als bei der doppelt verketteten Liste. Sie besitzt zudem einen Zeigerwert, der auf den Vorgänger des Elementes zeigt.[8](S.73-74) Der Durchlauf der Liste ist somit bei der einfach verketteten Liste immer vorgegeben: von vorne nach hinten. Für die Operation suchen muss die komplette Liste durchlaufen werden bis man bei dem gewünschten Element angekommen ist. Daher ist die Laufzeit $O(n)$. Da die Länge der Liste dynamisch angepasst wird, ist die minimale Laufzeit $O(1)$. Es wird also zunächst nur ein Element angelegt mit einem Nullzeiger. Das letzte Element der Liste zeigt also immer auf $null$. Um nun weitere Elemente hinzuzufügen, müssen nur die Zeiger der entsprechenden Elemente geändert werden. Will man beispielsweise eine 3 zwischen den Zahlen 2 und 4 einfügen, so muss der Zeiger der 3 auf das Element 4 gerichtet werden. Anschließend kann der Zeiger der 2 auf die 3 deuten und die Operation ist erledigt. Die Aktion ist somit in $O(1)$ abgeschlossen (siehe Abb. 2.3). Um aber an diese Elemente zu gelangen, startet man zuerst eine Suche nach der Zahl 2 mit Kosten von $O(n)$.
Das reine Lesen und Setzen eines Wertes läuft, ebenso wie das Löschen eines Wertes, in $O(1)$ ab. Davor erfolgt die Suche des Elementes bei allen drei Operationen in $O(n)$. Daher ist die maximale Laufzeit aller genannten Operationen von einfach verketteten Listen und Arrays $O(n)$. Dasselbe trifft für das vollständige Löschen der ganzen Liste zu, da hier über alle Elemente iteriert und jedes gelöscht werden muss. Der Vorteil von verketteten Listen besteht darin, dass jedes Element unterschiedlichen Typs sein kann und es keine begrenzte Länge für die Liste gibt.[2](S.107) Der Zugriff auf die Elemente in der Liste ist dagegen langsamer, da diese immer von Anfang bis Ende durchsucht werden müssen.
Hiermit sind alle fundamentalen Funktionen auf einer verketteten Liste, Array und Variable beschrieben. Um eine größere Funktionalität zu bieten, wurde die Anzahl der zur Verfügung stehenden Methoden des Programmes Älgorithm Visual Studioërweitert und kann durch die Architektur zukünftig ergänzt werden.

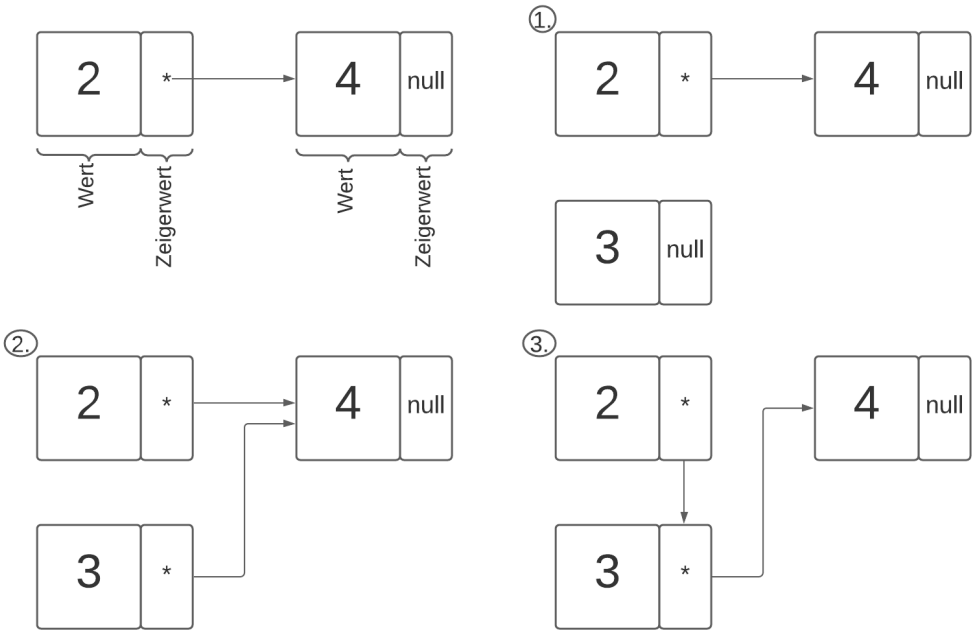

Abbildung 2.3: Einfügen eines neuen Elementes in eine verkettete Liste

2.3 Zusammenfassung

Die hier genannten Datenstrukturen, die in das Programm integriert wurden, sind Variablen, Arrays und verkettete Listen. Diese bilden das Fundament für kommende Arbeiten und sind zugleich häufig verwendete Datenstrukturen. Andere Formate wie Hash-Tabellen oder Graphen greifen immer wieder auf sie zurück. Bei Hash-Tabellen mit einer geschlossenen Adressierung sind für jeden Schlüssel mehrere Elemente zulässig.[8](S.120) Diese werden wiederum in einer Liste gespeichert, wofür man häufig ein Array verwendet. Ebenso können in Knotenpunkten von Bäumen mehrere Elemente gespeichert werden in Form eines Arrays. Zum Zwischenspeichern von Daten und Umstrukturieren der übergeordneten Datenstruktur werden außerdem meist Arrays verwendet, beispielsweise bei AVL-Bäumen.[8](S.154) Es ist möglich, ein Array in einen Baum umzuwandeln oder vice versa. Das unterstreicht die enge Bindung der Datenstrukturen.
Dieses Kapitel soll auf die Grundlagen hinweisen, auf die das Programm Älgorithm Visual Studioäufbaut. Die Konzepte und Inhalte einer Vorlesung aus dem Bereich der Datenstrukturen beinhaltet diese kurze Zusammenfassung beider Gebiete. Sie sind Element des Programmes und abstrakt integriert. Der lernende Benutzer soll ein vorlesungsnahes Programm bedienen, welches zu den gelehrten Konzepten eine Möglichkeit bietet, bestehende und eigene Algorithmen zu visualisieren.

3 Anforderungen

Dieses Kapitel dient der Erfassung der Anwendungsanforderungen. Die Definition dient nicht nur dem Kunden als Auftragsbeleg, sondern ebenso dem Programmierer und Entwickler als Leitfaden, in welcher Form etwas umgesetzt werden soll. Hierfür werden jedoch keine konkreten visuellen Bestandteile und Strukturierungen innerhalb des Programms genannt. Vielmehr wird eine generelle Bedeutung und Funktionalität in Textform verfasst. Weiterhin kann festgestellt werden, ob diese Anforderungen in dem genannten Maß erfüllt wurden.[12](S.51)
Zunächst werden Szenarien erläutert, die häufiger auftreten können. Dazu zählen in erster Linie Anwendungsfehler des Benutzers. Gleichzeitig wird dargestellt, wie das Programm darauf reagiert.

3.1 Szenarien

3.1.1 Szenario 1

ANNAHME: *Der Nutzer hat einen Algorithmus und eine Visualisierungsart ausgewählt.*

NORMAL: *Der Algorithmus wird ausgeführt und die Datenstrukturen visualisiert.*

MÖGLICHER FEHLER: *Der Algorithmus verwendet eine falsche Methode, um die Datenstruktur zu manipulieren.*

ANDERE AKTIVITÄTEN: -

SYSTEMSTATUS BEI ABSCHLUSS: *Es wird als Fehler angezeigt, dass die aufgerufenen Methoden nicht zu der Datenstruktur passen. Der Benutzer muss einen neuen Algorithmus angeben oder seinen alten anpassen. Die Anwendung ist noch am Laufen und wartet auf Interaktion mit dem Nutzer.*

3.1.2 Szenario 2

ANNAHME: *Der Nutzer hat einen Algorithmus und eine Visualisierungsart ausgewählt.*

NORMAL: *Der Algorithmus wird ausgeführt und die Datenstrukturen visualisiert.*

MÖGLICHER FEHLER: *Der Algorithmus ist nicht in einer Java-Datei gespeichert und kann nicht aufgerufen werden vom Programm. Das Programm terminiert unvorhergesehen.*

ANDERE AKTIVITÄTEN: -

SYSTEMSTATUS BEI ABSCHLUSS: *Es wird der Fehler angezeigt, dass es sich um keine Java-Datei handelt. Der Nutzer muss einen neuen Algorithmus angeben oder seinen alten anpassen. Die Anwendung ist noch am Laufen und wartet auf Interaktion mit dem Nutzer.*

3.1.3 Szenario 3

ANNAHME: *Der Nutzer hat einen Algorithmus und eine Visualisierungsart ausgewählt.*

NORMAL: *Der Algorithmus wird ausgeführt und die Datenstrukturen visualisiert.*

MÖGLICHER FEHLER: *Der Algorithmus des Nutzers beinhaltet mehrere Datenstrukturen gleichen Typs. Der Nutzer greift auf die falsche Datenstruktur zu und erhält ein unerwartetes Ergebnis.*

ANDERE AKTIVITÄTEN: -

SYSTEMSTATUS BEI ABSCHLUSS: *Der Algorithmus kann schrittweise durchlaufen werden, um den logischen Fehler einfacher zu entdecken. Nach Durchlauf des Algorithmus kann der Nutzer die Parameter des Programmes ändern und einen neuen Algorithmus auswählen.*

Die Anforderungen werden in mehrere Arten unterteilt: Nutzeranforderungen, nichtfunktionale und funktionale Anforderungen. Nutzeranforderungen richten sich besonders auf die Interaktion des Programmes mit dem Benutzer. Hier wird die Ein- und Ausgabe in schriftlicher Form festgehalten.[7](S.65) Bei den nicht-funktionalen Anforderungen werden Leistungs- und Effizienzanforderungen sowie technische und qualitative Bedingungen gestellt.[13](S.107-108) Externe und rechtliche Anforderungen wären ebenso Element davon. Wie der Name bereits sagt, richten sich die

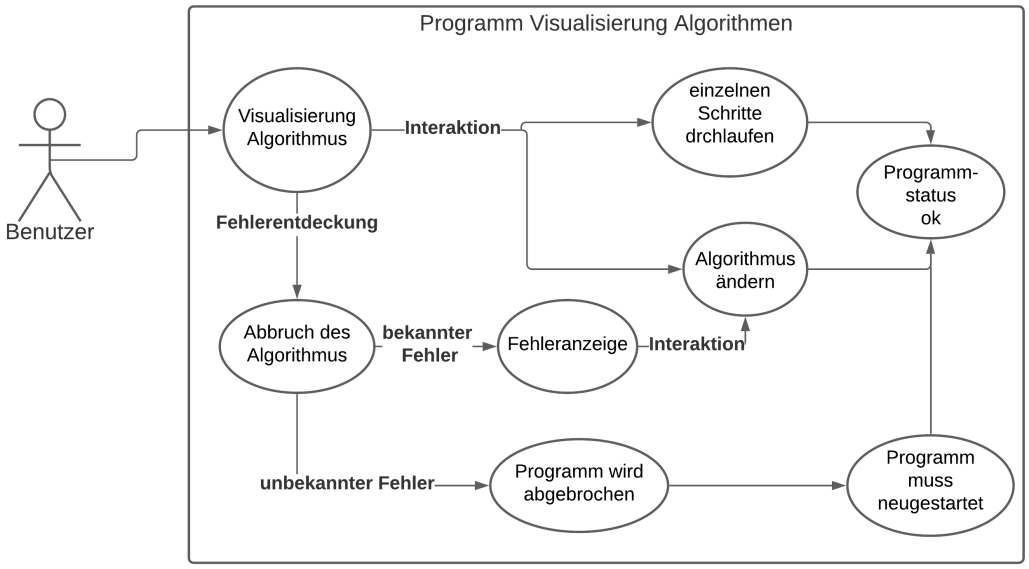

Abbildung 3.1: Use-Case des Umgangs mit einem Benutzerfehler im Algorithmus

funktionalen Anforderungen an die Funktionalität des Programmes. Welche Methoden und Ausführungen sollen erfolgen? Gleichermaßen gehören dazu Funktionen, die das Programm selbst im Hintergrund initiiert.[7](S.64) [13](S.107-108)
Alle Anforderungen wurden nach dem Eisenhower Prinzip in 4 Arten von Prioritäten klassifiziert.
A = Wichtig und dringend.
B = Wichtig und nicht dringend.
C = Nicht wichtig und dringend.
D = Nicht wichtig und nicht dringend.

3.2 Nutzeranforderungen

Der Nutzer...

U-REQ 1: ... soll seinen Algorithmus in den zugehörigen Datenstrukturen visualisieren. *Priorität: A*

U-REQ 2: ... soll seinen Algorithmus in das System integrieren können. *Priorität: A*

U-REQ 3: ... soll seinen Algorithmus auswählen können. *Priorität: A*

U-REQ 4: ... soll den Algorithmus Schritt für Schritt ausführen können. *Priorität: B*

U-REQ 5: ... soll die Schritte des Algorithmus rückgängig machen können. *Priorität: B*

U-REQ 6: ... soll die Befehlsliste, mögliche Hilfsmethoden und Instanzen des Algo-

rithmus einsehen können. *Priorität: D*
U-REQ 7: ... soll einen neuen Algorithmus nach Ablauf des alten wählen können. *Priorität: B*
U-REQ 8: ... soll die Bedienungshilfe des Programmes einsehen können *Priorität: A*
U-REQ 9: ... soll das Programm beenden können. *Priorität: A*
U-REQ 10: ... soll die Visualisierung von Algorithmen auf einem Array, von verketteten Listen und einzelnen Variablen darstellen können. *Priorität: A*
U-REQ 11: ... soll bei unterschiedlichen Fehlern eine verständliche Anzeige erhalten. *Priorität: A*

3.3 Nicht-funktionale Anforderungen

3.3.1 Produktanforderungen

NF-REQ 1: Das Programm soll anwenderfreundlich sein und eine deutliche Verbesserung der Programmierung von Algorithmen auf klassischen Datenstrukturen ermöglichen. *Priorität: A*
NF-REQ 2: Die Anwendung soll eine verständliche Visualisierung von Variablen, Arrays und verketteten Listen erzeugen. *Priorität: A*
NF-REQ 3: Der Anwender muss in der Lage sein, ohne externe Hilfe das Programm zu nutzen. *Priorität: A*
NF-REQ 4: Die Architektur der Software soll darauf ausgelegt sein, Erweiterungen für zusätzliche Datenstrukturen zu ermöglichen. *Priorität: A*

3.3.2 Nutzbarkeitsanforderungen

NF-REQ 5: Das Programm soll für alle gängigen Betriebssysteme nutzbar sein (Linux, Windows, iOS). *Priorität: A*
NF-REQ 6: Das System soll intuitiv bedienbar sein in Bezug auf Benutzerfreundlichkeit, Nutzbarkeit und Software-Ergonomie. *Priorität: B*
NF-REQ 7: Die Benutzeroberfläche soll ein minimalistisches Design haben. *Priorität: D*
NF-REQ 8: Der Algorithmus soll komplett oder schrittweise durchlaufen werden können. *Priorität: A*

3.3.3 Effizienzanforderungen

NF-REQ 9: Das Programm muss jede Anfrage des Benutzers innerhalb von 5 Sekunden ausführen. *Priorität: B*

3.3.4 Leistungsanforderungen

NF-REQ 10: Das Programm muss auf allen Betriebssystemen laufen können, die Java Version 10.0.1 oder höher installiert haben. *Priorität: A*
NF-REQ 11: Das Programm muss immer reagieren oder dem Benutzer die Möglichkeit geben, es zu schließen. *Priorität: B*

3.4 Funktionale Anforderungen

Das Programm...

F-REQ 1: ... soll die Algorithmen, die sich in einem Verzeichnis befinden, anzeigen und für den Benutzer auswählbar machen. *Priorität: A*
F-REQ 2: ... soll nach der Auswahl des Algorithmus durch den Benutzer diesen dynamisch in die Anwendung laden und dort lauffähig einbinden. *Priorität: A*
F-REQ 3: ... soll eine komplette oder Schritt für Schritt Visualisierung des Algorithmus ermöglichen. *Priorität: A*
F-REQ 4: ... soll bei der Ausführung einer vorgefertigten Funktion einen entsprechenden Befehl in das System für die Visualisierung einfügen. *Priorität: A*
F-REQ 5: ... soll bei dem Erstellungsbefehl einer Datenstruktur durch den Algorithmus diese und gleichzeitig die damit verbundenen Befehle zur Programmierung hinzufügen. *Priorität: A*
F-REQ 6: ... soll bei dem Erstellungsbefehl einer Datenstruktur durch einen Algorithmus die Visualisierung den derzeitigen Datenstrukturen hinzufügen und in das Layout einbinden. *Priorität: B*
F-REQ 7: ... soll nach Ablauf des Algorithmus diesen wieder ändern können. *Priorität: B*
F-REQ 8: ... soll die einzelnen Veränderungsschritte des Algorithmus vor- und zurücklaufen können. *Priorität: B*
F-REQ 9: ... soll bei Auftreten frequentierter Fehler diese abfangen und dem Benut-

zer verständlich visualisieren. *Priorität: B*
F-REQ 10: ... soll bei der Fehlerentdeckung Hilfestellung zur möglichen Lösung des Problems geben. *Priorität: B*
F-REQ 11: ... soll beim Durchlauf des Algorithmus alle Befehle darstellen, jedoch die Variablen und andere Datenstrukturen der Programmiersprache Java ignorieren. *Priorität: A*
F-REQ 12: ... soll beim kompletten Durchlauf des Algorithmus die Veränderungsschritte der Datenstrukturen verständlich visualisieren. *Priorität: A*
F-REQ 13: ... soll bei der schrittweisen Darstellung des Algorithmus die einzelnen Befehle im konkreten Algorithmus ausführen. *Priorität: A*
F-REQ 14: ... soll bei der stufenweisen Visualisierung des Algorithmus die Darstellung nach jedem Befehl pausieren und beim Betätigen des Buttons Nächster Schritt"den nächsten Befehl ausführen. *Priorität: A*
F-REQ 15: ... soll bei der sukzessiven Visualisierung des Algorithmus durch das Betätigen des Buttons "Vorheriger Schritt"den letzten Befehl rückgängig machen. *Priorität: A*
F-REQ 16: ... soll bei der Erstellung mehrerer Datenstrukturen in einem Algorithmus diese übersichtlich visualisieren. *Priorität: B*
F-REQ 17: ... soll durch einen "BeendenButton terminiert werden und alle Instanzen und zugehörigen Speicher wieder freigeben. *Priorität: B*
F-REQ 18: ... soll durch den SZurückButton die vorherige Fensteranzeige visualisieren. *Priorität: A*

3.5 Zusammenfassung

Die hier aufgelisteten Anforderungen entsprechen nicht mehr den der anfänglichen. Diese wurden im Laufe der Zeit immer wieder überarbeitet und an die derzeitigen Erkenntnisse und Überlegungen angepasst. Teile dieser Anforderungen sind durch den Programmierer hinzugefügt und andere durch Herrn Dr. Fischer sinngemäß übernommen worden.
Bereits im Aufbau des Programmes muss darauf geachtet werden, dass dem Anwender eine möglichst benutzerfreundliche und dennoch sinnvolle Interaktion zur Verfügung steht. Mit dieser Thematik befasst sich u.a. das nachfolgende Kapitel Architektur.

4 Architektur

Die Finalversion des Älgorithm Visual Studioënthält die Datenstrukturen Variable, Array und verkettete Liste. Erweiterungsmöglichkeiten durch z.B. Hash-Tabellen, Graphen und Bäume sind vorgesehen. Ein leichtes Einfügen dieser und zusätzlicher Features soll durch die Anlage der Struktur ermöglicht werden. Um diese Anforderung zu erfüllen, muss ein übergeordnetes Gerüst erstellt werden, welches diesem gerecht wird. Das Gerüst wird gestützt durch bekannte Entwurfsmuster, welche Problematiken auf eine bestimmte Weise bewältigen. Sie sind allgemein "bewährte Muster oder Lösungen für Probleme, die immer wieder auf eine bestimmte Art auftreten."[10](S.1) Sie haben viele positive Effekte, neben der Verständlichkeit und Erweiterbarkeit, auf die Software. Eine verbesserte Testbarkeit und Wartbarkeit ist zudem durch die Verwendung solcher Muster gegeben. Diese Aspekte sollen besonders für zukünftige Arbeiten berücksichtigt werden. Anschließend wird der Aufbau und die verwendeten Entwurfsmuster beschrieben. Im letzten Abschnitt des Kapitels wird die Funktionsweise der vorher beschrieben Komponenten behandelt und an Hand der wichtigsten Programmstellen erläutert.
Ein wichtiger Hinweis zu dem zugehörigen Code des Älgorithm Visual Studioünd den nachfolgenden Kapiteln: Die genannten Klassen, Methoden und Variablen haben deutsche Bezeichnungen, im Programmcode jedoch sind jene Namen ins Englische übersetzt worden.

4.1 Allgemeine Struktur

Entwurfsmuster legen in allen Abstraktionsebenen Strukturen fest. Dabei sind Architekturmuster übergeordnete Einheiten, welche die Organisation im Makro-Bereich der Software festlegen. Sie definieren die bedeutenden Systemobjekte und deren Beziehung untereinander. Die Interaktion und das Verhalten von Bausteinen wird gleichermaßen dadurch beschrieben. Zudem werden Schnittstellen zu anderen Systemen oder anderen Teilen innerhalb des Systems genannt.[13](S.11-12) Dagegen sind Entwurfsmuster sowohl für Makro- als auch Mikrobereiche definiert, vermehrt jedoch für Strukturen in kleineren Dimensionen. Design-Pattern (engl. für Entwurfsmuster) beschreiben das genauere Verhalten und Vorgehen von abstrakten Objekten in einem System. Strukturelle Muster und Erzeugungsmuster zählen aber gleichermaßen zu den Design-Pattern.[10](S.3) Architekturmuster sind für eine fundamentale

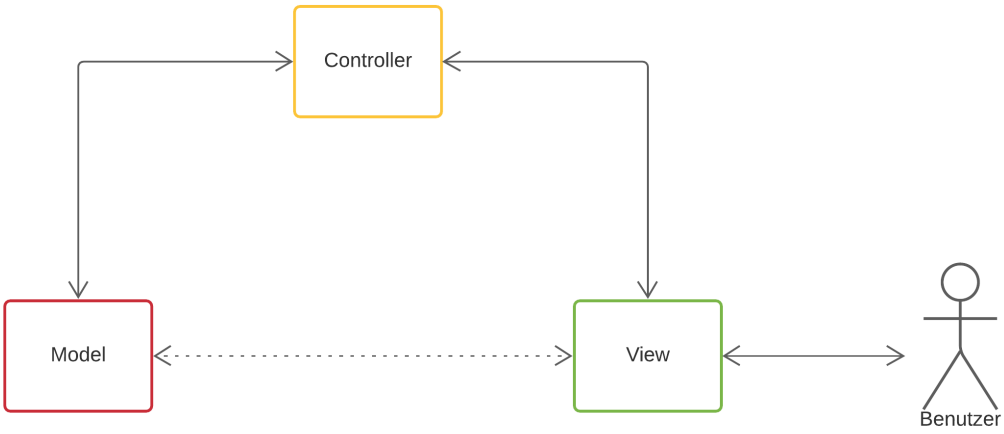

Abbildung 4.1: Schaubild des Model-View-Controller

Struktur im Aufbau der Software zuständig und Entwurfsmuster beschreiben das untergeordenete Verhalten der Systembausteine.
Als allgemeines Architekturmuster wurde hier das MVC-Muster verwendet. Es besteht aus den Komponenten Model, View und Controller, welche zusammen die berühmte Abkürzung ergeben. Das Objekt View befasst sich mit der Darstellung der Daten, der Controller mit der Organisation der Interaktionsdaten und das Model beinhaltet die Logik und die restlichen Informationen des Systems.[7](S.127) Damit wird ein System auf oberster Ebene in 3 Objekte eingeteilt, die klare und feste Aufgaben vergeben. Der Controller sorgt für das Weiterleiten von Informationen zwischen der View, welche die Daten darstellt, und dem Model, welches sie verwaltet. Das Model wiederum verarbeitet neue Eingaben, speichert sie ab und schickt neue Daten an den Controller weiter. Dieser wandelt die Informationen in eine entsprechende Form um, damit das View-Objekt diese Werte dem Benutzer anzeigen kann. Die View hat dabei keinerlei Logik inne, sondern stellt nur die gegebenen Daten dar. Der Controller steuert diese Informationen und übergibt sie der View in der richtigen Form. Er ist somit eine Art Bindeglied zwischen Darstellung und Logik des Systems. Hinter dem Model befindet sich demnach ein komplettes System mit vielen weiteren Objekten, Mustern, Datenbanken und Abstraktionsschichten.

4.2 Aufbau und UML-Diagramme

Im Nachfolgenden werden die weiteren verwendeten Entwurfsmuster genannt und erläutert, die im Älgorithm Visual StudioÄnwendung gefunden haben. Als erstes wird der Algorithmus des Benutzers betrachtet, der in das System eingebunden werden muss. Der Anwender wählt einen Algorithmus aus, der sich in einem vorgesehenen Ordner befindet. Das bedeutet, dass zur Laufzeit die Algorithmus-Dateien angezeigt

werden und die ausgewählte Datei in das System dynamisch integriert wird. Um dies zu gewährleisten, wurde das Strategie-Muster verwendet. Es gehört zu den Verhaltensmustern und sorgt für das Einbinden einer Komponente zur Laufzeit in einen vorgeschriebenen Kontext.[10](S.101) Dabei wird eine abstrakte Oberklasse erstellt, welche eine oder mehrere abstrakte Methoden deklariert. Diese Oberklasse, auch Strategie genannt, dient als gemeinsame Schnittstelle und der einfachen Integration in einen Kontext.[3](S.385) Die von der Strategie erbenden Klassen implementieren die in der Oberklasse deklarierten, abstrakten Methoden. So hat jede Klasse eine eigene Variante dieser Funktion bzw. Funktionen und stellt somit eine konkrete Strategie dar. Da alle erbenden Strategien die gleiche Superklasse haben, ist ein einfaches Hinzufügen der konkreten Strategie durch folgende Java Syntax möglich: $Oberklasse\ oberklasse\ =\ new\ Unterklasse();$. Dadurch kann im Kontext die Funktion dynamisch ausgeführt werden unabhängig ihrer Implementierung.[10](S.105) In diesem Programm ist die Oberklasse Strategie der $AbstrakterAlgorithmus$. Der Nutzer muss also eine Klasse anlegen, die von der Klasse $AbstrakterAlgorithmus$ erbt und einen konkreten Algorithmus darstellt. Der $konkreterAlgorithmus$ beinhaltet die abstrakte Methode $algorithmusAusführen$. Diese Funktion soll von dem Benutzer mit bereitgestellten Funktionen gefüllt werden und implementiert den benutzerdefinierten Algorithmus. Zur Laufzeit muss der Anwender einen dieser konkreten Algorithmen auswählen, damit das Programm eine Instanz dieser Klasse erstellt und die definierte Funktion $algorithmusAusführen$ aufrufen kann. So kann immer ein Ein- und Ausklinken eines benutzerdefinierten Algorithmus gewährleistet werden. Das Strategie-Muster wird in der Oberklasse im Original als Interface implementiert. Dadurch ist eine Implementierung der erbenden Klassen gleichermaßen gefordert.[10](S.101) Die hier beschriebene Version des Entwurfsmusters ist somit eine Variation des ursprünglichen Musters.

Nach der Einbindung des konkreten Algorithmus soll nun eine Visualisierung des Algorithmus stattfinden. Dabei gibt es zwei Arten der Darstellung: ein kompletter Durchlauf des Algorithmus und eine sukzessive Darstellung der Datenstrukturmanipulationen. Letztere stellt eine besondere Herausforderung dar, da jede Veränderung einer Datenstruktur per Knopfdruck durch den Benutzer ausgeführt wird. Jede Anpassung muss zugleich umkehrbar sein, d.h. es kann beliebig vor und zurück gesprungen werden zwischen allen benutzerdefinierten Funktionen. Damit eine solche Anpassung möglich ist, wird das Command-Pattern (Befehlsmuster) verwendet. Dieses Muster kapselt eine Funktion in eine eigene Klasse ab. Ein Wiederholen und Umkehren des Befehls ist dadurch möglich. Zudem können mehrere einzelne Befehle in einen größeren Makro-Befehl zusammengefügt werden.[3](S.292) Im ursprünglichen Ansatz des Musters ist ein Interface für das abstrakte Objekt $Befehl$ vorgesehen. Ein konkreter Befehl implementiert das übergeordnete Interface mit der Funktion $ausführen$. Diese Funktion soll eine gewünschte Veränderung auf einem Empfänger ausführen. Dazu muss bei der Initialisierung des konkreten Befehls die zugehörige Methode und der Empfänger angelegt werden. Der Empfänger ist die Da-

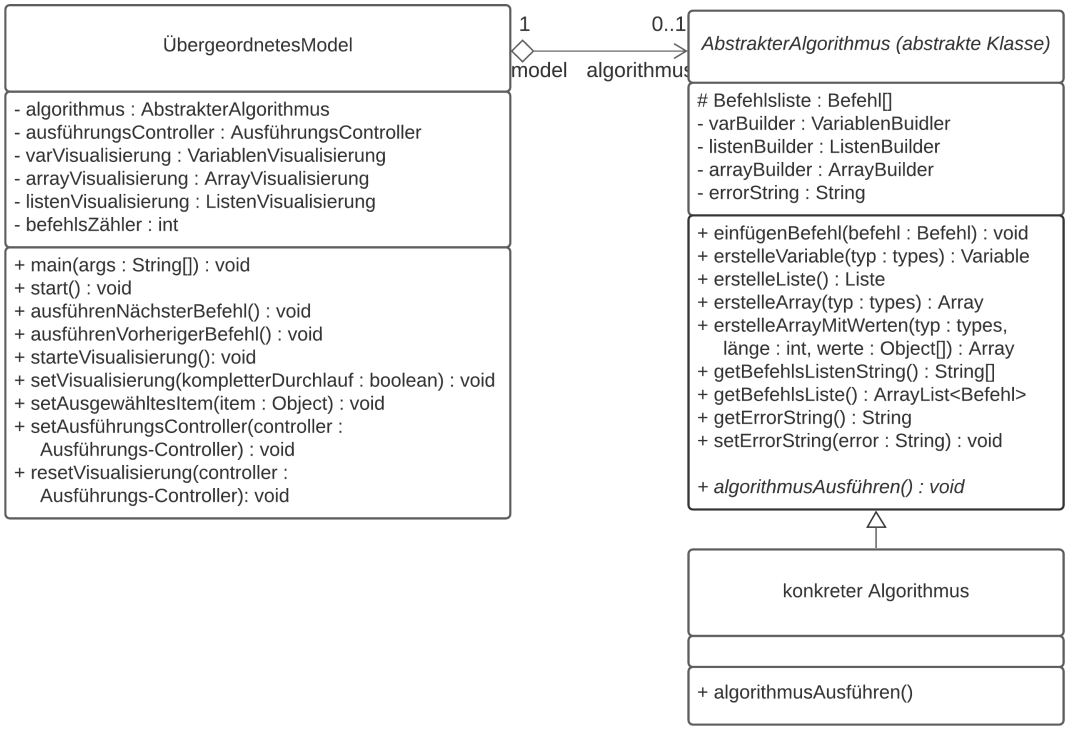

Abbildung 4.2: Strategie-Entwurfsmuster: Der $AbstrakterAlgorithmus$ entspricht der Strategie und das $ÜbergeordnetesModel$ dem Kontext.

tenstruktur, auf der operiert wird. Der Erbauer der Befehle war anfangs der abstrakte Algorithmus. Durch verschiedene Gründe (siehe Kapitel Implementierung) wurde die Erstellung der Befehle den abstrakten Datenstrukturen zugewiesen, welche den Zugriff auf die eigentliche Datenstruktur regeln. Sie werden lediglich von dem Anwender benutzt, verbergen die Erstellung der Befehle und delegieren den Zugriff auf die zugehörigen, tatsächlichen Datenstrukturen. Wie in der folgenden Abbildung zu sehen ist, wurde eine weitere Abstraktionsstufe zwischen dem Befehl und einem konkreten Befehl hinzugefügt. Diese Zwischenschicht hilft dem $ÜbergeordnetesModel$ zu erkennen, auf welcher Datenstruktur der Befehl operiert. Das Model verkörpert wiederum den Aufrufer der Befehle in dem Visualisierungsprozess.

Zuletzt müssen zur Laufzeit die jeweiligen Datenstrukturen erstellt werden. Zu einem guten Programmierstil gehört eine Kapselung des Erstellungscodes in eine separate Klasse. Daher befinden sich die Erstellungsfunktionen in eigenen Objekten, welche das klassische Builder-Pattern (Erbauermuster) nachbilden. Das Muster eignet sich besonders für komplexe Instanzen, die eine mehrstufige Konfiguration des zu erstellenden Objektes benötigen. Dies gilt zwar nicht für die anfänglichen Datenstrukturen, die hier verwendet wurden, jedoch sind Graphen, Hash-Tabellen und Bäume aufwendiger zu erstellen. Für eine Erweiterung dieser Arbeit soll die Option bestehen, diese Komponenten leicht in das vorhandene Programm einzubinden. So steht jeder Erbauer zunächst für sich und erstellt die Schablonen-Datenstruktur und die Info-Datenstruktur. Die Info-Variable beispielsweise ist die eigentliche Variable

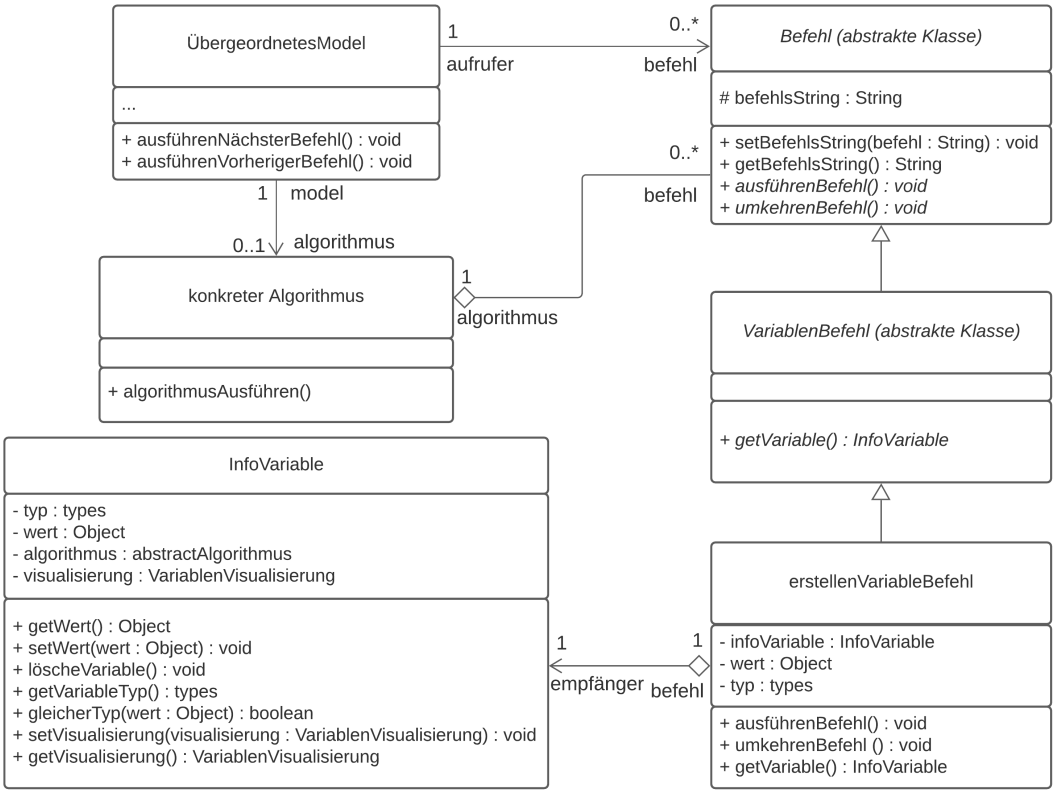

Abbildung 4.3: Befehls-Entwurfsmuster: Empfänger stellt die Info-Variable dar, den Aufrufer das $\ddot{U}bergeordnetesModel$.

mit dem entsprechenden Wert und Typ. Diese beiden werden von dem Erbauer erstellt und fördern eine übersichtlichere Strukturierung.

4.3 Ablauf und Sequenzdiagramme

Mit der vorherigen Passage ist die allgemeine Struktur des Älgorithm Visual Studiofestgelegt. Offen bleibt, wie die einzelnen Komponenten untereinander und in welcher Reihenfolge agieren. Sequenzdiagramme unterstützen das Verständnis der Beziehung und Funktionsweise der einzelnen Klassen. Dadurch wird ersichtlich, zu welchem Zeitpunkt die Funktionen von wem aufgerufen werden. Aus diesen Informationen lässt sich der Ablauf des Programmes zusammensetzen. Die zentralen Stellen in dem Programmablauf werden nachfolgend beschrieben.
Das $\ddot{U}bergeordnetesModel$ stellt den Beginn des Programmes dar. Es erschafft das Fenster und übergibt an die $Auswahl\text{-}Algorithmus\text{-}View$. Diese stellt mit dem zugehörigen Controller alle konkreten Algorithmus Klassen aus dem Ordner dar und lässt den Benutzer eine davon auswählen. Durch die Bestätigung wird dem übergeordneten Model die Auswahl übergeben, damit er eine Instanz der Algorithmusklasse erstellt. Anschließend wird die Methode $algorithmusAusführen$ aufgerufen und die darin befindlichen Funktionen durchlaufen. Zeitgleich wird bei einem Feh-

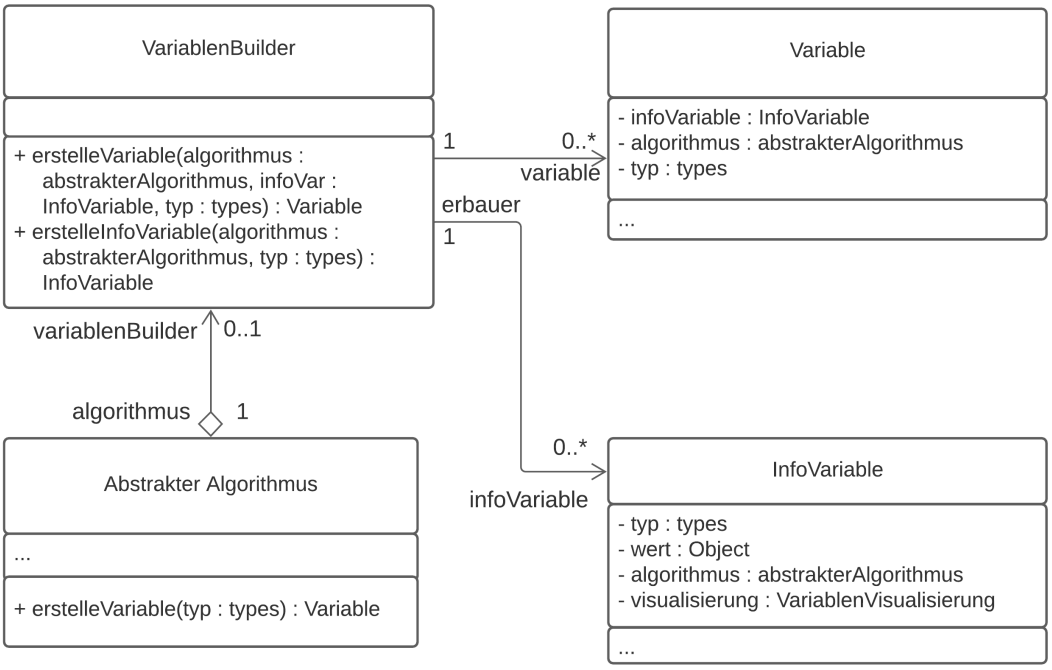

Abbildung 4.4: Erstellung Variable: Variablen-Erbauer erstellt $Variable$ und $Info\text{-}Variable$.

ler der $Error\text{-}String$ des konkreten Algorithmus gesetzt. In jedem Fall wird dem $AlgorithmusAuswählenController$ dieser String überreicht. Ist er leer, wird der Benutzer zur $Auswahl - Visualisierung - View$ weitergeleitet. Im anderen Fall ist der String mit einer Fehlermeldung in dem konkreten Algorithmus gesetzt, worauf dieser innerhalb der $Error\text{-}View$ dargestellt wird. Dort hat der Anwender die Möglichkeit, einen anderen Algorithmus auszuwählen, die Hilfe-Seite aufzurufen oder das Programm zu beenden. Eine Terminierung der Anwendung ist jederzeit möglich. Veranschaulicht wird dieser beschriebene Ablauf durch das Sequenzdiagramm in Abbildung 4.5.

Während der Ausführung des benutzerdefinierten Algorithmus müssen mehre-

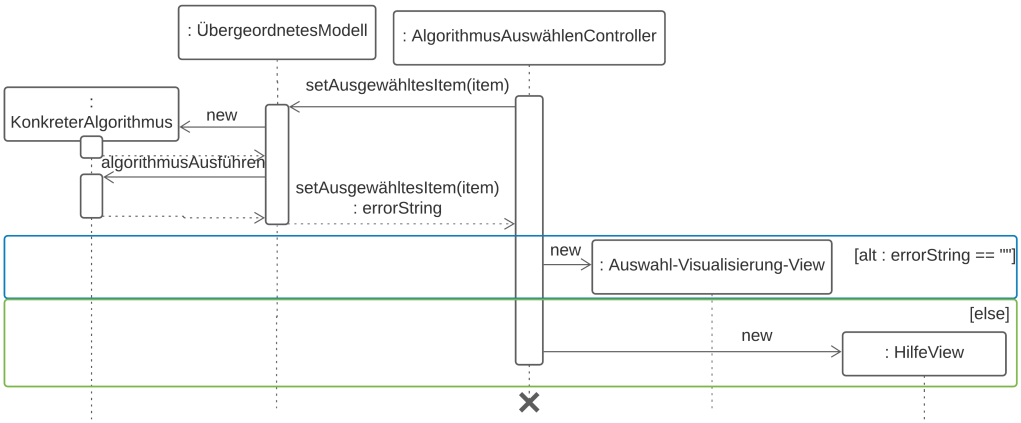

Abbildung 4.5: Sequenzdiagramm von der Auswahl eines Algorithmus

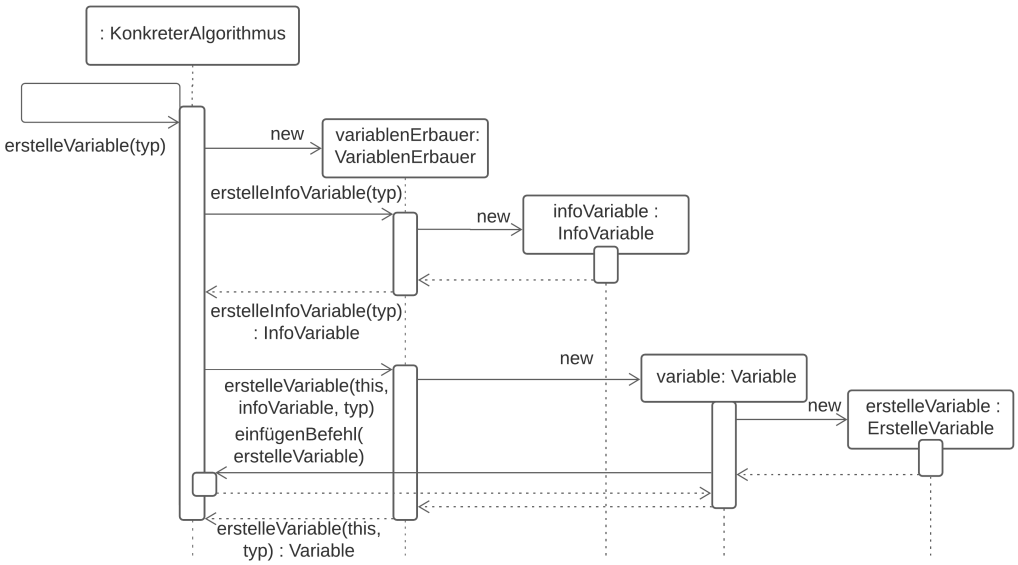

Abbildung 4.6: Sequenzdiagramm von der Erstellung einer Variable

re Komponenten erstellt werden. Dazu zählen die Erbauer der jeweiligen Datenstrukturen, die Datenstrukturen selbst und die Befehle. Wird zunächst eine Funktion innerhalb des Algorithmus aufgerufen wie z.B. $erstelleVariable$, so muss, falls die Klasse noch keinen Variablen-Erbauer besitzt, er diesen erstellen. Danach wird in dem Funktionsaufruf $erstelleVariable$ innerhalb des Erbauers erst die $InfoVariable$ und anschließend die Schablonenklasse $Variable$ initialisiert. Grund dafür ist, dass bei dem Aufruf des Konstruktors der Variablenklasse die $InfoVariable$ übergeben werden muss. Dies bindet eine Info-Datenstruktur an eine Schablonen-Datenstruktur. Bei der Initialisierung der $Variable$ wird zudem eine Instanz des Befehls $ErstelleVariable$ gesetzt und durch den Funktionsaufruf $einfügenBefehl(erstelleVariable)$ der Befehlsliste im konkreten Algorithmus hinzugefügt. Somit sind Befehl, Schablonenklasse und Info-Datenstruktur initialisiert und die $Variable$ wird dem $konkreter\ Algorithmus$ übergeben.

Das Setzen eines neuen Wertes einer Variable wird über die Funktion $setWert$ der Schablonen-Variable initialisiert. Dort wird der entsprechende Befehl $setVariable$ von der Klasse $Variable$ erstellt und wieder der Befehlsliste in dem konkreten Algorithmus angehängt. Anschließend wird die eigentliche Aktion auf der $InfoVariable$ ausgeführt. Ist der neue Wert der gleiche Datentyp wie die Variable selbst, kann die Funktion normal ausgeführt und die nächste Funktion im Algorithmus bearbeitet werden. Im anderen Fall, wird der $Error\text{-}String$ im $konkreterAlgorithmus$ gesetzt und nach Ablauf des Algorithmus wird dieser Fehler dem Anwender durch eine zusätzliche Seite angezeigt.

Für den Fall, dass der Algorithmus ohne Fehler durchlaufen wurde, kann eine Visualisierungsmethode gewählt werden. Es steht eine komplette oder manuelle schrittweise Darstellung zur Wahl. Unabhängig davon muss das $ÜbergeordnetesModel$

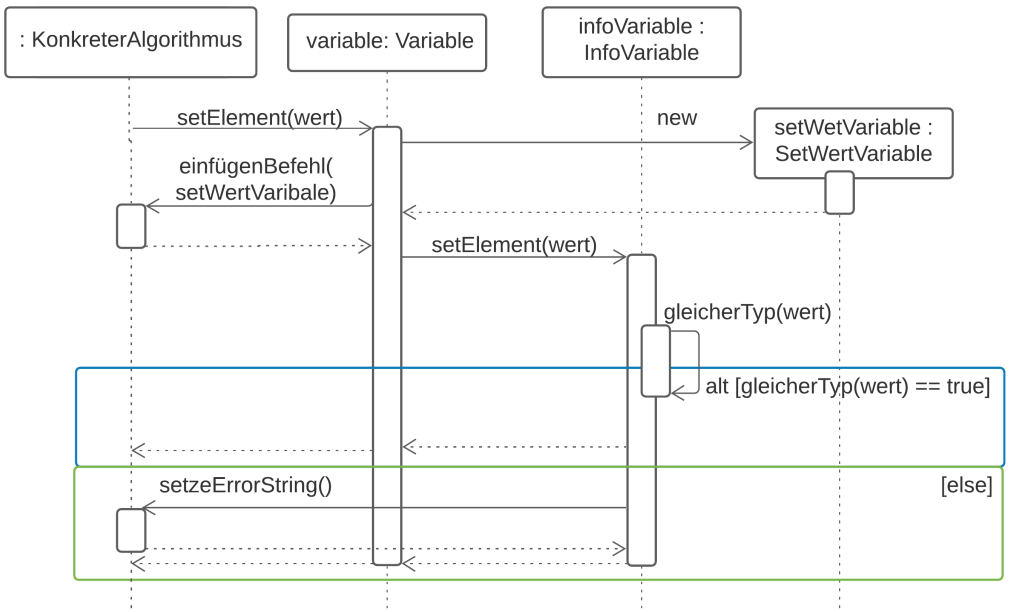

Abbildung 4.7: Sequenzdiagramm von dem Setzen einer Variable

durch die Befehlsliste des konkreten Algorithmus iterieren und die Visualisierung organisieren. Hat der Nutzer die stufenweise Methode gewählt, kann er durch das Betätigen des Buttons $NächsterSchritt$ sich den nächsten Befehl anzeigen lassen. Das Drücken des Buttons ruft im Hintergrund die Funktion $ausführenNächsterBefehl$ in der Klasse $ÜbergeordnetesModel$ auf. Das Model wiederum erhält die Befehlsliste des Algorithmus durch den Aufruf $getBefehlsListe$. Nun wird der nächste Befehl ausgewählt. In dem Sequenzdiagramm 4.8 ist dies der $erstelleVariable$ Befehl. Von diesem lässt sich das Model die $InfoVariable$ übergeben, auf der der Befehl ausgeführt werden soll. Zunächst wird getestet, ob die $VariablenVisualisierung$ der $InfoVariable$ gesetzt ist. Sollte zudem keine $VariablenVisualisierung$-Klasse existieren, muss gleichzeitig durch das Model eine erstellt werden. Danach kann die $VariablenVisualisierung$ in der $InfoVariable$ gesetzt werden. Erst jetzt kommt es zur eigentlichen Befehlsausführung. Durch den Aufruf $ausführen$ innerhalb des Variablen-Befehls wird der initiale Wert der $InfoVariable$ wiederhergestellt. Die $InfoVariable$ existiert daher noch und ihr Wert wird nur überschrieben. Die Aktion wird in dem Befehl mit Hilfe des Methodenaufrufs $setVariable(initialerWert)$ auf der $InfoVariable$ umgesetzt. Innerhalb der $InfoVariable$ wird nun die eigentliche Operation ausgeführt und ihr Wert neu beschrieben. Darüber hinaus wird auf der gesetzten $VariablenVisualisierung$ die zugehörige Darstellung aufgerufen. Durch $setVariable(infoVariable, typ)$ wird in der Klasse $VariablenVisualisierung$ die Änderung vorgenommen. Nach jeder Anpassung der Darstellung wird die Funktion $generiereNode()$ auf der $VariablenVisualisierung$ gestartet. Wie der Name bereits sagt, wird ein Node des JavaFX-Frameworks erstellt, welcher alle momentanen Variablen und deren aktuellen Wert in dem abstrakten Node vereinigt. Dieses

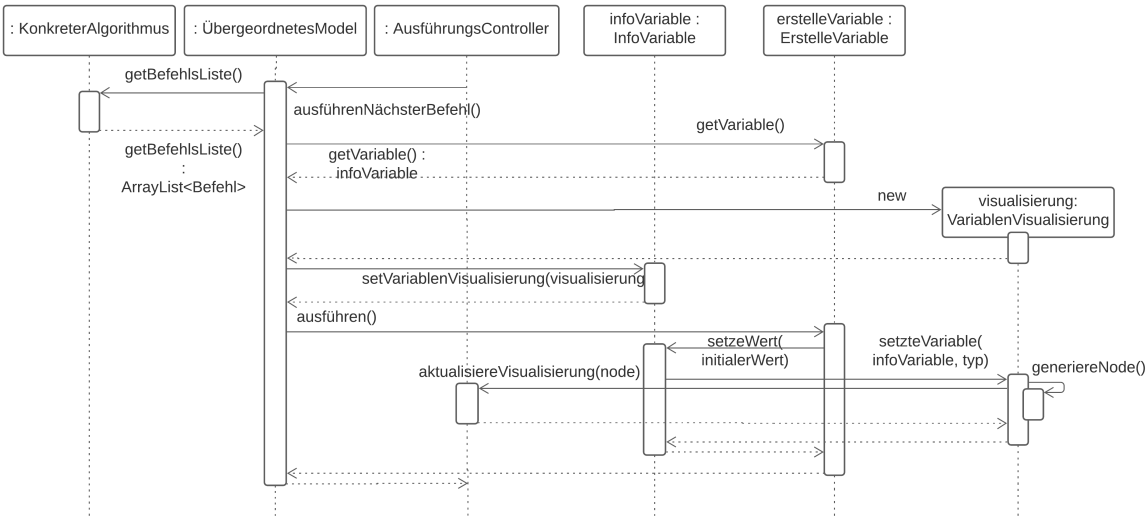

Abbildung 4.8: Sequenzdiagramm von der Visualisierung des $erstelleVariable$-Befehls

Node-Objekt wird der Funktion $aktualisiereVisualisierung(node)$ als Parameter überreicht und die Änderung von dem $AusführungsController$ in der View übernommen. Danach haben $AusführungsController$, $VariablenVisualisierung$ und $InfoVariable$ ihre Arbeit erledigt und der Befehl ist vollständig ausgeführt. Gleiches gilt für das $ÜbergeordnetesModel$, welches als letzte Instanz die Kontrolle wieder an den $AusführungsController$ übergibt und auf weitere Interaktion durch den Anwender wartet.

4.4 Zusammenfassung

Die Architektur stellt das Grundgerüst einer Anwendung dar und dient gleichzeitig als Veranschaulichung des zu erstellenden Programmcodes. Dank einer guten Planung und Organisation lässt sich in der Implementierung viel Zeit und Arbeit sparen. Zudem muss für kommende Arbeiten ein strukturierter und erweiterbarer Aufbau hinterlassen werden. Diese Anforderungen werden durch bekannte Entwurfsmuster gewährleistet und mit Hilfe von UML-Diagrammen veranschaulicht. Die Schaubilder wurden gemäß den Anforderungen erstellt. Angesichts der permanenten Anpassungen der Anwendung wurden die Neuerungen immer auf die Anforderungen und demnach ebenso auf die UML-Diagramme übernommen. Die Gesamtübersicht der Anhangsdatei ÜML-Diagrammïst die Finalversion des Älgorithm Visual Studio". Im Gegensatz dazu setzten sich die Sequenzdiagramme zeitlich im Laufe der Implementierung zusammen.
Unterstützt durch die beiden Darstellungen konnten die Funktionsweise und Struktur des Programmes zusammengefasst werden. Sie beinhalten den Aufbau und die

letztendliche Verhaltensweise des Programmes in Form von Verlaufsgraphiken, die durch das nachfolgende Kapitel Implementierung ergänzt werden.

5 Implementierung

Der Implementierungsprozess ist in diesem Kapitel das zentrale Thema. Begleitet wird dieser mit den Herausforderungen und Problemen bei der Erstellung. Selbst mit einer perfekten Planung und Strukturierung der Software werden sich immer kleinere oder größere Hindernisse aufzeigen, die überwunden werden müssen. So kann sich das Design und der Aufbau der Software nachträglich jederzeit ändern. Diese Dynamik und die Gestaltung und Umsetzung der Software sollen anschließend dargestellt werden. Nach der Strukturierung des Programmes durch Entwurfsmuster wurde die Wahl getroffen, welche Programmiersprache und welches Oberflächenframework für das Projekt verwendet wurde. Die Auswahl der Programmiersprache beschränkte sich zunächst auf bekannte objektorientierte Sprachen wie Java, C++ und Python. Letzteres wurde wegen der schlechten Umsetzung der Klassenkonzepte aus der Auswahl entfernt. Durch die Möglichkeit der Nutzung von abstrakten Klassen und die statisch typisierte Kompilierung fiel die Wahl letztendlich auf Java. In Kombination mit dem Oberflächenframework JavaFX ist weiterhin eine moderne und einfache Umsetzung des MVC-Entwurfsmusters möglich. Die Views werden in eigene fxml-Dateien gespeichert und eine Controller-Klasse an die View-Datei gekoppelt.

5.1 Ablauf und Organisation

Für den Prozess des Programmierens wurde ein Arbeitsplan konzipiert, der in diesem Abschnitt beschrieben wird. Da manche Programmteile vor anderen erstellt werden müssen, ist eine Abfolge der Komponenten wichtig. Bevor beispielsweise die Befehle aus dem Entwurfsmuster gebaut werden, sollte eine Umgebung bestehen, in der sie eingebettet werden können. Dazu zählen Empfänger, also die Datenstrukturen, und der Aufrufer, welcher anfangs der abstrakte Algorithmus war.
Die ersten zu implementierenden Komponenten sind die Datenstrukturen. Für jede Datenstruktur ist zunächst eine Klasse erstellt worden (siehe Abschnitt "Herausforderungen und Umsetzung"). Diese Klassen beinhalten ihre jeweiligen Datentypen und Werte und sind mit passenden Funktionen für ihre Datenstruktur erweitert. So beinhaltet jede Klasse alle zugehörigen Operationen und Werte. Danach wird der abstrakte Algorithmus erstellt. Er gibt die allgemeine Form des benutzerdefinierten Algorithmus vor. In seiner abstrakten Funktion $algorithmusAusführen$ wird der

letztendliche Algorithmus eingebettet. Zunächst ist er für die Erstellung der Befehle zuständig. Die Datenstruktur kann zwar die Befehle ebenso instanziieren, jedoch ist diese Variante nicht im eigentlichen Sinne des Command-Patterns. Nach der Fertigstellung des Algorithmus können die jeweiligen Erbauerklassen hinzugefügt werden, welche die Datenstrukturen dynamisch in das System einbinden. In diesem Anwendungsfall sind die Erbauer klein und müssen nicht, wie in der Grundidee des Builder-Patterns, komplexer konfiguriert werden. Diese Abkapselung ist vor allem für zukünftige Datenstrukturen konzipiert, die einen aufwendigeren Erstellungsprozess benötigen. Die Einteilung hilft gleichermaßen der besseren Strukturierung und Wartung für kommende Neuerungen. Anschließend werden die einzelnen Befehlsklassen hinzugefügt. Bei der Initialisierung werden sie mit allen nötigen Parametern gesetzt, welche die Manipulation der Datenstruktur betreffen. So können aus den Befehlsklassen alle Informationen gewonnen werden: die betroffene Datenstruktur mit allen Parametern. Infolgedessen ist der Großteil der Logik des Programmes implementiert. Nun werden die einzelnen Komponenten (Befehle, Erbauer, abstrakter Algorithmus und Datenstrukturen) zusammengeführt und auf ihr korrektes Zusammenspiel geprüft.

Das Zusammenführen der Komponenten findet innerhalb des übergeordneten Models statt. Es verbindet außerdem die Oberfläche des Programmes und die Visualisierungsklassen. Das Model setzt sich demnach Stück für Stück zusammen. Der nächste Schritt befasst sich mit den Views und allen JavaFX Elementen in fxml-Dateien. Diese sind zunächst ohne besonderen visuellen Style und auf ihre reine Funktion ausgerichtet. Die Controllerklassen zu den Views werden separat hinzugefügt. Hier werden zudem die Daten der Logikschicht den Viewkomponenten übergeben und graphisch angepasst. In die Controllerelemente werden ebenso die Übergänge zu anderen Views integriert. Das Model initialisiert die erste View und das Fenster der Anwendung. Mit dem Erstellen neuer Controller-, View- und Visualisierungklassen werden dem Model zugleich weitere Funktionen hinzugefügt, welche Daten zwischen den Komponenten verteilen. Innerhalb dieser Funktionen werden gleichermaßen die restlichen Logikkomponenten organisiert. Der letzte Schritt der Implementierung bildet die Visualisierung der Datenstrukturen. Diese werden als Javaklassen konzipiert, welche für die einzelnen Veränderungen auf der jeweiligen Datenstruktur eine Darstellung erzeugen. So hat die Visualisierung einer Variablen die Funktion $erstelleVariable$ zum Erstellen einer neuen Variable, $setVariable$ zum Setzen eines neuen Wertes und $löscheVariable$ zum Löschen des Wertes und der Variablen. Die Funktion $getVaribale$ übergibt den aktuellen Wert einer Variablen, bedarf aber keiner Visualisierung und ist daher auch nicht in der Visualisierungsklasse enthalten. Analog dazu werden nur visualisierungsnotwendige Befehle in der Darstellungsklasse umgesetzt. Wenn nun das übergeordnete Model die Befehlskette durchläuft und ausführt, wird die $VariablenVisualisierung$ der zugehörigen Info-Variablen des Befehls gesetzt. Die $InfoVariable$ führt dann die Visualisierung auf der $VariablenVisualisierung$-Klasse aus. Dabei werden alle Datenstrukturen, die

zu dem Ausführungszeitpunkt des konkreten Algorithmus existieren, in der Visualisierungsklasse der Datenstruktur gespeichert. Nach der Ausführung einer Operation auf der Visualisierungsklasse wird die Darstellung aktualisiert und die Veränderung angezeigt. Somit sind alle Elemente aus dem UML-Diagramm im vorherigen Kapitel erstellt und die Implementierung ist abgeschlossen.

5.2 Herausforderungen und Umsetzung

Dieser Abschnitt befasst sich mit den größten Herausforderungen der Implementierung. Welche Teile der Codierung waren besonders anspruchsvoll und wie wurden die Hindernisse überwunden? Diese Fragen werden nachfolgend beantwortet. Daran schließt sich eine Zusammenfassung, welche Anforderungen erfüllt bzw. nicht erfüllt wurden.
Es ist nicht ungewöhnlich, dass sich bei der letztendlichen Umsetzung einer Software Probleme und neue Herausforderungen aufzeigen. Eine kleine Anpassung der Softwarestruktur kann weitreichende Folgen haben. So war anfangs eine einfache Klasse für jeden Datenstrukturtyp geplant. Die Datenstruktur hat dabei die Befehlserstellung übernommen. Da die Initialisierung aber im ursprünglichen Sinne des Patterns eigentlich für den abstrakten Algorithmus vorbehalten war, wurde die Erstellung ihm zugetragen. Dies sollte ebenso Endlosschleifen verhindern, da manche Befehle und Funktionen einer Datenstruktur andere Funktionen der eigenen Klasse aufgerufen haben. Währenddessen hat eine Funktion einen Befehl erstellt und eine andere Funktion aufgerufen. Diese hat einen neuen Befehl instanziiert, der wiederum eine andere Funktion ausgeführt hat. Dadurch wurde eine Aneinanderreihung von Befehlen erzeugt, die eigentlich nur durch einen einzigen repräsentiert werden sollte. Nachdem aber nun die Befehlserstellung in den abstrakten Algorithmus versetzt wurde, musste für jede ausführbare Funktion einer Datenstruktur eine Methode des abstrakten Algorithmus hinzugefügt werden. Jede dieser Funktionen erstellte einen zugehörigen Befehl und leitete die Operation auf die entsprechende Datenstruktur weiter. Infolgedessen war jede Methode jederzeit innerhalb des abstrakten Algorithmus ausführbar, ohne dass eine Initialisierung der passenden Datenstruktur stattgefunden hat. Beispielsweise konnte der Benutzer immer über die abstrakte Algorithmus-Klasse die Funktion $getVariable()$ aufrufen, ohne jemals eine Variable initialisiert zu haben. Diese Variante sorgt jedoch nicht für eine benutzerfreundliche Bedienung. Daher musste eine neue Art der Datenstruktur und Befehlserstellung erarbeitet werden, die diesen Anforderungen gerecht wird und trotzdem eine einfache Bedienung ermöglicht. Um die Nutzung eigener Funktionen der Datenstruktur bei externen Funktionsaufrufen zu bewerkstelligen, muss die Erstellung der Befehle von der tatsächlichen Datenstruktur getrennt werden. Um aber nicht alle Funktionen der Datenstruktur in den abstrakten Algorithmus zu implementieren, wurde eine separa-

te Klasse erstellt, die als weitere Abstraktion der tatsächlichen Datenstruktur für den Nutzer dient. Diese Klasse hat den Namen der jeweiligen Datenstruktur und erstellt die Befehle. Sie leitet die Funktionen wie $setVariable$ oder $löscheVariable$ an die eigentliche Datenstruktur, genannt Info-Datenstruktur, weiter. Die Info-Datenstruktur führt dann die letztendliche Operation auf sich aus und kann dabei andere Hilfsmethoden ihrer eigenen Klasse verwenden, ohne neue Befehle ungewollt zu erstellen. Gleichzeitig sind im abstrakten Algorithmus nur noch die Erstellungsfunktion(en) pro Datenstruktur integriert, die eine Entität der geforderten Datenstruktur zurück gibt. Diese Instanz muss dann genutzt werden, um neue Operationen auf sich selbst auszuführen, indem man die Funktionen der Datenstruktur wie folgt aufruft:
$Variable\ var\ =\ this.createVariable(types.NUMBER);$
$var.setVariable(10);$
Nachteil dieser Variante ist die Beanspruchung der Ressourcen durch eine weitere Klasse, die pro Instanz einer Datenstruktur zusätzlich anfällt und instanziiert werden muss.
Durch die Veränderungen im abstrakten Algorithmus und den Datenstrukturen mussten gleichermaßen auch die Befehle immer wieder angepasst werden. Zunächst wurden nicht nur die Daten in den Befehlen gespeichert, sondern ferner der Aufruf zur Manipulation der Datenstruktur gesetzt. Dies wurde aber zum reinen Abspeichern der Parameter und der zugehörigen Info-Datenstruktur während der Initialisierung geändert. Das Ausführen der Aktion erfolgt erst im zweiten Schritt. Primär soll der konkrete Algorithmus des Nutzers "kompiliert"werden. Das bedeutet, die Funktionen werden durchlaufen und es wird geprüft, ob alle Parameter richtig gesetzt wurden und keine Fehler in dem Algorithmus enthalten sind. Dabei werden dennoch alle Funktionen auf den jeweiligen Datenstrukturen ausgeführt, jedoch noch nicht visualisiert. Daneben werden ebenso die Befehle instanziiert und gespeichert. Ist der Algorithmus durchlaufen, so wird an Hand der Befehlsliste die Visualisierung vollzogen. Zugleich werden die Befehle durch die Funktion $ausführen$ aufgerufen, wodurch die entsprechende Operation auf ihrer gesetzten Info-Datenstruktur beginnt. Weiterhin gibt es eine Funktion, die die Aktion umkehrt, damit in der Visualisierung diese Veränderung wieder rückgängig gemacht werden kann. Der Aufruf dieser Methode ist lediglich für die sukzessive Darstellung des Algorithmus vorbehalten, da hier die Befehlsliste zurückgelaufen werden kann. Im späteren Implementierungsverlauf wurde eine weitere Abstraktionsebene für die Befehle hinzugefügt. Jede Datenstruktur hat dadurch eine weitere abstrakte Klasse zwischen der Superklasse $Befehl$ und dem konkreten Befehl erhalten: $VariablenBefehl$ für Variablen, $ArrayBefehl$ für Arrays und $ListenBefehl$ für verkettete Listen. Dies war nötig, damit das übergeordnete Model innerhalb der Befehlsliste zwischen den jeweiligen Datenstrukturen unterscheiden kann. Die entsprechende Visualisierungsklasse konnte infolgedessen dem Befehl und demnach der Datenstruktur zugeordnet werden. So wird beim Durchlauf der Befehlsliste die Visualisierungsklasse in den Datenstrukturen gesetzt und die Operation auf der Datenstruktur ausgeführt. Zeit-

gleich wird neben der Methode die passende Darstellung aufgerufen.
Zu den Herausforderungen zählte besonders die Visualisierung der Datenstrukturen. Es benötigt viel Zeit, sich in neue unbekannte Frameworks einzuarbeiten und ein tiefgründigeres Verständnis dafür zu erhalten. Während des Aufbaus der Software muss jedoch direkt die Darstellung mit eingebunden werden. Es blieb trotz Recherche offen, ob die Art der Darstellung mit der geplanten Struktur umgesetzt werden kann. Manchmal ist eine Durchführung eines Plans in eine Sprache oder Framework zwar möglich, aber dafür sehr umständlich. Glücklicherweise war die Verwirklichung der Visualisierungsklassen und deren Einbindung in die fxml-Dateien erfolgreich. Hier wurde ein Weg gefunden, der alle momentanen Datenstrukturen in der Visualisierungsklasse speichert und an den entsprechenden Controller schickt, damit dieser die Daten anzeigt. Hinzu kam die Herausforderung, eine richtige und ansprechende Darstellung zu konzipieren. Aufgrund der zeitlichen Befristung lag der Fokus auf Funktionalität und Darstellung der Datenstrukturen.
Eine Anforderung war das Abfangen von Programmierfehlern in dem konkreten Algorithmus eines Nutzers. Dies wurde ermöglicht, indem die Eingaben der benutzten Funktionen nach Verstößen untersucht wurden. Ist einer entdeckt, setzt die entsprechende Methode den $Error\text{-}String$ des Algorithmus auf die passende Fehlermeldung. Ein direktes Abfangen konnte zeitlich nicht umgesetzt werden und kann in zukünftigen oder nachträglichen Arbeiten an dem Programm verbessert werden. Innerhalb einer IDE sind die Meldungen in der Konsole sichtbar.
Die Visualisierungsklassen stellten die letzten Komponenten bei dem Implementierungsprozess dar. Eine vollständige Visualisierung aller Funktionen eines Array und einer Variablen wurde erfolgreich erstellt. Einzig die verketteten Listen konnten nicht fertiggestellt werden. Die Methoden des Erstellens und Löschens der Liste sind als einzige implementiert.
Der komplette Visualisierungsdurchlauf konnte zudem nicht funktionsfähig umgesetzt werden. Das Problem war, dass kein Weg gefunden wurde, nach jedem Befehl die Visualisierung zu pausieren und das Endergebnis des Algorithmus nicht direkt anzuzeigen. Dafür wurde die schrittweise Visualisierung des Algorithmus wirkungsvoll umgesetzt und kann innerhalb der beiden Datenstrukturen, Array und Variable, genutzt werden.

5.3 Zusammenfassung

Die Differenz zwischen der Planung und Implementierung wurde in diesem Kapitel transparent beschrieben. So wurde auch auf die Anforderungen eingegangen, welche durch den zeitlichen Faktor nicht umgesetzt werden konnten. Eine Ausbesserung kann durch eine nachfolgende Bearbeitung vollständig fertiggestellt werden. Gleichermaßen wurde auf die Strukturierung des Prozesses zur Softwareerstellung sowie

deren Probleme eingegangen. Anforderungen, die nicht erfüllt werden konnten, und einige weitere Erweiterungsmöglichkeiten werden im letzten Kapitel aufgegriffen. Sie geben einen Ausblick auf kommende Projektarbeiten an diesem Programm.

6 Zusammenfassung

6.1 Fazit

In diesem Abschnitt wird zu den anfänglichen Zielsetzungen Stellung genommen und ein Fazit für das Programm Älgorithm Visual Studio"gegeben. Dazu ist aufzuzählen, welche Anforderungen erreicht bzw. nicht erreicht wurden.
Die allgemeine Struktur sowie der Aufbau der Software wurden erstellt und mit der beschriebenen Funktionalität umgesetzt. Anhand des UML-Diagrammes (siehe Datei ÜML-Diagrammïm Anhang) sind alle Klassen und deren Anweisungen integriert. Der Algorithmus kann dynamisch eingebunden werden. Die Datenstrukturen und alle Funktionen sind implementiert und lauffähig. Eine passende Visualisierung aller Methoden ist für die Variable und das Array möglich. Bei den verketteten Listen konnten nicht alle Darstellungsfunktionen der $ListenVisualisierung$ implementiert werden. Die restliche Funktionalität der verketteten Liste ist jedoch vorhanden. Zudem wurden alle Befehlsklassen implementiert. Die Oberfläche ist minimalistisch, kann aber mit Styles weiter verschönert werden.
Ein generelles Abfangen der Fehler im Algorithmus ist integriert, allerdings sind nicht alle möglichen Fehlerarten abgedeckt. Der Fehler eines Algorithmus wird dem Benutzer auf einer Seite angezeigt. Dieser ist dabei immer der erste Fehler im Algorithmus. Die Beschreibung dieses Fehlers ist kurz und prägnant, kann aber durch weitere textliche Hilfestellungen vervollständigt werden.
Für den Fall, dass ein Benutzer Hilfsmethoden oder Variablen und Instanzen von Java Datenstrukturen verwendet, werden diese nicht in der Ausführung des Algorithmus angezeigt. Eine Anzeige dieser Zusatzinformationen war durch den Programmierer angestrebt, konnte aber aus zeitlichen Gründen nicht erstellt werden.
Der komplette Durchlauf der Visualisierung ist zum Abgabetermin fehlerhaft, da ein Pausieren der Darstellung nicht erreicht wurde. So wird direkt das Endergebnis des ausgewählten Algorithmus visualisiert, ohne die Anzeige nach jedem Veränderungsschritt kurz zu stoppen. Dagegen ist die schrittweise Visualisierung aller getätigten Befehle in beide Richtungen möglich und hebt die einzelnen Anpassungen der Strukturen hervor.
Das Erreichen der restlichen Ziele und Anforderungen ist in der Implementierung gelungen. Wenige Änderungen und Anpassungen müssen lediglich vorgenommen werden, damit das Programm für alle derzeitigen Datenstrukturen einsatzbereit ist. Das Älgorithm Visual Studio"kann dennoch in Betrieb genommen werden, wenn die

Benutzung sich zunächst auf die schrittweise Darstellung von Variablen und Arrays beschränkt. Die Fertigstellung wird für die Zukunft als Ziel gesetzt.

6.2 Ausblick

Nach diesem Absatz ist zwar die Beschreibung des Älgorithm Visual Studio"beendet, nicht aber dessen Entwicklung und Erweiterung. Wie so oft kann ein Programm durch zusätzliche Features immer wieder neu erfunden werden. Neben einer Sprachauswahl und einem besserem Oberflächendesign sind auch andere bedeutendere Methoden zukünftig möglich.
Wie zuvor erwähnt ist die Architektur des Programmes darauf ausgelegt, weitere Funktionalität aufzunehmen. Neue Datenstrukturen, die visualisiert werden sollen, sind dabei nur ein Beispiel. Es können demnach alle möglichen Strukturen in dem gleichen Muster wie Variable, Array und verkettete Liste der Anwendung hinzugefügt werden. Pro neuer Datenstruktur muss eine Schablonen- und Info-Klasse, inklusive ihrem Erbauer, eine Visualisierungsklasse und die Befehlsklassen erstellt werden.
Eine ausgereiftere Visualisierung kann nicht nur optisch ansprechender aussehen, sondern zudem Bewusstsein für die Kosten einer Operation auf der entsprechenden Datenstruktur schaffen. Zum Beispiel ist das Löschen eines Elementes in einem Array teuer, falls das entfernte Element nicht an letzter Stelle steht. Dies hat zur Folge, dass alle Elemente nach dem gelöschten eine Position nach vorne verschoben werden müssen. Eine realistischere Darstellung, welche diesen Aspekt hervorhebt, kann daher einen besseren Lernfortschritt erzielen.
Die finale Version des Älgorithm Visual Studioïst in Java programmiert und verlangt demnach die Implementierung des konkreten Algorithmus in der selben Sprache. Setzt man sich als Ziel, die Eingabe des konkreten Algorithmus sprachenunabhängig zu gestalten, ist eine Anpassung der Struktur nötig. Die Veränderung jedoch richtet sich vor allem auf den abstrakten Algorithmus. Dieser muss die Funktionalität besitzen, Pseudocode-Anweisungen in die entsprechenden Java-Instruktionen umzuwandeln. Der abstrakte Algorithmus könnte dadurch als eine Art Parser fungieren und die Funktionen auf die anderen Datenstrukturen weiterleiten.
Zuletzt kann die Performance durch weitere Architektur- und Code-Anpassungen verbessert werden. Dieser zwar sehr allgemeine Aspekt ist womöglich bei jeder Software zutreffend, kann aber neue Möglichkeiten bieten.
Die genannten Beispiele stellen bereits große Erweiterungen für das Programm Älgorithm Visual Studio"dar. Die Bedeutung der Anwendung wird besonders dadurch hervorgehoben, dass eine automatische Visualisierung der Datenstrukturen durch das Programm erstellt wird. Ein manueller Aufruf ist somit nicht nötig und wird durch die Funktionsaufrufe auf den Datenstrukturen verborgen. Zudem kann das Erkennen logischer Fehler durch das visuelle Ablaufen der Befehlsliste in beide Richtungen

erleichtert werden. Außerdem fördert es das Verständnis des eigenen Algorithmus. Diese beiden Aspekte und die vielen Erweiterungsmöglichkeiten machen dieses Programm zu einem hervorragendem und effektiven Werkzeug für Programmierer, die auf klassischen Datenstrukturen Algorithmen entwerfen wollen. Das Älgorithm Visual Studio"bildet das Fundament für eine Weiterentwicklung dieses Projektes. Seine Fortführung und Komplettierung bietet eine große Chance, ein grundlegendes und wichtiges Instrument für Programmieranfänger zu werden.

Literatur

[2]Thomas Letschert Andreas Gogol-Döring. *Algorithmen und Datenstrukturen für Dummies*. Wiley-VCH Verlag GmbH Co. KGaA, 2020 (siehe S. 13–15).

[3]Ralph Johnson John Vlissides Erich Gamma Richard Helm. *Design Paterns - Entwurfsmuster als Elemente wiederverwendbarer objektorientierter Software*. mitp Verlag GmbH Co. KG, 2015 (siehe S. 25).

[6]Manfred Sommer Heinz-Peter Gumm. *Programmierung, Algorithmen und Datenstrukturen*. Walter de Gruyter GmbH, 2016 (siehe S. 7).

[7]Anja Metzner. *Software Engineering-Kompakt*. Hanser, 2020 (siehe S. 18, 19, 24).

[8]Stefan Dieker Ralf Hartmut Güting. *Datenstrukturen und Algorithmen*. Springer Verlag, 2018 (siehe S. 7–13, 15, 16).

[10]Florian Siebler. *Design Patterns mit Java*. Hanser, 2014 (siehe S. 23, 25).

[11]Ronald L. Rivest Cliford Stein Thomas H. Cormen Charles E. Leiserson. *Introduction to Algorithms 3. Edition*. The MIT Press, 2009 (siehe S. 9).

[12]Stefan Toth. *Vorgehensmuster für Software-Architektur*. Carl Hanser Verlag, 2014 (siehe S. 17).

[13]A. Chughtai E. Ihler T. Kehrer U. Mehlig U. Zdun o. Vogel I. Arnold. *Software-Architektur*. Sketrum Akademischer Verlag, 2009 (siehe S. 18, 19, 23).

Webseiten

[1]64json. *Algorithm Visualizer*. 2016. URL: `https://algorithm-visualizer.org` (besucht am 2. Nov. 2021) (siehe S. 4).

[4]David Galles. *Data Structure Visualizations*. 2011. URL: `https://www.cs.usfca.edu/~galles/visualization/Algorithms.html` (besucht am 2. Nov. 2021) (siehe S. 4).

[5]Dr Steven Halim. *VisuAlgo*. 2011. URL: `https://visualgo.net/en` (besucht am 2. Nov. 2021) (siehe S. 4).

[9]Reminisque. *Algomation*. 2020. URL: `https://github.com/Reminisque/algomation` (besucht am 2. Nov. 2021) (siehe S. 4).

Abbildungsverzeichnis

Colophon

This thesis was typeset with LaTeX 2$_\varepsilon$. It uses the *Clean Thesis* style developed by Ricardo Langner. The design of the *Clean Thesis* style is inspired by user guide documents from Apple Inc.

Download the *Clean Thesis* style at `http://cleanthesis.der-ric.de/`.

Erklärung

Hiermit versichere ich, dass ich die vorliegende Arbeit selbstständig und ohne Benutzung anderer als der angegebenen Hilfsmittel angefertigt habe. Die aus anderen Quellen oder indirekt übernommenen Daten und Konzepte sind unter Angabe der Quelle gekennzeichnet. Die Arbeit wurde bisher weder im In- noch Ausland in gleicher oder ähnlicher Form in einem Verfahren zur Erlangung eines akademischen Grades vorgelegt.

Mainz, 20.12.2021

Manuel Selenka