ADV-Tree-Module

Studienarbeit

Abteilung Informatik

Hochschule für Technik Rapperswil

Herbstsemester 2018

Autoren: Jan Winter, Fabian Meier

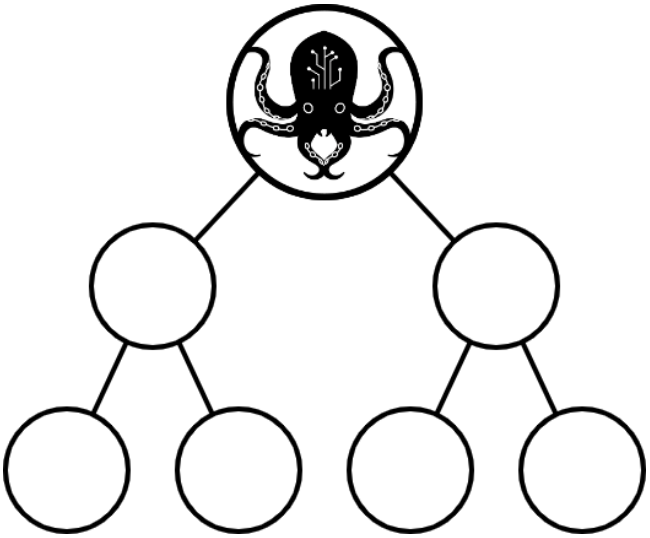
Betreuer: Thomas Letsch

HSR
HOCHSCHULE FÜR TECHNIK
RAPPERSWIL
FHO Fachhochschule Ostschweiz

HSR Hochschule für Technik Rapperswil
Abteilung Informatik

Studienarbeit

ADV-Tree-Module



Autoren Jan Winter
Fabian Meier

Betreuer Thomas Letsch

Zeitraum: 17.09.2018 - 21.12.2018

ADV-Tree-Module

Studenten

- Fabian Meier
- Jan Winter

Einführung

An der Hochschule für Technik Rapperswil (HSR) werden in den Modulen "Algorithmen und Datenstrukturen 1 und 2" theoretische Informatikkonzepte wie z.B. Sortier-, und Suchalgorithmen sowie die Implementation grundlegender Datenstrukturen unterrichtet.
Um das Erlernen dieser Konzepte zu vereinfachen, wurde im Rahmen einer Bachelorarbeit der "Algorithm & Data Structure Visualizer" (ADV) entwickelt. Dieser unterstützt die Studierenden beim Erlernen mittels Visualisierung der behandelten Algorithmen und Datenstrukturen.
Der ADV ist als erweiterbares Framework ausgelegt, bei welchem mit weiteren sog. Modulen bestimmte Algorithmen und/oder Datenstrukturen hinzugefügt werden können, z.B. Stack, Queue, Graphen etc..

Aufgabenstellung

In dieser Studienarbeit soll ein weiteres, generelles Modul für die Darstellung von Bäumen erstellt werden.

Im Weiteren soll für den Fall, dass der Baum in einem Array gespeichert wird, auch der Inhalt des Arrays dargestellt werden können.

Unter Berücksichtigung von aktuellen Software-Engineering-Methoden soll ein geeigneter Entwicklungsprozess definiert und darauf basierend das Projekt realisiert werden.

Technologien

- JavaFX
- Enterprise Architect

Generelles

- Die Vorgaben der Abteilung Informatik [1] sind einzuhalten, insbesondere die Anleitung zur Dokumentation [2].
- Die "Generelle Richtlinien für Studien- und Bachelorarbeiten" [3] sind einzuhalten.
- Mit dem CASE-Tool Enterprise Architect ist ein UML-Modell zu führen, welches synchron mit den Programm-Sourcen und der Projekt-Dokumentation ist.
- Ein Java-Entwickler muss mit der Projekt-Dokumentation in die Lage versetzt werden, die Applikation in Betrieb zu nehmen und weiter entwickeln zu können.

Termine

- Montag, 17.09.18 Beginn der Studienarbeit
- Freitag, 21.12.18 12:00 Uhr Abgabe der Studienarbeit

Betreuung

- Betreuer
  Thomas Letsch
  tletsch@hsr.ch
  055 - 22 24 567 (HSR Büro 5.204); 055 - 214 43 50 (Geschäft)
- Besprechungen
  Wöchentliche Besprechung jeweils Mittwoch 08:30 Uhr

Referenzen

[1] Skripte-Server: /Informatik/Fachbereich/Studienarbeit_Informatik/SAI
[2] Anleitung Dokumentation BA_SA_170905.pdf
[3] "Generelle Richtlinien für Studien- und Bachelorarbeiten"
(v1.8 / 18.02.2018, Thomas Letsch)

Rapperswil, 17.September 2018

Thomas Letsch

Abstract

An der Hochschule für Technik Rapperswil werden die Module Algorithmen und Datenstrukturen 1 und 2 unterrichtet. Zurzeit wird für die Visualisierung von komplexen Algorithmen der Graph-Visualization-Service (GVS) verwendet, um den Studenten das Erlernen zu vereinfachen. In einer vorangegangenen Bachelorarbeit wurde der Algorithm & Datstructure Visualizer (ADV) entwickelt, der den GVS ablösen soll. Darin wurden bereits einige Module entwickelt, beispielsweise zur Darstellung von Graphen oder Arrays. Bisher fehlt dem ADV jedoch die Funktionalität zur Darstellung von Bäumen. Das Ziel dieser Studienarbeit ist die Entwicklung eines Moduls, welches den ADV um die Baumdarstellung erweitert und ihn damit bereit für den Einsatz im Unterricht macht.

Bei der Spezifikation der Anforderungen an das ADV-Tree-Module wurde der Funktionsumfang zur Baumdarstellung des GVS berücksichtigt. Zusätzlich erweitert es den Funktionsumfang um die Darstellung eines Arrays bei binären Bäumen. Bei der Definition der Architektur orientierte sich das Modul stark an den Anforderungen zur Modulentwicklung des ADV, fokussierte sich aber auch auf die Erweiterbarkeit. Um die Umsetzbarkeit der Architektur zu überprüfen, wurde früh ein Prototyp entwickelt. Das Modul wurde in zwei Komponenten aufgeteilt. Die Komponente, die der Student bei sich in der Codebase verwendet, enthält Klassen zur Darstellung von Bäumen. Die zweite Komponente ist für die Visualisierung im Fenster des ADV zuständig. Für die Darstellung komplexer Algorithmen wurde das Modul so erweitert, dass diesem Kind-Module hinzugefügt werden können.

Als Resultat der Studienarbeit entstand ein Modul, das aus einem Array einen Binärbaum, aus einem Wurzel-Knoten einen binären oder generellen Baum und aus einer Collection von Knoten einen Wald darstellen kann. Bei den binären Bäumen ist es zudem möglich, die zugehörige Array-Darstellung zu visualisieren. Um Rotationen besser darzustellen, ermöglicht das Modul die Fixierung der Knoten im Fenster.

Management Summary

Ausgangslage

In den Modulen Algorithmen und Datenstrukturen 1 und 2 werden Algorithmen verschiedenster Komplexität unterrichtet. Um dem Studenten das Lernen zu erleichtern, wird im Unterricht ein Programm zur Visualisierung von Datenstrukturen eingesetzt. Die Applikation unterstützt die grundsätzliche Darstellung von Bäumen und Graphen, lässt allerdings einige Elemente aus dem Unterricht weg.
In einer vorangegangenen Bachelorarbeit wurde das Nachfolgeprogramm ADV entwickelt. Für den produktiven Einsatz im Unterricht fehlt diesem jedoch die Möglichkeit zur Darstellung von Bäumen. Das Ziel dieser Studienarbeit ist die Erweiterung des ADV um die Darstellung von Bäumen, damit das gegenwärtig im Unterricht eingesetzte Programm abgelöst werden kann.

Vorgehen, Technologien

In der ersten Phase wurden sämtliche primären sowie einige optionale Anforderungen an das ADV-Tree-Module gesammelt. Dazu wurde der Code sowie die Dokumentation des bereits vorhandenen GVS analysiert und Anforderungen vom Betreuer entgegengenommen.
Im nächsten Schritt wurde das bereits entwickelte ADV-Framework studiert. Dabei ging es darum, das neue ADV-Tree-Module in das vorhandene Framework einzugliedern, ohne die bestehenden Richtlinien zu verletzen. Zur Überprüfung der Realisierbarkeit der geplanten Architektur wurde ein Prototyp entwickelt, welcher die wichtigsten Komponenten des Moduls beinhaltete.
Während der Entwicklungs-Phase lag der Schwerpunkt zunächst auf den primären Anforderungen. Diese wurden Schrittweise implementiert und in der wöchentlichen Besprechung mit dem Betreuer betrachtet. Nachdem die primären Anforderungen fertig implementiert waren, konnte der Fokus auf die optionalen Anforderungen gerichtet werden. Diese wurden zuerst vom Betreuer priorisiert und danach in der Reihenfolge der Wichtigkeit implementiert.
Zum Abschluss der Entwicklungs-Phase wurden Usability-Tests durchgeführt. Dabei ging es darum, Schwierigkeiten bei der Bedienung des Moduls zu entdecken sowie Hinweise für Verbesserungsvorschläge zu erhalten. Die Anregungen wurden dokumentiert, mit dem Betreuer besprochen und in den meisten Fällen in das Modul integriert.

Ergebnisse

Als Ergebnis dieser Arbeit entstand das ADV-Tree-Module, welches die Funktionalität zur Darstellung von Bäumen der bisher eingesetzten Applikation vollständig abdeckt. Ausserdem erweitert es diese um die fehlenden Elemente aus dem Unterricht und bietet dem Studenten bessere Hilfestellungen.
Durch die Usability Tests konnte das Programm durch nützliche Hinweise verbessert werden, sodass es die Probleme der Studenten aus Algorithmen und Datenstrukturen 1 und 2 adressiert und ihnen dadurch das Erlernen erleichtert.

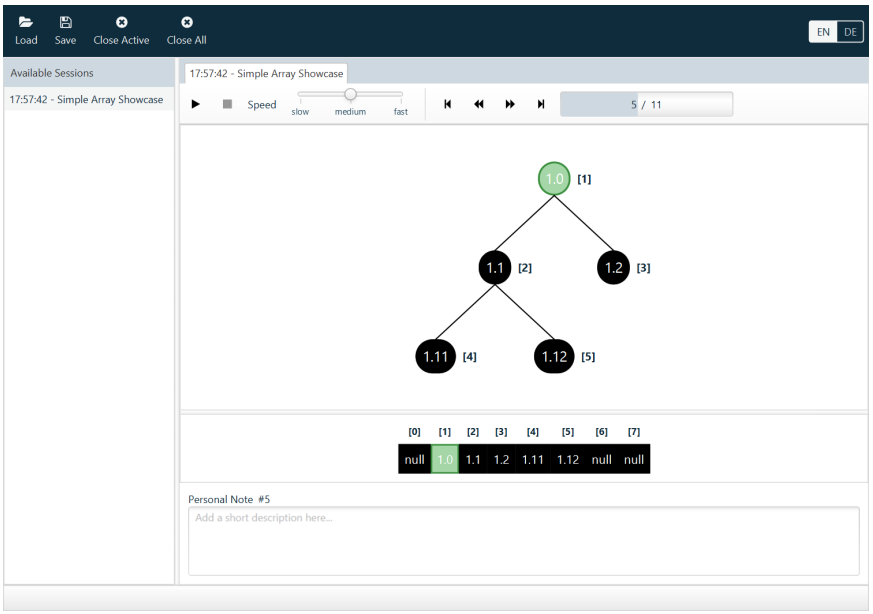


Abbildung 0.1.: Darstellung eines Binärbaums im ADV-Tree-Module

Ausblick

Mit dem ADV-Tree-Module bietet der ADV sowohl die gesamte Funktionalität der derzeit eingesetzten Applikation, als auch einige zusätzliche Elemente. Damit kann der ADV die bisherige Applikation nun voll umfänglich ersetzen.

Danksagungen

Wir danken folgenden Personen für ihre Unterstützung während unserer Studienarbeit:

– Unserem Betreuer Thomas Letsch für die unterstützende und angenehme Zusammenarbeit während der Studenarbeit
– Den Studenten, die bei unseren Usability-Tests mitgemacht haben

Inhaltsverzeichnis

Glossar

ADV Commons Von ADV Lib und ADV UI verwendete Bibliothek für gemeinsame Teile. 35, 41, 43

ADV Lib Enthält Schnittstellen zu allen unterstützen Datenstrukturen. Sie wird als Bibliothek im Code des Studenten eingebunden und überträgt die Daten an das ADV UI. 34, 35, 40, 41, 43

ADV UI Applikation für die Darstellung der Datenstrukturen. 4, 22, 34, 35, 39–41, 43

ADV-Tree-Module Produkttitel und Produkt der vorliegenden Studienarbeit. i–iii, 3–5, 12, 18–21, 23, 28, 30–34, 36–39, 43, 52

Algorithm & Datstructure Visualizer (ADV) Applikation zur Visualisierung von Algorithmen und Datenstrukturen. i, 8

Container Unabhängig laufende Einheit. 2, 3, 32–34, 37

Continous Integration und Deployment (CI/CD) Fortlaufende Erstellung von Artefakten und anschliessende Bereitstellung. vi, 14

Cron-Job Ein Cron-Daemon ermöglicht die zeitbasierte Ausführung von Prozessen in Linux. 15

Forest Ein Forest ist eine Sammlung unabhängiger Bäume. vi, 20, 21, 27, 28, 64, 65, 71

Github Ein Onlinedienst, der Software-Entwicklungsprojekte auf seinen Servern über das Versionsverwaltungssystem git bereitstellt. 10, 15

Graph-Visualization-Service (GVS) Vorgänger-Applikation des ADV. i, 8

Iteration Abschnitt des Entwicklungsprozesses in welchem Arbeiten in verschiedenen Disziplinen durchgeführt werden. 9, 11

JavaFX JavaFX ist ein Framework zur Erstellung plattformübergreifender Java-Applikationen. 2, 4, 39

Kontextdiagramm Darstellung der Schnittstellen von und zum System. 31

ModuleGroup Klasse, welche die Modul-Daten kapselt und sie zum ADV UI überträgt. 3, 33, 40, 41

Rational Unified Process (RUP) Iteratives Vorgehensmodell zur Softwareentwicklung, entwickelt von Rational Software. 9

Redmine Redmine ist eine freie, webbasierte Projektmanagementsoftware. 9–11, 15

Scrum Schlankes Vorgehensmodell zur agilen Softwareentwicklung, in welchem dem Kunden immer ein Produkt bereitgestellt wird. 9, 10

Snapshot Repräsentation einer Datenstruktur zu einem bestimmten Zeitpunkt. 4, 31, 40, 61, 72, 73

Story Points Bewertung der Schwierigkeit eines Arbeitspaketes durch die Vergabe von Punkten. 10

Story-Splitting Aufteilung eines zu grossen Arbeitspaketes in Kleinere. 9

Tree-Module Ein Teilmodul des ADV-Tree-Module. 31, 41, 51, 53

Wireframe Konzeptionelle Darstellung einer Ansicht. vi, 11, 18, 25

Akronyme

ADV Algorithm & Datstructure Visualizer. i–iii, 2–5, 12, 19–23, 30–32, 34, 38, 39, 43, 51, 52, 54, 58, 65, 71, 73, *Siehe:* Algorithm & Datstructure Visualizer (ADV)

CI/CD Continous Integration und Deployment. 14, *Siehe:* Continous Integration und Deployment (CI/CD)

GVS Graph-Visualization-Service. i, ii, 2, 3, 8, 19, *Siehe:* Graph-Visualization-Service (GVS)

Wald Forest. i, 2, *Siehe:* Forest

Abbildungsverzeichnis

Tabellenverzeichnis

Teil I

Technischer Bericht

KAPITEL 1

Technischer Bericht

1.1. Ausgangslage und Problembeschreibung

In einer vorangegangen Bachelorarbeit ist für die Visualisierung von Datenstrukturen und Algorithmen der ADV entwickelt worden. Er soll den im Unterricht eingesetzten GVS ablösen, da dieser nur die Darstellung von Bäumen und Graphen unterstützt. Der ADV ist modular aufgebaut und so kann unabhängig vom Kern eine neue Datenstruktur durch ein Modul hinzugefügt werden. Um den ADV produktiv einzusetzen fehlt ihm allerdings ein Modul für die Baumdarstellung.
Im Rahmen dieser Studienarbeit wird ein Modul zur Darstellung von Bäumen entwickelt. Die gesamte Funktionalität des GVS bezüglich der Bäume soll im Modul vorhanden sein. Zusätzlich wird es um die Darstellung des Arrays für binäre Bäume, die Generierung eines Binärbaumes aus einem Array und die Verbesserung der Usability bei bestehenden Funktionen erweitert.
Das Modul wird für den Studenten als Hilfestellung für das Erlernen von Baum-Datenstrukturen und den dazugehörigen Algorithmen verwendet. Damit der Student sich auf die Implementation konzentrieren kann, besteht die Möglichkeit den Code des Studenten und den Code für die Darstellung im ADV zu trennen. Bei groben Fehlern benachrichtigt das Modul den Studenten sofort über das Problem. Bei falscher Implementierung durch den Studenten wird es ihm in der Visualisierung aufgezeigt.
Das Modul nutzt den Kern des ADV für die Darstellung der Baumstrukturen und des Arrays. Darum wird es in Java entwickelt und die Darstellung erfolgt mit JavaFX.

1.2. Lösungskonzept

Das Modul beinhaltet vier verschiedene Funktionalitäten, deren Schnittstellen grösstenteils unabhängig voneinander sind: Darstellen eines binären Baumes mit Array, Darstellung eines Binärbaumes aus dem Array, Visualisierung eines generellen Baumes und die Darstellung eines Waldes. Aus diesem Grund sieht das Lösungskonzept eine Unterteilung des Moduls in vier Teilmodule vor, wobei gemeinsam benutzte Teile ausgelagert werden. Durch diese Kapselung wird dem Studenten die Benutzung vereinfacht. Er wird nicht mit Schnittstellen-Informationen überladen.
Wie der ADV teilt sich das Modul in zwei unabhängige Container auf. Der erste Container bietet dem Studenten die Modul-Schnittstellen und wird von ihm als Bibliothek in

seinen Code eingebunden. Der zweite Container ist für die Visualisierung der Baumstrukturen zuständig. Bei der Ausführung des Codes des Studenten werden die Daten vom ersten Container an den zweiten übermittelt.

1.3. Umsetzung

Die Umsetzung des ADV-Tree-Modules geschah schrittweise. Zur Vorbereitung wurde das Handbuch zur Modul-Entwicklung des ADV studiert. Dabei wurden auch die benötigten Entwicklungs-, Build- und Coverage-Tools eingerichtet. Nach der Ausarbeitung des Lösungskonzepts konnte mit der Entwicklung des Moduls gestartet werden.
Zur Überprüfung des Konzepts wurde in einer ersten Phase ein Prototyp erstellt, welcher die wichtigsten Komponenten der beiden Container Lib und UI beinhaltete. Nach erfolgreicher Implementierung wurde mit der Entwicklung der Kern-Funktionalität des Moduls begonnen. Dies beinhaltete das Übernehmen und Übersetzen der Features aus dem GVS. Zusätzlich wurde in diesem Schritt das Teilmodul implementiert, welches aus einem Array den Binärbaum berechnen und visualisieren kann.
In der darauffolgenden Phase wurden einige optionale Anforderungen implementiert. Gleichzeitig wurden die Vorbereitungen für die Usability-Tests getroffen. Die Durchführung der Usability-Tests geschah in der letzten Phase. Mit den Usability-Tests wurde die Bedienbarkeit des ADV-Tree-Modules geprüft. Änderungs- und Verbesserungsvorschläge wurden dokumentiert und grösstenteils auch umgesetzt.

1.4. Ergebnisdiskussion

Während der Studienarbeit wurde der ADV um das ADV-Tree-Module zur Darstellung von Trees erweitert. Die Funktionalität des GVS zum Darstellen von Trees wurde im ADV-Tree-Module übernommen. Zusätzlich zu den bestehenden Features kann das ADV-Tree-Module auch binäre Bäume aus Arrays erstellen.
Aufgrund eines beinahe reibungslosen Ablaufs bei der Entwicklung der Grundfunktionalitäten konnten weitere optionale Anforderungen implementiert werden. Diese zusätzlichen Features sollten vor allem dazu dienen, die Verständlichkeit der Darstellung der Bäume zu verbessern, wie beispielsweise das Fixieren der Nodes in Binärbäumen. Durch die Usability-Tests konnte die bestehende Funktionalität auf die Bedienbarkeit und Verständlichkeit geprüft werden. Dies bot weitere Möglichkeiten, das Modul zu verbessern.
Für die grundlegenden Anforderungen konnte das ADV-Tree-Module ohne Änderungen am ADV-Framework in die bestehende Architektur eingegliedert werden. Bei einigen optionalen Anforderungen erforderte die Implementierung jedoch eine Anpassung an den Core-Teilen der ADV-Projekte. So musste zum Beispiel für das Positionieren von Kind-Modulen die ModuleGroup-Klasse erweitert werden.
Code-Reviews und Refactorings während der Entwicklungs-Phase halfen, die geforderte Code-Qualität einzuhalten.

1.4.1. Eingetretene Risiken

Die erwarteten technischen Risiken konnten durch das Durchführen der Massnahmen vermieden werden. Durch die Einführung von Java 11 und das Beenden des Supports für Java 10 trat jedoch auch ein unerwartetes Problem ein. Die Lösung des Problems erforderte die Umstellung des ADV auf OpenJDK 11, die Auflösung von Konflikten der externen Bibliotheken mit der neuen Java-Version und ein separates Einbinden der JavaFX-Dependencies im ADV UI.

1.4.2. Offene Features

Aufgrund der begrenzten Dauer der Studienarbeit konnten einige optionale Anforderungen sowie einige Verbesserungsvorschläge aus den Usability-Tests nicht umgesetzt werden. Betroffen davon ist beispielsweise die optionale User-Story «Ein- und Ausblenden der Indizes im UI» aus den Anforderungsspezifikationen.
Weitere mögliche Features entstanden durch Ideen des Projektteams und durch Vorschläge aus den Usability-Tests. Diese betreffen zum Teil nicht nur das ADV-Tree-Module sondern das gesamte ADV-Framework.

Node-Beschreibung
: Neben den Tree-Nodes sollte eine zusätzliche Beschreibung eingefügt werden können, um Algorithmen genauer zu erklären. Beispielsweise bei den Rotationen im AVL-Baum, sollten die Nodes mit «X», «Y» und «Z» beschriftet werden können.

Snapshot Bugfix
: Beim Ausführen einer User-Codebase mit mehreren Snapshots ohne vorheriges Starten des ADV UI, endet das Programm in einer Endlosschleife. Dieser Fehler sollte behoben werden, sodass die Codebase in jedem Fall korrekt durchläuft und das UI entsprechend dargestellt wird. Ein erster Versuch zur Lösung dieses Problems wurde im Rahmen der Studienarbeit unternommen und ist im Abschnitt «Snapshot Bug» beschrieben.

1.4.3. Snapshot Bug

Während der Entwicklung des ADV-Tree-Modules kam es häufig dazu, dass eine User-Codebase ausgeführt wurde und das Programm in einer Endlosschleife endete. Durch mehrmaliges Ausprobieren konnte erkannt werden, dass das Problem nur eintrat, wenn eine User-Codebase mit mehreren Snapshots ohne vorheriges Starten des ADV UIs ausgeführt wurde. Sobald das Programm abgebrochen wird, erscheint die ADV-Benutzeroberfläche. Wird die User-Codebase nun erneut gestartet, beendet das Programm ohne Probleme und das Modul wird korrekt dargestellt.
Im Rahmen der Studienarbeit wurde bereits ein Anlauf unternommen, den Fehler zu beheben. Jedoch konnte das Problem bisher nicht genau geortet und entfernt werden. Es wird angenommen, dass es sich beim Problem um einen Nebenläufigkeits-Fehler im ADV UI handelt. Es könnte sein, dass es zwischen der ADVApplication-Klasse und dem SocketServer-Thread, welcher aus der ADVApplication gestartet wird, zu einem Deadlock kommt, welcher zur Blockierung des Programms führt. Herausgefunden wurde zudem, dass das Problem nicht auftritt, wenn die Logger-Einträge weder in ein File

noch in die Konsole geschrieben werden. Wenn zum Beispiel in der User-Codebase ein eigenes logback.xml-File erstellt wird und darin die `appender-ref` -Zeilen auskommentiert werden, dann werden keine Einträge in die Konsole oder ins Log-File geschrieben. In diesem Fall läuft das Programm durch.

1.4.4. Ausblick

Insgesamt freut sich das Projektteam über das entstandene ADV-Tree-Module. Durch die Arbeit mit dem bestehenden ADV-Framework entstanden nur selten Probleme, welche wiederum schnell gelöst werden konnten.
In der Zukunft können sowohl der ADV als auch das ADV-Tree-Module durch weitere Features erweitert werden. Mit der Unterstützung der Visualisierung von Bäumen enthält der ADV die gesamte Funktionalität des GVS. Damit kann der ADV den GVS in den Unterrichts-Modulen Algorithmen und Datenstrukturen 1 und 2 vollumfänglich ablösen.

Teil II

Projektdokumentation

KAPITEL 2

Projektplan

Änderungsgeschichte

Datum	Version	Änderung	Autor
24.09.2018	0.1	Dokument erstellt	Team
05.10.2018	0.2	Überarbeitung nach Betrachtung	Team
10.12.2018	1.0	Überarbeitung der Meilensteine	Team

Tabelle 2.1.: Änderungsgeschichte

2.1. Projekt Übersicht

2.1.1. Einführung Projektplan

Dieses Dokument bietet eine Übersicht über das Projekt ADV-Tree-Module. Es beschreibt die Organisation, das Management und die Massnahmen zur Qualitätssicherung während des Projekts.

2.1.2. Ausgangslage

In den Modulen Algorithmen und Datenstrukturen 1 und 2 wurde bisher der Graph-Visualization-Service (GVS) eingesetzt. Dieser half den Studenten die Inhalte der Module zu visualisieren und so besser zu verstehen. Der GVS konnte jedoch nur Graphen und Trees visualisieren und war nicht einfach erweiterbar. Aufgrund dieser Ausgangslage wurde während der Bachelorarbeit [5] im Frühlingssemester 2018 ein neues Tool, der Algorithm & Datstructure Visualizer (ADV), entwickelt. Dieser bietet gegenüber dem GVS sowohl die Erweiterung durch Module als auch bessere Usability.

2.1.3. Zweck und Ziel der Studienarbeit

Im Rahmen der Bachelorarbeit wurden bereits erste Module entwickelt. Darunter auch die Darstellung von Arrays und Graphen. Gegenüber dem GVS fehlt allerdings die Möglichkeit Bäume zu visualisieren. Das Ziel dieser Studienarbeit ist, ein weiteres Modul für die Darstellung von Bäumen zu entwickeln. Ausserdem soll es möglich sein, Bäume zu visualisieren, die in Arrays gespeichert sind. Dabei soll auch der Inhalt des Arrays angezeigt werden.
Zudem möchten wir während der Studienarbeit weitere Erfahrungen im Bereich Software-Engineering sammeln. Ebenfalls können wir bisher gelerntes Wissen aus diversen Studien-Modulen in einem Software-Projekt praktisch anwenden.

2.2. Projektorganisation

Das Team ist in gleichgestellten Mitgliedern organisiert. Alle Arbeiten werden selbstständig innerhalb des Teams aufgeteilt.

2.2.1. Externe Schnittstellen

Die Betreuung der Studienarbeit übernimmt Herr Thomas Letsch. Die Kommunikation findet hauptsächlich per Mail oder über Besprechungen statt.

2.2.2. Arbeitspakete

Um das Projekt zu einem erfolgreichen Ende zu bringen, wird mit einem agilen iterativen Entwicklungsprozess mit Rahmenbedingungen (vorgegebene Termine) gearbeitet. Damit dies einfacher umsetzbar wird, verwenden wir in dieser Studienarbeit das Projektmanagement-Tool Redmine. Es bietet zahlreiche Funktionen, die einen agilen Entwicklungsprozess vereinfachen. Alle Arbeitspakete werden dazu in Redmine erfasst. Beim Erfassen wird der Aufwand des Pakets geschätzt. Damit ein Arbeitspaket in einer Iteration abgeschlossen werden kann, sollte der geschätzte Aufwand höchstens 17 Stunden entsprechen. Arbeitspakete, die diese Limite überschreiten werden mittels Story-Splitting aufgeteilt.

2.2.3. Iterationsplanung

Um einen iterativen und agilen Entwicklungsprozess zu garantieren, werden die Iterationsphasen (Inception, Elaboration, Construction und Transition) aus dem Rational Unified Process (RUP) benutzt. Innerhalb der Iterationen werden Sprints nach Scrum durchgeführt. Die genaue Aufteilung der Phasen und der Sprints sind der Artefakt-Übersicht (Siehe Anhang) zu entnehmen. Vonseiten der Aufgabenstellung [2] sind wichtige Termine, wie der zweitletzte Release am Dienstag, 4. Dezember 2018 um 07:00 Uhr, der letzte Release am Freitag, 21. Dezember 2018 um 12:00 Uhr und die Abgabe des Abstracts bis am Dienstag, 18. Dezember 2018 zu beachten. Aus diesem Grund werden die Meilensteine und Sprints entsprechend an die vorgegebenen Termine angepasst.
Vor Beginn einer Iteration werden sämtliche Arbeitspakete zugeordnet, die darin abgearbeitet werden. Dabei soll die total geschätzte Zeit den Wert von 55 Stunden nicht überschreiten. Dies soll die Wahrscheinlichkeit erhöhen, dass alle gewünschten Features implementiert werden. Sollten vor Ablauf einer Iteration alle Arbeitspakete erfüllt sein, so können Arbeitspakete von späteren Iterationen vorgezogen werden.

2.2.4. Schätzungen und Zeitauswertung

Als Abweichung zu Scrum verwenden wir zur Schätzung keine Story Points sondern direkt die Zeit in Stunden (bspw. 1.5h), was Redmine so auch standardmässig unterstützt. Dadurch kann Redmine auch direkt eine Zeitauswertung erstellen. Redmine unterstützt auch den Export der Zeitauswertung. Aus diesem Grund werden die Auswertungs-Diagramme mit Excel erstellt.

2.2.5. Wochenbesprechungen

Während der Dauer der Studienarbeit finden jede Woche, am Mittwochmorgen um 08:30 Uhr, Besprechungen mit dem Betreuer statt. Der Inhalt dieser Besprechungen sind der Rückblick auf die letzte Woche, aktuelle Ereignisse (Fragen, Entscheidungen, etc.) und der Ausblick auf die nächste Woche. Es wird auch ein Protokoll vom Team erstellt und anschliessend an den Betreuer geschickt. Das Protokoll wird ausserdem in einem privaten Github-Repository abgelegt, um es im Team einfacher auszutauschen (siehe Tabelle 2.3).

2.2.6. Arbeitstage

Das Team arbeitet durch den gegebenen Studenplan jeweils am Montag und Freitag zusammen an der HSR. Aus diesem Grund ist auch das Iterationsende (Freitag) gegeben.
Falls wegen Zeitknappheit auch an einem anderen Tag gearbeitet wird, kommuniziert das Team bei Fragen per WhatsApp, um so möglichst schnell eine Antwort zu bekommen.

2.3. Management Abläufe

2.3.1. Zeitaufwand

Da die Studienarbeit 8 ECTS ergibt, sollte der totale Zeitaufwand bei etwa 480 Arbeitsstunden liegen. Auf die 14 Semesterwochen aufgeteilt ergibt dies pro Person einen Aufwand von etwa 17 Stunden.
Grundsätzlich sind die Iterationsziele einzuhalten. Wenn erkannt wird, dass ein Ziel nicht erreicht werden kann, dann sollte für die Iteration mehr Zeit aufgewendet werden. Falls dies trotzdem nicht reicht, darf ein Arbeitspaket auf die nächste Iteration verschoben werden.

2.3.2. Zeitliche Planung

Die zeitliche Planung wird mit Redmine realisiert. Der Redmine-Server ist über `http://sinv-56071.edu.hsr.ch/redmine/` erreichbar. Während der Bearbeitung eines Arbeitspakets wird die aufgewendete Zeit regelmässig aktualisiert. Dies ermöglicht schliesslich einen Vergleich der geplanten und aufgewendeten Zeit.
Eine Übersicht über sämtliche Iterationen und geplanten Artefakte ist in der Artefakt-Übersicht (Siehe Anhang) enthalten. Die aktuellste Version ist unter `https://github.com/W1nt0r/adv-dokumentation/tree/master/01_Projektplan/Releases` verfügbar. Ausserdem ist dort auch eine Beschreibung des Inhalts der Versionen enthalten.

2.3.3. Meilensteine

Eine Übersicht der Meilensteine ist in der Artefakt-Übersicht zu finden. In diesem Abschnitt wird erklärt, was in den Meilensteinen erarbeitet werden wird.

21.09.2018	MS0	Vision – Redmine eingerichtet, inkl. Backups – Dokumentations-Template erstellt
05.10.2018	MS1	Projektplan und Anforderungsspezifikation – Meilensteine festgelegt – Werkzeuge festgelegt – Qualitätshilfsmittel festgelegt – Technische Risiken gesammelt und aufgelistet – Anforderungen gesammelt und dokumentiert – Wireframe Prototypen erstellt

19.10.2018	MS2	Architektur- und Designspezifikation sowie Prototyp – Werkzeuge aufgesetzt und funktionsfähig – Technische Risiken abgedeckt – Architektur- und Designentscheidungen dokumentiert – Lauffähiger Prototyp erstellt – Anforderungsspezifikation für Implementation fertiggestellt
16.11.2018	MS3	Release 1 – Grundfunktionen des ADV-Tree-Module implementiert – Lauffähig für Usability-Tests
30.11.2018	MS4	Usability Tests – Usability-Tests erfasst und durchgeführt – Erste System-Tests erfasst und durchgehführt – ADV-Tree-Module gemäss Usability-Test-Resultaten angepasst
14.12.2018	MS5	Release 2 – Weiterentwicklung ADV-Tree-Module – Alle Module bereit für Release – Abstract erstellt – Zweitletzer Release des ADV-Tree-Modules
21.12.2018	MS6	Dokumentation – Dokumentation fertiggestellt – Release ADV v2.0

Tabelle 2.2.: Beschreibung Meilensteine

2.4. Risikomanagement

Die Risiken werden in der Excel-Datei 'TechnischeRisiken' (Siehe Anhang) erfasst. Eine aktuelle Version ist unter `https://github.com/W1nt0r/adv-dokumentation/tree/master/01_Projektplan/Releases` verfügbar. Da die Erfassung der Risiken ein Bestandteil des Projektplans ist, wird die Version für das Release entsprechend des Projektplans geführt. Daher werden die Technischen Risiken in der Artefakt-Übersicht nicht aufgelistet.

2.5. Qualitätsmassnahmen

Durch die Bachelorarbeit ADV [5] sind einige Werkzeuge zur Sicherung der Softwarequalität schon vorgegeben. Im Rahmen dieser Studienarbeit haben wir diese erweitert oder entsprechend angepasst. Grundsätzlich gilt die Anforderung, dass der Ausfall einer Komponente höchstens einen Arbeitsverlust von 8 Stunden verursachen darf.

2.5.1. Metrik-Tools

Die Metrik-Tools werden entsprechend der Bachelorarbeit ADV [5] übernommen und so weiter eingesetzt. Nähere Informationen sind in der Dokumentation der Bachelorarbeit zu finden.

2.5.2. Continous Integration und Deployment (CI/CD)

Ebenfalls wird die gleiche CI/CD Kette aus der Bachelorarbeit ADV [5] wiederverwendet. Darunter befindet sich auch das Build-Tool Gradle.

2.5.3. Repositories

Die Sourcecode-Repositories werden von der Bachelorarbeit ADV [5] übernommen. Weitere Repositories wurden speziell für diese Studienarbeit eingerichtet. Der Nutzen dieser Repositories wird in den folgenden Abschnitten erklärt.

Repository	Art	Link
adv-dokumentation	Dokumentation	`https://github.com/W1nt0r/adv-dokumentation`
adv-besprechungen	Protokoll	`https://github.com/W1nt0r/adv-besprechungen`
adv-backup	Backup	`https://github.com/Supernaibaf/adv-backup`

Tabelle 2.3.: Zusätzliche Repositories

2.5.4. Dokumentation

Für alle Dokumente wurde das private Github-Repository adv-dokumentation (Tabelle 2.3) erstellt. Dies ermöglicht die gleichzeitige Bearbeitung am gleichen Dokument und die Versionisierung der Dokumente. Ausserdem gibt es so auch eine automatische Sicherung der Dokumente. Durch den agilen Entwicklungsprozess besteht während dem Projekt auch die Möglichkeit die Dokumente im Verlauf des Projektes anzupassen und zu verfeinern.
Um grössere Verluste zu verhindern, werden regelmässig Commits erstellt und ins Github-Repository hochgeladen. Dies geschieht mindestens nach Beendigung einer Section im Dokument.

2.5.5. Backup

Auf dem virtuellen Server mit der Redmine Instanz werden täglich alle Daten von Redmine auf dem Server gesichert. Damit die Daten bei einem Ausfall des Servers nicht verloren gehen, haben wir ein privates Backup-Repository (Tabelle 2.3) eingerichtet. Auf dem virtuellen Server wird täglich ein Cron-Job-Job ausgeführt, welcher die gesicherten Daten auf das Repository sichert.

2.5.6. Testen

Unit Tests

Für das Unit-Testing verwenden wir die bestehenden Frameworks aus der Bachelorarbeit ADV [5]. Es werden Microtests auf Klassenstufe, sowie Integration Tests auf Package Stufe erstellt. Diese dienen dazu, das Zusammenspiel der Komponenten zu testen.

Systemtests

Leider kann nicht alles automatisiert getestet werden. Aus diesem Grund werden Testcases, welche nicht mit Integration Tests abgebildet werden können, als Systemtests formuliert und von Hand durch die Entwickler getestet. Das Resultat umfasst ein Protokoll mit den Ergebnissen und im Fehlerfall genaue Details zur Abweichung.

Usability Tests

Da der Entwickler mit der Zeit eine eingeschränkte Sicht auf das Produkt bekommt, werden im Rahmen der Studienarbeit etwa in der Mitte der Construction-Phase Usability Tests mit Personen durchgeführt, die das Modul AD2 schon besucht haben oder gerade besuchen.

2.5.7. Entwicklung

Vorgehen

Für jedes zu implementierende Feature wird ein Arbeitspaket erfasst. Vor der Implementation eines Features weist sich der Entwickler das Arbeitspaket zu und aktualisiert

dies danach durch regelmässige Erfassungen des Aufwands.
Auch hier werden grössere Verluste durch regelmässige Commits vermieden. Dies geschieht mindestens nach dem Entwickeln eines funktionsfähigen Feature-Teils und den zugehörigen Unit-Tests.

Code Reviews

Um die Code-Qualität zu verbessern werden regelmässig manuelle Code-Reviews durchgeführt. Die Code-Reviews finden jeweils am Freitag statt und sollten etwa eine Stunde dauern. Dabei wird jeweils ein Code-Stück ausgewählt und analysiert. Unschöne Stellen oder Fehler werden im Review-Arbeitspaket protokolliert. Dabei wird erfasst, um welchen Code-Teil es sich handelt, wie schwer der Fehler ist und bis wann er behoben sein soll. Am Ende des Reviews werden für die Vorfälle Arbeitspakete definiert und einem Entwickler zugewiesen.

Code Style Guidelines

Da die Qualitätswerkzeuge aus der ADV-Bachelorarbeit [5] übernommen werden, gilt dies auch für die Code-Guidelines. Die definierten Checkstyle-Regeln werden übernommen.

Definition of Done

Ein Arbeitspaket gilt als abgeschlossen, sobald sämtliche Akzeptanzkriterien erfüllt sind. Ebenfalls darf keines der Qualitäts-Tools einen Error oder eine Warning zeigen. Des Weiteren müssen alle Unit-Tests ohne Fehler ausführbar sein und dürfen nicht fehlschlagen.

KAPITEL 3

Anforderungsspezifikationen

Änderungsgeschichte

Datum	Version	Änderung	Autor
01.10.2018	0.1	Dokument erstellt	Team
12.10.2018	0.2	Überarbeitung nach Betrachtung	Team
15.10.2018	0.3	Use Case im Fully-dressed Format	Team
09.11.2018	0.4	Neue User Stories hinzugefügt	Team
22.11.2018	0.5	Neue optionale User Stories hinzugefügt	Team
17.12.2018	1.0	User-Story Prefixes hinzugefügt	Team

Tabelle 3.1.: Änderungsgeschichte

3.1. Einführung

Der Zweck dieses Dokuments liegt darin, sämtliche Anforderungen an das ADV-Tree-Module aufzulisten und zu spezifizieren. Die Requirements werden in die Untergruppen funktionale Anforderungen und Qualitätsanforderungen unterteilt. Die Wireframes zeigen einen Umsetzungs-Entwurf der Anforderungen im User-Interface.

3.2. Funktionale Anforderungen

Das folgende Kapitel beschreibt sämtliche Anforderungen an die Funktionalität desADV-Tree-Modules. Dabei sollen alle Funktionen zur Darstellung von Trees aus dem GVS 2.0 beibehalten werden. Einschränkungen dieser Funktionalität sind mit dem Auftraggeber abzusprechen.

3.2.1. Stakeholder und Aktoren

In diesem Abschnitt werden die Stakeholder des ADV-Tree-Modules beschrieben und welche Anforderungen sie an dieses stellen. Die Aktoren entsprechen beim ADV-Tree-Module den Stakeholdern.

Der Student

Zum aktuellen Zeitpunkt muss der Student das GVS [4] allein oder gemischt mit dem ADV [5] verwenden. Aus Komfortgründen will der Student nur noch den ADV benützen. Er erwartet, dass er Bäume (generelle und binäre) ohne Arraydarstellung, wie gehabt (aus dem GVS) im ADV anschauen und Schritt für Schritt durchnavigieren kann. Zusätzlich will er bei einem Binary Tree das Array selber mit der Verknüpfung zur Baumstruktur angezeigt bekommen.

Der Dozent

Der Dozent möchte wie im GVS (Siehe Studienarbeit zum GVS 2.0 [4]), dass der Student die Funktionalität eines Baumes implementiert. Er selber will den Gerüstcode für das ADV-Tree-Module bereitstellen, damit der Student fast nichts vom ADV-Tree-Module mitbekommt.

3.2.2. User Stories

Die User Stories zeigen die Anforderungen an das System aus der Sicht eines Benutzers.

US1: Darstellung Tree mit Root

Als Student möchte ich einen generellen Tree oder Binary Tree durch das Übergeben der Root im ADV darstellen lassen. So kann ich den Aufbau des Trees und die Beziehungen zwischen den Nodes besser verstehen.

US2: Array-Darstellung aus Binary Tree mit Root

Als Student möchte ich beim Übergeben eines Binary Tree mit Root das zu diesem Binary Tree gehörende Array berechnen und anzeigen lassen. Dadurch lerne ich wie ein Binary Tree in einem Array gespeichert werden kann.

US3: Darstellung Binary Tree aus Array

Als Student möchte ich ein Array bzw. eine Array-List übergeben können, aus dem der ADV den dazugehörigen Binary Tree berechnet und visualisiert. Dadurch kann ich das Speichern eines Binary Trees im Array besser verstehen.

US4: Ein- und Ausblenden des Arrays

Als Student möchte ich das grundlegende Array des Binary Trees ein- und ausblenden können. So kann ich störende Elemente verbergen, wenn ich nur die Tree-Struktur betrachten möchte.

US5: Darstellung von Forest

Als Student möchte ich mehrere Nodes einzeln oder als Collection übergeben können. DasADV-Tree-Module sucht sich die Beziehungen zwischen den Nodes selbst und stellt die entsprechenden Trees dar. Durch das Übergeben aller Nodes kann ich auch Forests darstellen lassen.

US6: Benennen der Nodes

Als Student möchte ich die Nodes eines Trees benennen können, damit ich diese nach einer Änderung in der Struktur des Trees schnell wiedererkenne.

US7: Node und Edge Styling

Als Student möchte ich eine Node oder Edge im Tree mit Styles hervorheben können, sodass ich bei einem Tree-Algorithmus beispielsweise die aktuell betroffenen Elemente erkenne. Falls auch das Array ausgegeben wird, sollen darin die entsprechenden Einträge ebenfalls markiert werden.

US8: Optional: Darstellung von Array unterhalb von Binary Tree

Als Student möchte ich bei einem Binary Tree, dass das ADV-Tree-Module das Array gleich unter dem Tree darstellt. Dadurch steigert sich für mich die Übersichtlichkeit gegenüber der Darstellung neben dem Binary Tree.

US9: Optional: Fixieren von Tree-Nodes

Als Student, möchte ich, dass die Nodes eines Trees beim Durchgehen der Session-Navigation fixiert sind, sodass ich bei einer Rotation sofort erkenne, welche Nodes ihre Position geändert haben und welche nicht.

US10: Optional: Anzeigen der Indizes im Array-Modul

Als Student möchte ich beim Array-Modul die Indizes der einzelnen Einträge anzeigen lassen. Damit kann ich bei der Darstellung der Binary Trees mit Arrays die Array-Nodes den Tree-Nodes besser zuordnen.

US11: Optional: Ein- und Ausblenden der Indizes im UI

Als Student möchte ich die Indizes und den Array beim Binary Tree mit Array direkt im UI ein- und ausblenden können. Dies erlaubt das Ein- und Ausblenden mit einem Klick und benötigt keinen zusätzlichen Code im User-Codebase.

US12: Optional: Styling des Binary Tree aus dem Array

Als Student möchte ich, wenn ich ein Array oder eine Array-List übergebe, zusätzlich Styles für die Nodes angeben können. Bei der Anwendung von Algorithmen ermöglicht mir dies eine bessere Übersichtlichkeit und Verständlichkeit des angewandten Algorithmus.

US13: Optional: Erweiterung um Kind-Module

Als Student möchte ich die Möglichkeit haben vorhandene Module in das ADV-Tree-Module einzubinden. Dies ermöglicht mir bei der Anwendung von Algorithmen weitere Einsichten auf die anderen Datenstrukturen, die bei diesem Algorithmus zum Einsatz kommen.

US14: Optional: Darstellung der Forests aus den Root-Knoten

Als Student möchte ich die Möglichkeit haben einen Forest nur durch die Angabe der Root-Knoten darstellen zu können. Dies erlaubt mir das schnelle Hinzufügen eines Trees in den Forest.

3.2.3. Use Case Beschreibungen

Die Hauptaufgabe des ADV-Tree-Modules ist die Visualisierung von Trees mit und ohne Arraydarstellung. Aus diesem Grund gibt es nur wenige Use Cases. Alle anderen Details sind im Kapitel User Stories enthalten.

Tree darstellen

Ziel	Der Student möchte den von ihm implementierten Tree darstellen lassen, um eine bessere Übersicht zu erhalten.
Level	User-Goal
Hauptakteur	Student
Stakeholder	Der Dozent möchte, dass der Student möglichst einfach seine Implementation im ADV darstellen lassen kann.
Preconditions	Der Gerüstcode zur Benutzung des ADV-Tree-Modules und die Implementation des Trees müssen verfügbar sein.
Postconditions	Der ADV ist aufgestartet und zeigt den Tree im Fenster des ADV an.

Hauptszenario	1. Der Student startet den Gerüstcode. 2. Das System startet den ADV. 3. Das System übermittelt die Nodes und Edges an den Server (ADV UI). 4. Das System zeigt den implementierten Baum graphisch an.
Erweiterungen	1a. Der Student erstellt Code, der das Modul aufruft und ihm ein Array als Parameter mitschickt. 3a. Das System übermittelt die Array-Elemente an den Server (ADV UI). 4a. Das System berechnet aus dem Array einen Binary-Tree. 1. Das System unterteilt die Ansicht in den Teil Tree-Darstellung und Array-Darstellung. 2. Das System zeigt den Binary-Tree zum übergebenen Array an. 3. Das System zeigt das übergebene Array an.
Häufigkeit des Auftretens	Mehrmals pro AD-Übungslektion

Tabelle 3.2.: Fully-dressed Use Case Tree darstellen

3.3. Qualitätsanforderungen

3.3.1. Austauschbarkeit

Das ADV-Tree-Module sollte generell implementiert werden. Generell heisst in diesem Fall, dass es keine Rolle spielt, ob ein Knoten zwei Kind-Knoten (binär) oder n Kind-Knoten hat. Die einzige Ausnahme hierzu bildet ein Baum mit Arraydarstellung. Dort wird nur ein Binary Tree unterstützt. Diese generelle Implementation ermöglicht, dass die dahinterliegende Baumimplementation gegen eine andere Baumimplementation ausgetauscht werden kann.

3.3.2. Verständlichkeit Tree

Ein Student der die Module Algorithmen und Datenstrukturen 1 und 2 besucht hat, sollte wichtige Elemente wie Root, Internal Nodes und Leaves in der Darstellung benennen können.

3.3.3. Verständlichkeit Array-Tree

Ein Knoten und der Array-Eintrag sollten so gekennzeichnet sein, dass der Student zuordnen kann, welcher Array-Eintrag zu welchem Knoten gehört.

3.3.4. Modularität

Das ADV-Tree-Module sollte so entwickelt werden, dass es vom ADV als Modul eingebunden werden kann und dass es wie die übrigen Module (Array, Graph, etc.) in andere Module integriert werden kann.

3.3.5. Erlernbarkeit

Der Gerüstcode für einen einfachen Baum (ohne Ausbalancieren, einfügen, etc.) sollte vom Dozenten über ein einfaches Codebeispiel im Benutzerhandbuch implementiert werden können. Um kompliziertere Beispiele (mit Einfügen, Ausbalancieren, etc.) zu ermöglichen, sollte der Dozent mithilfe der Dokumentation im Benutzerhandbuch den Gerüstcode erstellen können.

3.3.6. Testbarkeit

Die Klassen sollten so konzipiert werden, dass sie in Isolation getestet werden können und somit Microtests möglich sind.

3.3.7. Nachvollziehbarkeit

Um Fehler im Programm nachvollziehen zu können soll ein zweidimensionales Logging eingesetzt werden (Siehe Richtlinien der Studienarbeit [3]). Das Logging besteht aus zwei Dimensionen. In der ersten Dimension wird das Package angegeben und in der zweiten das Log-Level (wie Info, Warning, Error, etc.). Das Log-Level pro Package muss zur Laufzeit änderbar sein (z.B. bei Package A sollen nur Errors geloggt werden und bei Package B auch Warnings).

3.4. Wireframes

Dieses Kapitel enthält die Wireframes zur Darstellung der verschiedenen Tree-Varianten.

3.4.1. Node Tree

Die Basis-Variante der Node Tree Darstellung enthält nur den Tree selber ohne andere Zusätze, wie die Array-Darstellung.

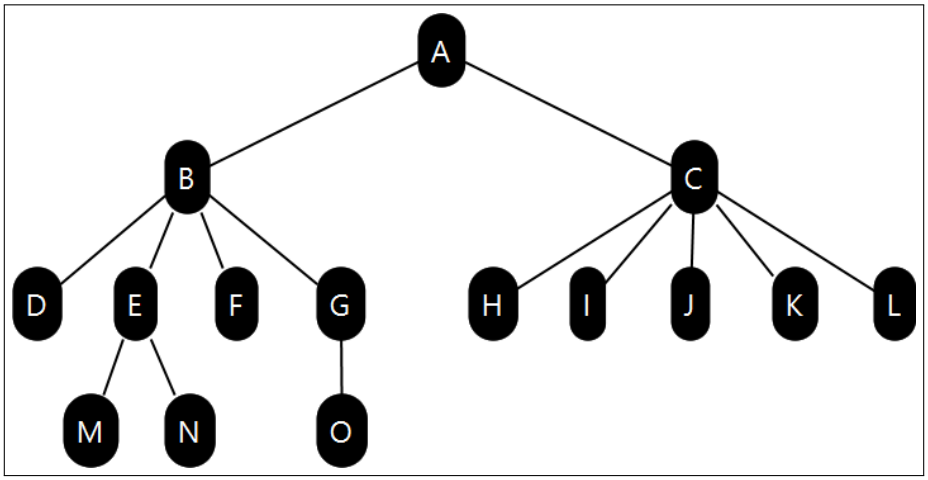


Abbildung 3.1.: Genereller Tree mit Root

3.4.2. Array Tree

Beim Array Tree können nur Binary Trees dargestellt werden. Es gibt die Möglichkeit nur den Tree darzustellen oder den Tree und das Array.

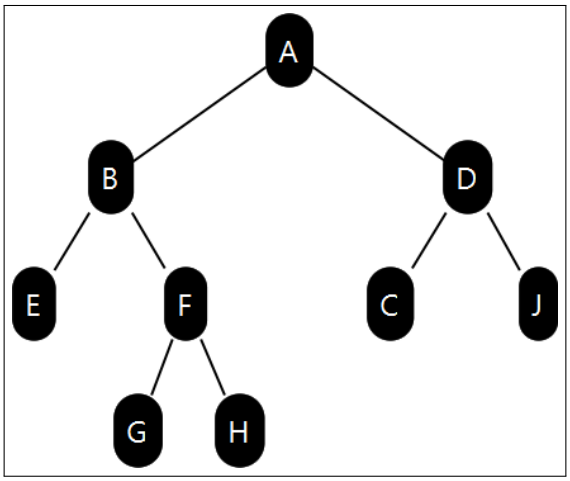


Abbildung 3.2.: Binary Tree

Variante Indexed

In der Variante Indexed werden beim Tree die Array-Indizes neben den entsprechenden Knoten angezeigt, um dem Studenten einen besseren Überblick zu geben, wo er das Element im Array finden kann.

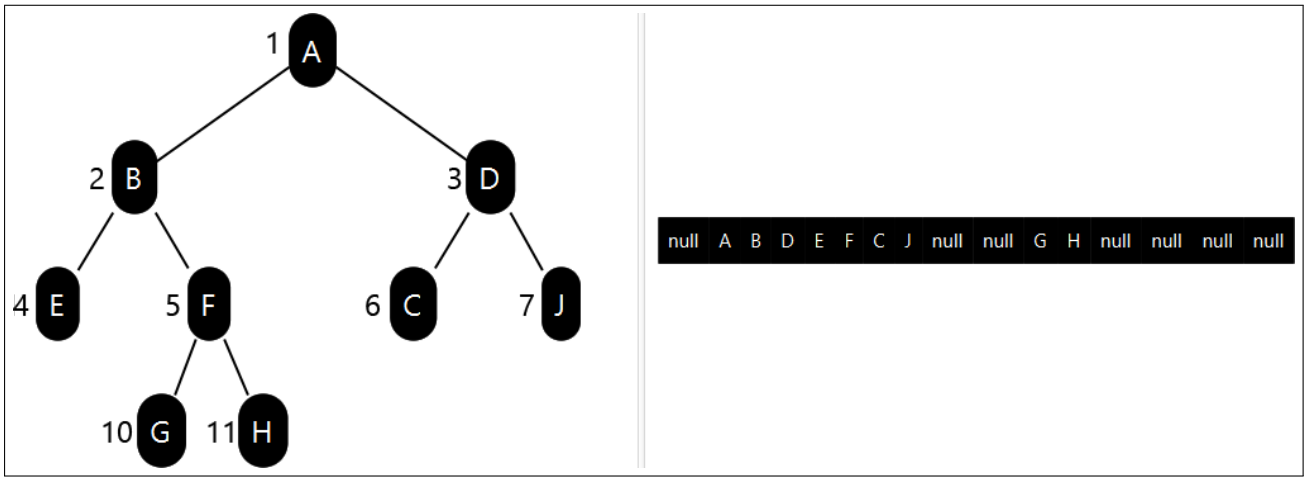


Abbildung 3.3.: Binary Tree mit Array-Darstellung

Variante Indexed und Marked

In der Variante Indexed und Marked wird die Möglichkeit angeboten, einen Knoten des Trees und die entsprechende Stelle im Array zu markieren. Dies kommt zum Beispiel beim Einfügen eines Knoten in den Tree zum Einsatz.

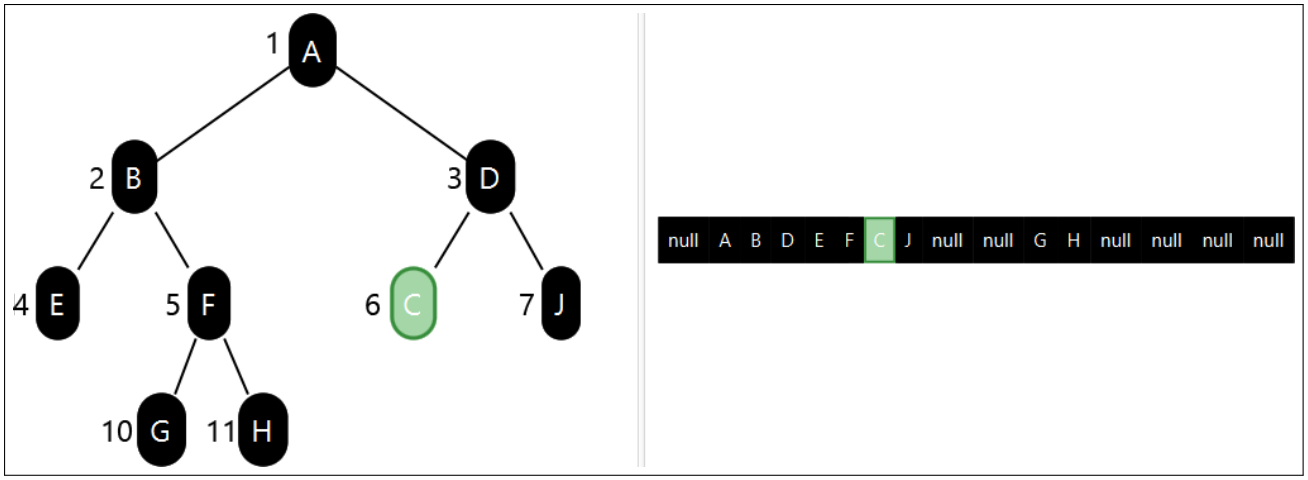


Abbildung 3.4.: Markierte Node im Binary Tree mit Array-Darstellung

3.4.3. Forests

Die Darstellung der Forests ist ähnlich zur Visualisierung der Node Trees. Der Unterschied liegt darin, dass mehrere unabhängige Trees nebeneinander angezeigt werden können.

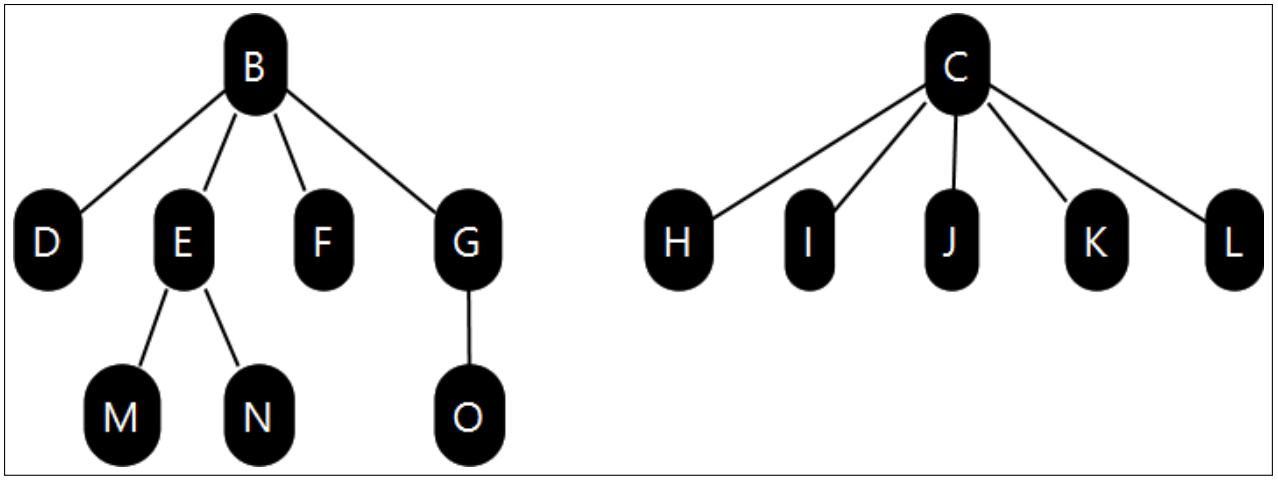


Abbildung 3.5.: Darstellung eines Forests

3.5. Domain Modell

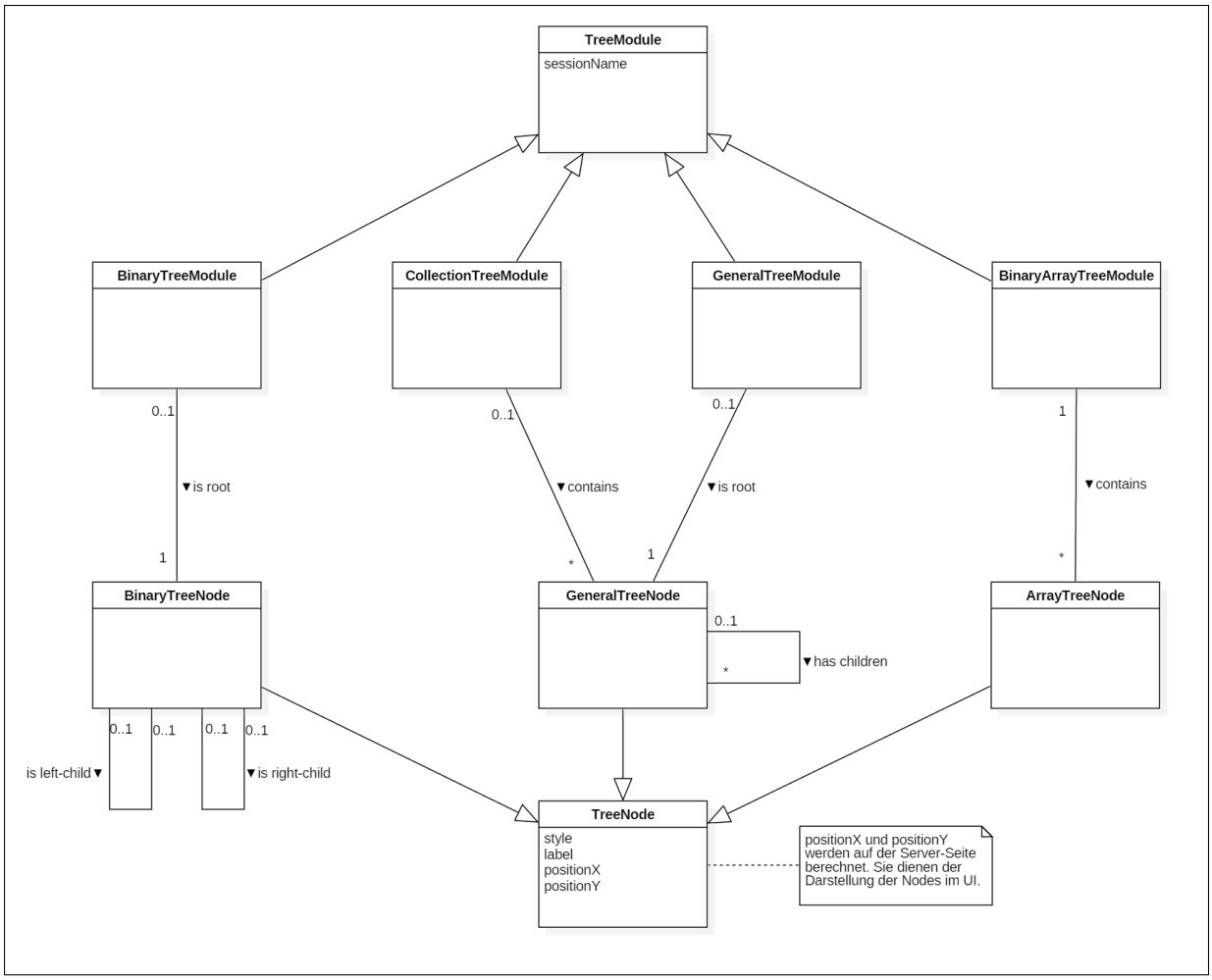


Abbildung 3.6.: Domain-Modell des ADV-Tree-Modules

3.5.1. TreeModule

Das Tree Module bildet die Hauptkomponente des ADV-Tree-Modules. Es ist unterteilt in die vier Hauptaufgaben des Moduls. Dies sind das Darstellen von Trees mit Root (BinaryTreeModule und GeneralTreeModule), das Darstellen von Forests (CollectionTreeModule) und das Darstellen von Binary Trees aus Arrays (BinaryArrayTreeModule).

3.5.2. TreeNode

Die Nodes sind die Bestandteile der Trees. Auch hier wird für jede Aufgabe eine passende Node gewählt.
Da die ArrayTreeNode keine Children hat kann sie nur für das BinaryArrayTreeModule verwendet werden.

KAPITEL 4

Architektur- und Designspezifikation

Änderungsgeschichte

Datum	Version	Änderung	Autor
08.10.2018	0.1	Dokument erstellt	Team
15.10.2018	0.2	Abhängigkeit zu Array-Modul und Übermittlung Client – Server hinzugefügt	Team
26.10.2018	0.3	Umstellung Java 11 und neue JavaFX Besonderheiten dokumentiert	Jan Winter
09.11.2018	0.4	Kapitel Kind-Module hinzugefügt	Team
26.11.2018	1.0	Anpassung Umstellung Java 11 und Umsetzung Fixieren von Tree-Nodes	Fabian Meier
07.12.2018	1.0	Erweiterung um Modul Positionierung und BinaryArrayTreeEnhancedModule	Team

Tabelle 4.1.: Änderungsgeschichte

4.1. Einführung

Dieses Dokument gibt einen Überblick über die Architektur- und Designentscheidungen des ADV-Tree-Modules. Es ist stark mit den Qualitätsmassnahmen aus der Anforderungsspezifikation verknüpft.
Da in dieser Studienarbeit der ADV weiterentwickelt wird, werden die Grundstrukturen der Software-Architektur von dort übernommen. Dieses Dokument zeigt nur diejenigen Komponenten, die im Verlaufe des Projekts zu der existierenden Architektur hinzugefügt werden.
Genaue Angaben zur Architektur des ADV sind in der ADV-Bachelorarbeit [5] zu finden.

4.2. Kontextdiagramm

Das Kontextdiagramm zeigt das zu implementierende ADV-Tree-Module und dessen externe Abhängigkeiten. Der Student möchte das Tree-Module in seiner Codebase benutzen. Sobald er die Codebase ausführt und den ADV startet fordert der ADV die notwendigen Daten beim Tree-Module an, um den Baum zu visualisieren.

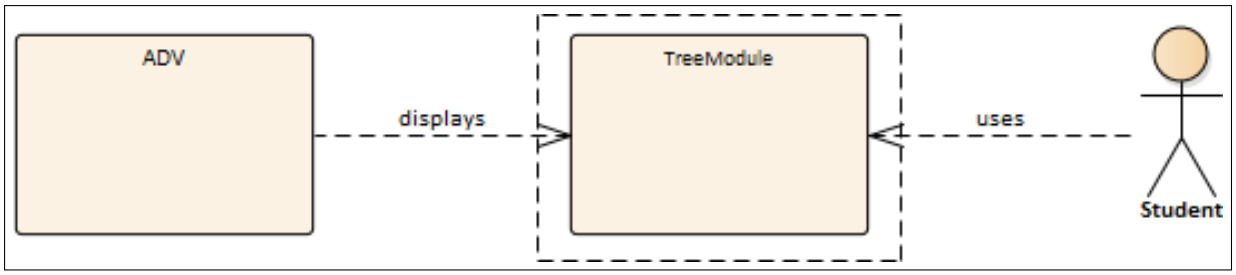


Abbildung 4.1.: Kontextdiagramm des ADV-Tree-Modules

4.2.1. Schnittstellen

Student

Der Student möchte das ADV-Tree-Module in seiner Codebase benutzen. Der Dozent hat ihm dazu den Gerüstcode bereitgestellt, der das Modul integriert, um die Baumimplementationen im UI darzustellen. Sobald der Student seine Implementation fertiggestellt hat, führt er den Code aus, worauf der ADV gestartet wird und einen Snapshot des Tree-Modules erstellt.

ADV

Bei einem Snapshot des Tree-Modules greift der ADV auf die vom Student implementierten Nodes zu. Diese werden daraufhin serialisiert und an das UI gesendet. Das UI fordert dann die zur Visualisierung benötigten Informationen vom Tree-Module an. Dazu gehört beispielsweise die Information, wo und wie eine Node im UI dargestellt werden soll.

4.3. Schichten

Durch den Aufbau des ADV ist die Aufteilung des Modules in die drei Container UI, Lib und Commons vorgegeben. Das ADV-Tree-Module verwendet innerhalb dieser Container das Layer-Pattern [6], um die Software zu strukturieren. Daraus resultiert das folgende Schichtenmodell:

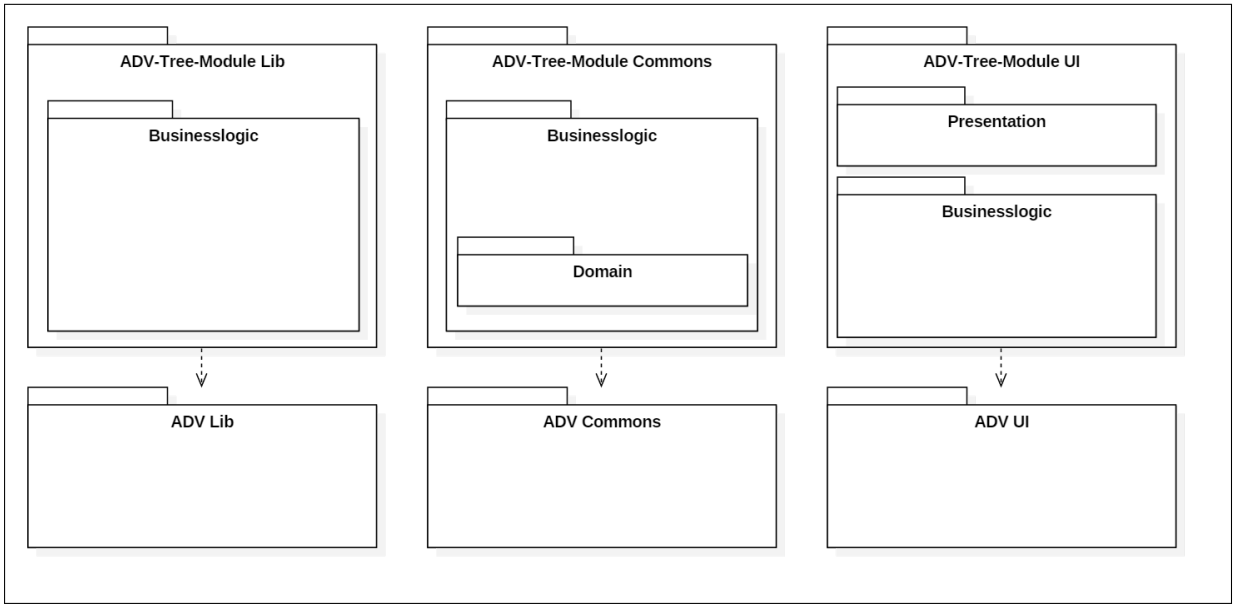


Abbildung 4.2.: Übersicht Schichtenmodell

Im Kontext der Strukturierung des ADV-Tree-Modules, konfrontiert mit der Anforderung die Modularität zu garantieren, wird eine Aufteilung in ein bis zwei Schichten je nach Container verwendet.
Verworfen wird die Idee, die Domain-Unterschicht im Commons Container als eigene Schicht zu definieren. Da die Klassen aus der Domain-Unterschicht im UI Container in der Businesslogic-Schicht und der Presentation-Schicht verwendet werden, würde dies für die Presentation-Schicht das Überspringen der Businesslogic-Schicht bedeuten. Im ADV-Tree-Module wird eine strikte Schichtung verwendet, die ein solches Überspringen verbietet.
Durch die Schichtung wird eine bessere Wartbarkeit, Austauschbarkeit und Verständlichkeit mit der Konsequenz eines grösseren Entwicklungsaufwandes und eventuellen Performance-Einbussen erreicht.
Eine genauere Darstellung der Schichten mit den wichtigsten Klassen befindet sich im Anhang unter dem Titel «Klassendiagramme». Dieser Anhang zeigt die umgesetzte Architektur.

4.3.1. ADV-Tree-Module Lib

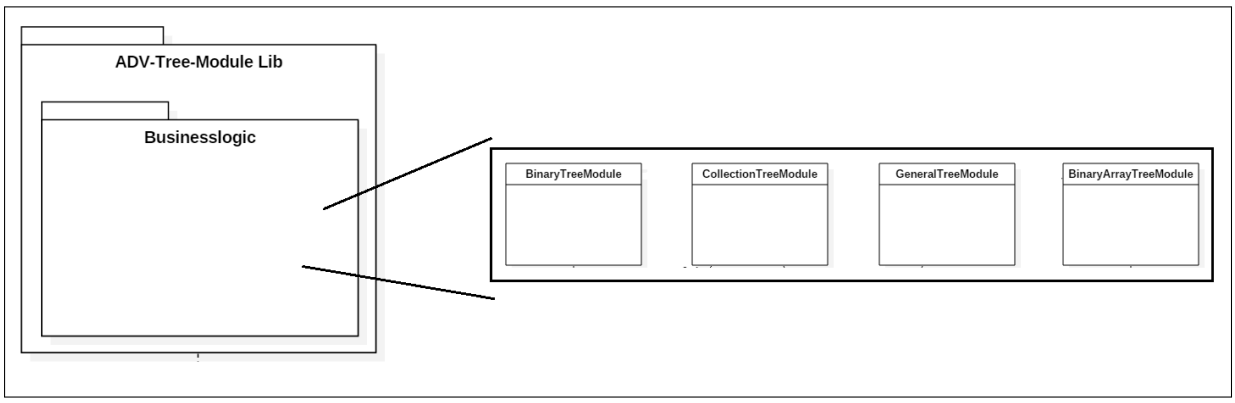


Abbildung 4.3.: Übersicht ADV-Tree-Module Lib

Der Lib Container enthält die verschiedenen TreeModules, welche im Domain-Modell definiert wurden. Das ADV-Tree-Module verwendet vier verschiedene Module. Verworfen wird die Idee, das BinaryTreeModule und das BinaryArrayTreeModule zu einem Modul zusammenzufassen, da es zu Verwirrung bei der Entwicklung des Gerüstcodes führen würde. Die beiden Module arbeiten zwar beide mit Binary Trees, aber das BinaryTreeModule bietet in seiner Funktionalität das Komplement des BinaryArrayTreeModule. Nähere Informationen zu diesen zwei Modulen sind im Kapitel «Wichtige Komponenten» zu finden.
Durch die Aufteilung in vier Module ist jedes Modul für genau eine Funktionalität verantwortlich (Single Responsibility Prinzip). Dadurch gewinnt das ADV-Tree-Module an Wartbarkeit und Verständlichkeit. Zudem ermöglicht dies ein unabhängiges Testen der verschiedenen Funktionalitäten, wie es in den Qualitätsmassnahmen gefordert wird.

Abhängigkeit zu Array-Modul

Damit die Array-Darstellung in den Modulen BinaryTreeModule und BinaryArrayTreeModule nicht implementiert werden muss, wird das bereits vorhandene Array-Modul verwendet. Vor der Übertragung an den Server wird das Array-Modul als Kind-Modul in die ModuleGroup eingefügt, sodass das User-Interface die Module korrekt erkennt und nebeneinander visualisiert.
Durch das Verwenden des Array-Moduls entsteht zwar unter den Modulen eine Abhängigkeit, jedoch kann so duplizierter Code vermieden werden.

Erweiterung um Kind-Module

Die Anforderungsspezifikation beinhaltet eine Anforderung, die verlangt, dass dem ADV-Tree-Module bestehende Module hinzugefügt werden können, um Einsicht in die Datenstruktur dieser Module zu ermöglichen. Das ADV-Tree-Module setzt dies um, indem jedes TreeModule die Möglichkeit bietet Kind-Module hinzuzufügen. Dies erlaubt es dem Studenten Datenstrukturen für zukünftige Algorithmen ohne Anpassung des ADV-Tree-Modules hinzuzufügen.

4.3.2. ADV-Tree-Module UI

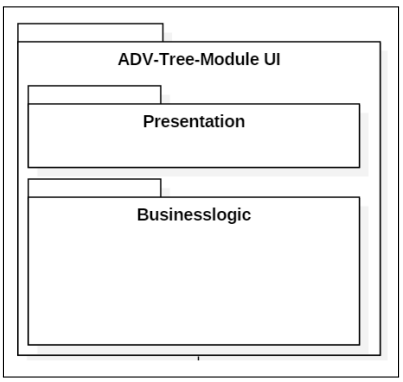


Abbildung 4.4.: Übersicht ADV-Tree-Module UI

Der UI Container ist für das Darstellen der Domain-Elemente aus dem Commons Container verantwortlich.

4.3.3. ADV-Tree-Module Commons

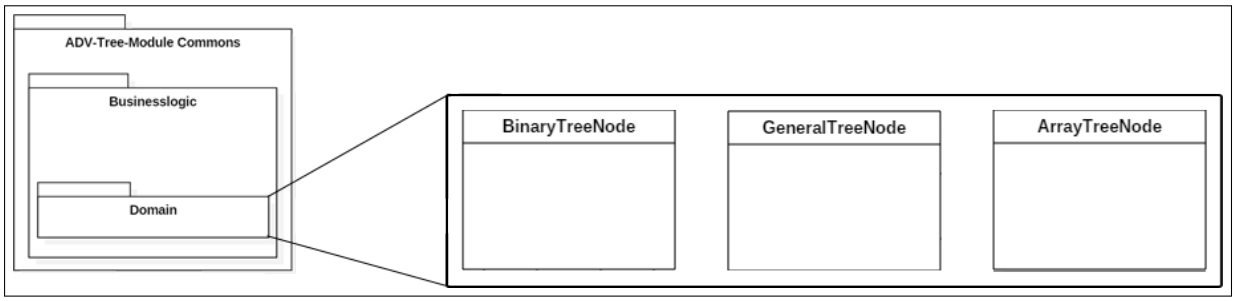


Abbildung 4.5.: Übersicht ADV-Tree-Module Commons

Der Commons Container beinhaltet diejenigen Komponenten, welche sowohl vom ADV Lib, als auch vom ADV UI benötigt werden. Dazu gehören zum einen sämtliche Konstanten des ADV-Tree-Modules und zum anderen die Domain-Objekte. Im Falle des ADV-Tree-Modules sind dies die Tree-Nodes aus dem Domain-Modell.

Übermittlung Client – Server

Durch den ADV [5] sind die Elemente zur Übermittlung (ADVElement und ADVRelation) vorgegeben. Diese sind sehr allgemein gehalten, um Module zu ermöglichen. Im Fall der Trees des ADV-Tree-Modules verlieren die Knoten dabei die Information über die Kindknoten. Aus diesem Grund implementiert das ADV-Tree-Module im Commons Container eine Komponente für die Relationen. Damit können die Parent-Child-Beziehungen auf den Server übertragen und dort zurückgewonnen werden.

Verschiebung der ArrayTreeNode nach ADV Lib

Im Verlauf der Studienarbeit hat sich gezeigt, dass die ArrayTreeNode im ADV UI nicht benötigt wird. Aus diesem Grund ist entschieden worden, dass sie aus dem ADV Commons ins ADV Lib verschoben wird.

4.4. Wichtige Komponenten

Dieses Kapitel beschreibt wichtige Komponenten des ADV-Tree-Modules, deren Zweck nicht ganz klar ist.

4.4.1. BinaryTreeModule

Das BinaryTreeModule ist darauf ausgelegt, den Root-Knoten eines Binary-Trees zu erhalten. Daraus erkennt es den Binary-Tree und berechnet das zugehörige Array.

4.4.2. BinaryArrayTreeModule

Das BinaryArrayTreeModule ist das Gegenstück zum BinaryTreeModule. Es nimmt ein Array von Elementen entgegen und wandelt diese in ArrayTreeNode um.

4.4.3. ArrayTreeNode

Das BinaryArrayTreeModule nimmt ein Array mit Elementen entgegen. Beim Berechnen des Baumes muss es aus diesen Elementen Knoten erstellen. ArrayTreeNode symbolisiert diese Knoten, weil sie keine Knoten mit Kindknoten im üblichen Sinn sind.

4.5. Umsetzung Qualitätsmassnahmen

Dieses Kapitel beschreibt sämtliche Qualitätsattribute aus den Anforderungsspezifikationen und zeigt, wie diese mit der gewählten Architektur umgesetzt werden.

4.5.1. Austauschbarkeit

Das ADV-Tree-Module wird in vier verschiedene Module (BinaryArrayTreeModule, BinaryTreeModule, CollectionTreeModule und GeneralTreeModule) unterteilt, welche jeweils ein Teilgebiet der geforderten Qualitätsanforderung abdecken.

4.5.2. Verständlichkeit Tree

Um die Verständlichkeit der Trees zu fördern, wird für diese die Notation aus den Studien-Modulen Algorithmen und Datenstrukturen 1 und 2 verwendet. Jede Node wird durch ein abgerundetes Rechteck dargestellt. Die Parent-Knoten werden durch Geraden mit ihren Kindern verbunden. Ein erster Entwurf ist aus dem Kapitel Wireframes der Anforderungsspezifikation zu entnehmen.

4.5.3. Verständlichkeit Array-Tree

Für die Visualisierung von Binary Trees mit dem zugehörigen Array wird die Variante mit Kind-Modulen gewählt. Dies ermöglicht das Verwenden des bereits implementierten Array-Moduls. Jedoch können so keine direkten Verbindungen zwischen den Array-Elementen und den Tree-Nodes gemacht werden. Um die Verständlichkeit dennoch sicherzustellen, werden neben den Nodes im Tree die Indexe der entsprechenden Array-Einträge dargestellt. Ein entsprechender Entwurf ist im Kapitel Wireframes der Anforderungsspezifikation vorhanden.

4.5.4. Modularität

Durch die Eingliederung in die bestehenden ADV Container (UI, Lib und Commons) ist das ADV-Tree-Module als Modul integriert und erfüllt somit diese Qualitätsanforderung.

4.5.5. Erlernbarkeit

Gemäss den Qualitätsanforderungen der Anforderungsspezifikation wird ein Benutzerhandbuch mit Code-Beispielen erstellt.

4.5.6. Testbarkeit

Durch die Aufteilung des ADV-Tree-Modules in vier verschiedene Module sind diese unabhängig voneinander testbar.

4.5.7. Nachvollziehbarkeit

Im ADV-Tree-Module wird wie beim ADV SLF4J mit Logback benutzt. Dies bietet verschiedene Log-Levels an (2. Dimension) und ermöglicht die Einstellung des Log-Levels auf Package-Ebene (1. Dimension) per XML-Datei oder zur Laufzeit per JConsole.

4.6. Umstellung Java 11

Der ADV wurde ursprünglich mit Java 9 entwickelt, allerdings bietet Oracle mittlerweile kein Java 9 mehr an. Entweder muss auf die Version 8 oder 11 umgestiegen werden. Java 8 kommt für das ADV-Tree-Module nicht infrage, da der ADV potentiell neue Funktionen beinhaltet. Aus diesem Grund wird Java 11 eingesetzt. Die Umstellung auf Java 11 zieht einige Konsequenzen mit sich. Bei allen Komponenten im CI/CD Prozess muss, wo angegeben, die Version auf 11 erhöht werden. In den einzelnen Projekten (UI, Lib und Commons) müssen veraltete oder nicht mehr funktionierende Bibliotheken ausgetauscht werden.
Mit Java 11 hat Oracle neue Lizenzbestimmungen eingeführt. Bisher konnte für alle Projekte das Oracle-JDK verwendet werden. Nun muss für Open-Source Projekte auf das Open-JDK [1] gewechselt werden. Da es sich beim ADV um ein Open-Source-Projekt handelt, ist er von dieser Umstellung betroffen.
Mit Java 11 ist JavaFX nicht mehr im JDK inbegriffen, sondern als separate Bibliothek bereitgestellt. Diese Bibliothek ist allerdings plattformabhängig. Zur Lösung dieses Problems gibt es zwei Möglichkeiten. Zum einen kann für jede Plattform eine separate Jar-Datei zur Verfügung gestellt werden. Dieses beinhaltet jeweils die entsprechende JavaFX-Bibliothek. Eine andere Möglichkeit ist das Erstellen einer einzelnen Jar-Datei, welches die Dependencies sämtlicher Plattformen beinhaltet.
Für das ADV-UI-Jar wird die Variante mit einem grossen Jar-File gewählt. Durch diese Entscheidung verdoppelt sich die Grösse des ADV-UI-Jars. Jedoch erleichtert sich dadurch das Bereitstellen einer User-Codebase, da das Jar-File direkt eingebunden werden kann und später nicht für die jeweiligen Plattformen angepasst werden muss. Zusätzlich vereinfacht dies die Entwicklung des ADV UI, da für das Deployment nur ein Jar bereitgestellt werden muss.
Eine weitere Konsequenz der Umstellung ist, dass die Benutzer des ADV mindestens Java 11 auf ihrem Rechner installiert haben müssen. Da JavaFX im ADV als separate Library hinzugefügt wird, kann das ADV UI nicht mit älteren Java-Versionen ausgeführt werden.

4.7. Umsetzung Fixieren von Tree-Nodes

Für die Umsetzung der Anforderung «Fixieren von Tree-Nodes» stehen zwei Varianten zur Verfügung. Bei der ersten Variante müssen im ADV UI die einzelnen Snapshots an einem Ort gespeichert werden, wo der Layouter auf sie zugreifen kann. So könnte er alle Snapshots miteinander vergleichen und die Nodes an der korrekten Stelle positionieren.
Die zweite Variante sieht eine Änderung an der Klasse ModuleGroup vor. Diese wird um das Feld `metaData` erweitert, über welches zusätzliche Daten zwischen dem ADV Lib und dem ADV UI ausgetauscht werden können. Der Benutzer muss in der Code-Base die maximale linke und rechte Tree-Höhe angeben. Diese Informationen werden über die Meta-Daten an das ADV UI übergeben, wo der Tree entsprechend dargestellt wird.
Für das ADV UI wird die zweite Variante gewählt, welche eine Erweiterung der ModuleGroup vorsieht. Dieser Weg ist einfacher zu implementieren, da weniger Änderungen am ADV-Framework gemacht werden müssen. Jedoch muss der Benutzer hier selbst die gewünschte Tree-Höhe übergeben, da es bei dieser Variante nicht möglich ist, diese Daten im ADV UI automatisch zu ermitteln.

4.8. Modul Positionierung

Zur Realisierung der Anforderung «Darstellung von Array unterhalb von Binary Tree» stehen zwei Optionen zur Verfügung. Eine Variante ist das Setzen einer Position für die Kind-Module im ADV Lib, sodass sie im UI an der entsprechenden Stelle dargestellt werden. Bei der anderen Variante wird die Darstellung des Arrays vom Tree-Modul übernommen, wobei die Position durch den Entwickler fest gesetzt wird.
Zur Umsetzung wird die Variante mit dem Setzen der Kind-Modul-Position gewählt. Dies erfordert eine Veränderung in allen drei ADV-Projekten, kann im Gegensatz zur anderen Variante aber auch in den übrigen Modulen verwendet werden. Im ADV Commons muss die ModuleGroup Klasse um ein Feld mit der Position des Moduls erweitert werden, damit die Position vom ADV Lib an das ADV UI übertragen werden kann. Im ADV UI muss die Komponente zur Darstellung aller Module einer Session überarbeitet werden, um die Positionen zu unterstützen. Im ADV Lib muss die Position vom Modul auf die ModuleGroup übertragen werden.
In der Klassenansicht des Enterprise Architects sind nur die Änderungen im Tree-Module und Array-Module dargestellt, da die Änderungen in den anderen Modulen analog dazu sind und somit keine zusätzlichen Informationen liefern.

4.9. BinaryArrayTreeEnhancedModule

Das BinaryArrayTreeEnhancedModule bietet zusätzliche Komfort-Funktionalität zum BinaryArrayTreeModule und soll nur für den Dozenten zur Verfügung stehen. Für das Verständnis der Trees, ist das Kennen des Zusammenhangs zwischen Binary Tree und unterliegendem Array wichtig. Durch das Verwenden der neuen Methoden im BinaryArrayTreeEnhancedModule geht dieser Lerneffekt jedoch verloren. Aus diesem Grund soll das BinaryArrayTreeEnhancedModule nicht durch Studenten verwendet werden und wird daher nicht im Benutzerhandbuch erwähnt.

4.10. Darstellung im Enterprise Architect

Die Übersicht des Enterprise Architects für das ADV-Tree-Module zeigt alle Klassen, welche mit dem neuen Modul in den Projekten ADV Commons, ADV Lib und ADV UI hinzugefügt werden. Zusätzlich zeigt sie auch Klassen des ADV-Cores, welche für die benötigte Funktionalität verändert werden müssen. Der Grund für diesen Entscheid ist, dass nachfolgende Entwickler das ADV-Tree-Module schnell weiterentwickeln oder verbessern können. Es gibt Ausnahmen für die Darstellung von Klassen aus dem Kern des ADV. Diese sind in den anderen Kapiteln in diesem Dokument beschrieben.
Beim Synchronisieren des ADV-Lib-Modells im Enterprise Architect mit dem Source Code wird die Fehlermeldung «DAO.Field [3163]» angezeigt. Nach dem Bestätigen der Meldung mit «OK» wird die Synchronisation ohne Probleme fortgesetzt. Der Fehler tritt auf, weil das Feld `nodeInformationList` und dessen Typdeklaration in der Klasse `CollectionTreeBuilder` eine von Enterprise Architect definierte Maximallänge überschreitet.
Ein weiteres Problem beim Synchronisieren des ADV-Lib-Modells tritt beim Vergleichen mit einem vorherigen Backup auf. Nach dem Durchführen des Reverse Engineerings erhöht sich die Anzahl Records der Zeile «t_xref» um 3 gegenüber dem Modell vor der Synchronisation.

Teil III

Anhang

ANHANG A

Artefakt Übersicht

A.1. Übersicht

Phase	Inception	Elaboration				Construction								Transition
Iteration	1	2		3		4		5		6		7		8
Woche	1	2	3	4	5	6	7	8	9	10	11	12	13	14
Datum	17.09	24.09	01.10	08.10	15.10	22.10	29.10	05.11	12.11	19.11	26.11	03.12	10.12	17.12
Meilensteine	M0	M1		M2		M3				M4		M5		M6
Artefakte														
Besprechungsprotokolle	x	x	x	x	x	x	x	x	x	x	x	x	x	x
Artefakt-Übersicht	v0.1	v0.2			v0.3				v0.4		v0.5			v1.0
Wireframes			v0.1	v0.2										v1.0
Projektplan		v0.1	v0.2											v1.0
Anforderungsspezifikation			v0.1	v0.2	v0.3				v0.4		v0.5			v1.0
Architektur- und Designspezifikation				v0.1	v0.2				v0.3		v0.4			v1.0
Prototyp					v0.1									
Systemtests Definitionen										v0.1			v0.2	v1.0
Usabilitytests Definitionen										v0.1				v1.0
Systemtests Protokolle											v0.1		v0.2	v1.0
Usabilitytests Protokolle											v0.1			v1.0
ADV Release									v1.1			v1.2		v2.0
ADV-Tree-Module									v0.1			v0.2		v1.0
Abstract													v0.1	v1.0
Management Summary														v1.0
Technischer Bericht														v1.0
Benutzerhandbuch														v1.0
Metriken														v1.0
Zeitauswertung														v1.0
Glossar und Abkürzungsverzeichnis														v1.0
Selbstständigkeitserklärung														v1.0
Persönliche Berichte														v1.0
Poster														v1.0

Meilensteine

- M0 Vision
- M1 Projektplan und Anforderungsspezifikation
- M2 Architektur- und Designspezifikation und Prototyp
- M3 Release 1
- M4 Usability Tests
- M5 Release 2
- M6 Dokumentation

A.2. Versionen

Artefakt-Übersicht	**Wireframes**	**Projektplan**
Übersicht über die Wochen und die für dann geplanten Artefakte	Zeichnerischer Entwurf der Darstellung der Bäume im ADV	Dokumentation des Managements und der Organisation während der Studienarbeit
v0.1 - Artefakte - Meilensteine - Iterationen - Phasen	v0.1 Erster Entwurf	v0.1 - Projektübersicht - Iterationsplanung - Projektorganisation - Meilensteine - Risikomanagement - Qualitätsmassnahmen
v0.2 Genaue Versionsplanung	v0.2 Verbesserungen nach Rücksprache mit Betreuer	v0.2 Verbesserungen nach Rücksprache mit Betreuer
v0.3 Inkrementelle Überarbeitung	v1.0 Endgültige Version	v1.0 Endgültige Version
v0.4 Inkrementelle Überarbeitung		
v0.5 Inkrementelle Überarbeitung		
v1.0 Endgültige Version		
Anforderungsspezifikation	**Architektur- und Designspezifikation**	**Prototyp**
Zusammenstellung aller Anforderungen an die ADV-Tree-Module	Zusammenstellung der Architektur, Architekturentscheidungen und des schlussendlichen Designs	Implementierung der Architektur und erster einfacher Funktionalitäten
v0.1 - Brief Use-Cases - Qualitätsanforderungen - Aktorenbeschreibung - Stakeholder	v0.1 Vorläufige Architektur und Architekturentscheidungen	v0.1 Prototyp Node-Tree-Modul
v0.2 Verbesserungen nach Rücksprache mit Betreuer	v0.2 Designentscheidungen	
v0.3 - Fully dressed Use-Cases (nach Bedarf)	v0.3 Eventuelle Nachbesserung der Architektur	
v0.4 Erfassung neuer Anforderungen	v0.4 Nachbesserung der Architektur	
v0.5 Erfassung neuer Anforderungen	v1.0 Endgültige Version	
v1.0 Endgültige Version		
Systemtests Definitionen	**Usabilitytests Definitionen**	**Systemtests Protokolle**
Auflistung nicht automatisierbarer Testszenarien	Definition der Aufgabenstellungen für Testpersonen	Auswertungen der Systemtest-Durchführungen
v0.1 Erfassen der Systemtests anhand Implementationsstand	v0.1 Erstellung der Usability-Testszenarien	v0.1 Ausführung der ersten Systemtests
v0.2 Erfassen aller Systemtests	v1.0 Endgültige Version	v0.2 Ausführung sämtlicher Systemtests
v1.0 Endgültige Version		v1.0 Endgültige Version
Usabilitytests Protokolle	**ADV Release**	**ADV-Tree-Module**
Auswertungen der Usability-Test-Durchführungen	Bereitstellung der JAR-Files für ADV	Implementation der Darstellung allgemeiner Trees, Binary Trees mit und ohne Array
v0.1 Durchführung der Usability-Tests	v1.1 Release mit erstem Node-Tree-Modul	v0.1 Erste Implementation für Usability-Tests
v1.0 Endgültige Version	v1.2 Release mit Verbesserungen aus Usability-Test und optionalen Anforderungen	v0.2 Verbesserung anhand Usability-Tests und optionale Anforderungen
	v2.0 Endgültiger Release	v1.0 Endgültige Version
Besprechungsprotokolle	**Abstract**	**Management Summary**
Dokumentation der wichtigsten Entscheidungen aus den Besprechungen	Knappe Zusammenfassung der Ergebnisse und Resultate der Studienarbeit	Zusammenfassung für Vorgesetzten des Vorgesetzten nach vorgegebener Gliederung
	v0.1 Erste Version zum Gegenlesen	v1.0 Endgültige Version
	v1.0 Endgültige Version	
Technischer Bericht	**Benutzerhandbuch**	**Metriken**
Kurzer Bericht zur Studienarbeit mit Übersicht, Ergebnissen, Schlussfolgerungen, Glossar und Literaturverzeichnis	Anleitung zur Benutzung der Tree-Module mit Beispielen und zur Verwendung der Entwicklungswerkzeuge	Auswertung der Projektqualität anhand der Metriken
v1.0 Endgültige Version	v1.0 Endgültige Version	v1.0 Endgültige Version
Zeitauswertung	**Glossar und Abkürzungsverzeichnis**	**Selbstständigkeitserklärung**
Auswertung des Zeitaufwands mit Soll/Ist-Vergleich	Auflistung und Beschreibung sämtlicher verwendeter Abkürzungen und Begriffe	Unterschriebene Erklärung zur Eigenständigkeit der Studienarbeit
v1.0 Endgültige Version	v1.0 Endgültige Version	v1.0 Endgültige Version
Poster	**Persönliche Berichte**	
Übersicht des Projekts auf einem Plakat	Persönliche Reflexion zur Studienarbeit	
v1.0 Endgültige Version	v1.0 Endgültige Version	

ANHANG B

Technische Risiken

Risikomanagement

Projekt: ADV Tree Module
Erstellt am: 24.09.2018
Autor: Fabian Meier, Jan Winter
Gewichteter Schaden: 3.2

Nr	Titel	Beschreibung	max. Schaden [h]	Eintrittswahrscheinlichkeit	Gewichteter Schaden	Vorbeugung	Verhalten beim Eintreten
R1	Inkompatibilität Java 10	Durch das Aktualisieren des ADV auf Java 10 treten beim Deployment Fehler auf	6	30%	1.8	Möglichst früh ein Deployment machen, um mögliche Fehler schnellstmöglich zu beseitigen. Orte erkennen, bei denen bisher Java 9 verwendet wurden.	Versionen der Libraries und Plugins anpassen. Schrittweises Abarbeiten der Fehlermeldungen.
R2	Ausfall des Redmine-Servers	Während der Studienarbeit fällt der Redmine-Server aus	4	5%	0.2	Regelmässige Backups und speichern in GitHub-Repository	Rücksprache mit Server-Verantwortlichem und Wiederherstellung von Redmine mit Backup
R3	Modul-Abweichung	Das zu implementierende ADV-Tree-Modul unterscheidet sich stark von den Beispiel-Modulen, was eine Implementierung erschwert	8	10%	0.8	Frühes Erstellen eines Prototyps	Rücksprache mit den ADV-Entwicklern
R4	Zu geringe Abstraktion des ADV	Das ADV-Tool ist zu wenig abstrakt, was eine Implementierung des ADV-Tree-Moduls erschwert	8	5%	0.4	Frühes Erstellen eines Prototyps	Spezifischere Implementationen des ADV-Tree-Moduls (Verlust der Generalität)
Summe			**26**		**3.2**		

ANHANG C

Benutzerhandbuch

Änderungsgeschichte

Datum	Version	Änderung	Autor
12.11.2018	1.0	Dokument erstellt	Team
26.11.2018	1.0	Fixed-Nodes und addRoot Funktionalität hinzugefügt	Team
07.12.2018	1.0	Module Positioning hinzugefügt	Team

Tabelle C.1.: Änderungsgeschichte

C.1. Einführung

Dieses Dokument bietet eine Erweiterung des bereits vorhandenen ADV-Benutzerhandbuches [5]. Es ist als Teil des Kapitels «Modules» im ADV-Benutzerhandbuch anzusehen.
Das Handbuch dient dem Benutzer als Hilfe zur Visualisierung von selbst implementierten Trees im ADV. Für ein einfacheres Verständnis wird diese Hilfe mit Code-Beispielen und Screenshots ergänzt.
Für den ADV wurden für die unterschiedlichen Tree-Typen jeweils eigens dafür angepasste Module entwickelt. Aus diesem Grund enthält dieses Benutzerhandbuch für jedes dieser Tree-Module eine eigene Anleitung.

C.2. Installationsanleitung

Das ADV-Tree-Module wird mit dem ADV mitinstalliert. Aus diesem Grund kann die Installationsanleitung des ADV [5] konsultiert werden.

C.3. Positionierung von Modulen

Bei Modulen ist es möglich eine Position festzulegen. Diese Position wird verwendet, wenn das Modul als Kind-Modul einem anderen Modul hinzugefügt wird. Das Modul wird dann im UI an der jeweiligen Position relativ zum Hauptmodul dargestellt. Das Modul kann entweder links, rechts, oben oder unten positioniert werden. Dies ist für alle Module möglich, nicht nur für die folgenden Tree-Module. Das Benutzerhandbuch beschreibt, wie die Positionierung mit den Tree-Modulen funktioniert. Die Verwendung in den anderen Modulen ist identisch.

C.4. General Tree Module

Das General Tree Module dient der Darstellung von allgemeinen Trees. Dazu gehören sämtliche Trees, deren Nodes mehr als zwei Kinder haben können.

C.4.1. Übersicht

GeneralTreeModule

Das GeneralTreeModule kapselt die Daten für einen generellen Tree. An das Modul muss nur die Root des Trees übergeben werden. Das Modul sucht darauf eigenständig alle Kinder, Enkelkinder, etc. der Wurzel und kann damit den Tree entsprechend darstellen.

`GeneralTreeModule( String sessionName)`	Dem Konstruktor muss ein Session-Name übergeben werden. Dieser dient als Name für die Ansicht im UI.
`GeneralTreeModule( ADVGeneralTreeNode<T> root, String sessionName)`	Zusätzlich kann im Konstruktor auch die Root des Trees übergeben werden.
`void setRoot( ADVGeneralTreeNode<T> newRoot)`	Setzen einer neuen Root.
`void setPosition( ModulePosition position)`	Setzen der Modul-Position im UI. Mögliche Werte sind: DEFAULT, LEFT, TOP, RIGHT, BOTTOM

Tabelle C.2.: Methoden GeneralTreeModule

Interface ADVGeneralTreeNode<T>

Alle Nodes eines allgemeinen Trees, welche im ADV dargestellt werden sollen, müssen das Interface ADVGeneralTreeNode<T> implementieren.

`ADVStyle getStyle()`	Gibt den Style der Node zurück, mit dem die Node im UI dargestellt werden soll.
`T getContent()`	Gibt den Inhalt der Node zurück, welcher in der Visualisierung dargestellt werden soll.
`List<? extends ADVGeneralTreeNode<T>> getChildren()`	Gibt alle Kinder der Node als Liste zurück.

Tabelle C.3.: Methoden ADVGeneralTreeNode<T>

C.4.2. Beispiel

Dieses Beispiel zeigt zuerst einen einfachen allgemeinen Tree. In einem zweiten Schritt wird ein Knoten aus dem Tree entfernt, worauf er erneut visualisiert wird.

```
public class SimpleGeneralTreeNode implements ADVGeneralTreeNode<String> {

   private List<SimpleGeneralTreeNode> children;
   private ADVStyle style;
   private String content;

   public SimpleGeneralTreeNode(String content) {
      this.content = content;
      children = new ArrayList<>();
   }

   /* Getters, Setters, addChild and removeChild */
}
```

Auflistung C.1: Beispiel-Implementation ADVGeneralTreeNode<T>

```
public class SimpleGeneralTreeProgram {

   public static void main(String[] args) throws ADVException {
      // connect to (and optionally start) the UI
      ADV.launch(null);

      // create nodes and set relations
      SimpleGeneralTreeNode root = new SimpleGeneralTreeNode("ROOT");

      SimpleGeneralTreeNode nodeA = new SimpleGeneralTreeNode("A");
      root.addChild(nodeA);

      SimpleGeneralTreeNode nodeB = new SimpleGeneralTreeNode("B");
      nodeA.addChild(nodeB);
      SimpleGeneralTreeNode nodeC = new SimpleGeneralTreeNode("C");
      nodeA.addChild(nodeC);
      SimpleGeneralTreeNode nodeD = new SimpleGeneralTreeNode("D");
      nodeA.addChild(nodeD);

      GeneralTreeModule module =
            new GeneralTreeModule(root, "Simple General Tree");

      // send current state of graph to the UI to be displayed
      ADV.snapshot(module);

      // remove child from nodeA
      nodeA.removeChild(nodeC);
      ADV.snapshot(module);
   }

}
```

Auflistung C.2: Beispiel-Darstellung eines General Trees

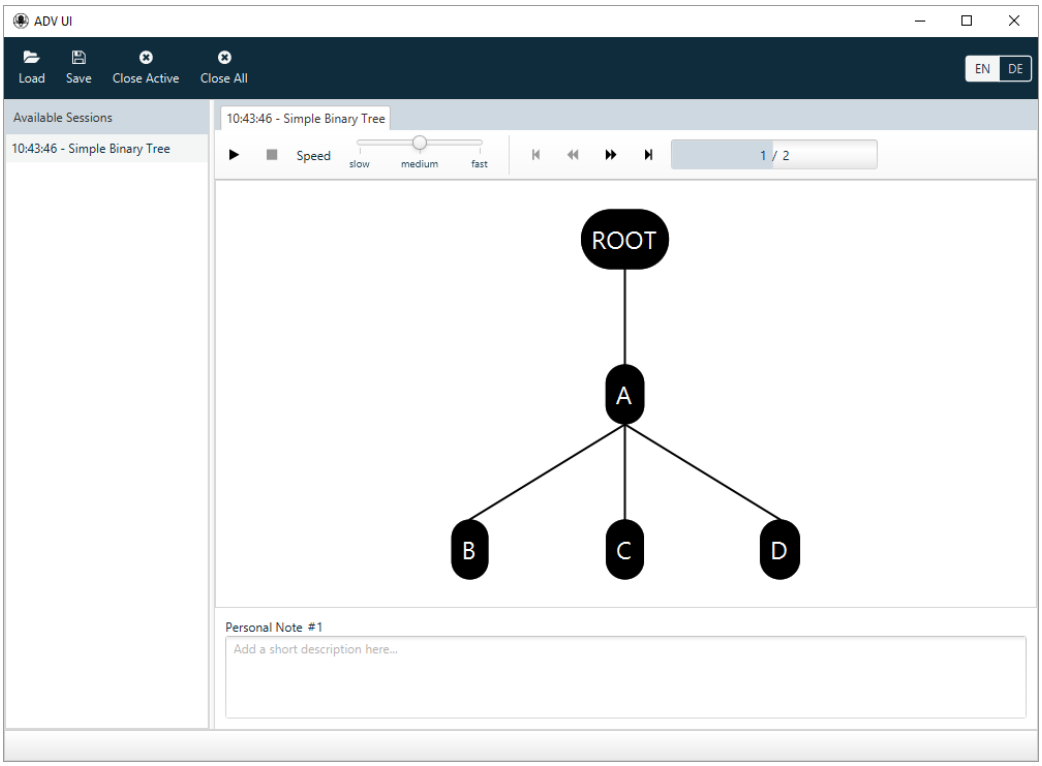


Abbildung C.1.: Beispiel-Darstellung eines General Trees

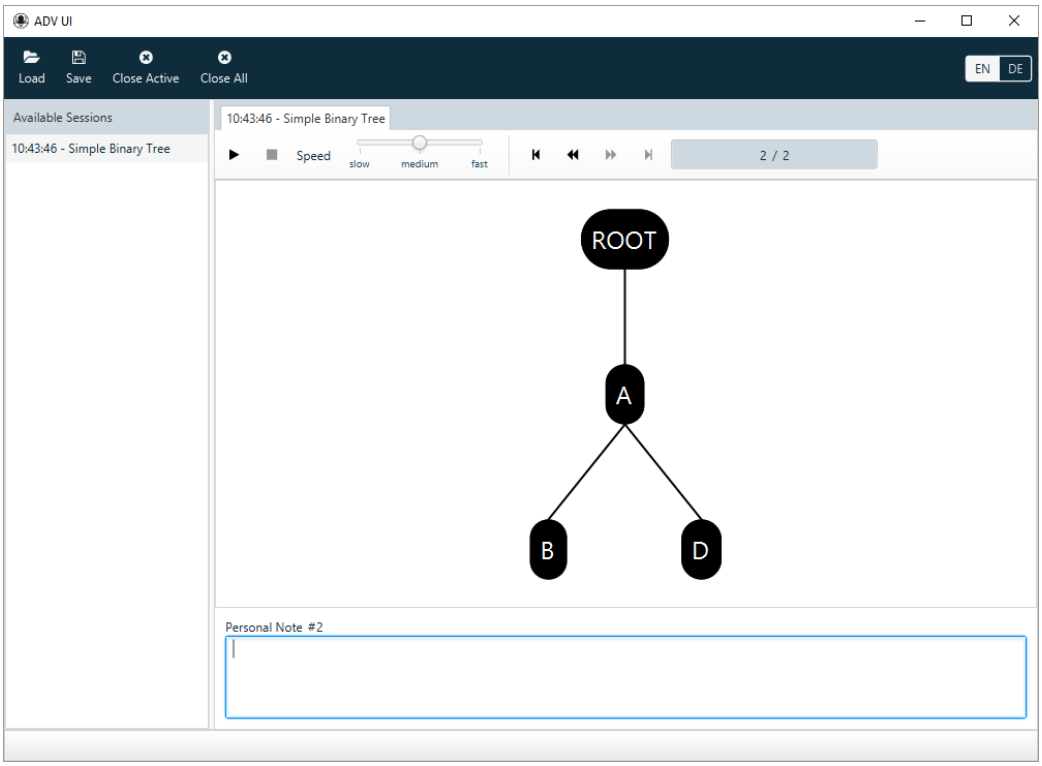


Abbildung C.2.: Beispiel-Darstellung eines General Trees nach dem Entfernen einer Node

C.5. Binary Tree Module

Das Binary Tree Module dient der Darstellung von Binary Trees. Dies sind Trees, deren Knoten höchstens ein linkes und ein rechtes Kind haben.
Eine Zusatzfunktion des Binary Tree Moduls ist das Darstellen eines Arrays zum Binary Tree. Diese zusätzliche Visualisierung zeigt, wie der Binary Tree in einem Array gespeichert werden kann. Es wird empfohlen, das Array nur bei kleinen Trees darstellen zu lassen, da das Array bei grossen Trees unlesbar wird.

C.5.1. Übersicht

BinaryTreeModule

Das BinaryTreeModule kapselt die Daten für einen Binary Tree. Wie beim generellen Tree reicht auch hier das Übergeben der Root, mit der dann alle anderen Knoten des Trees gefunden und dargestellt werden können.
Das Modul bietet die Möglichkeit die Knoten im UI in einer fixierten Position darzustellen, was vor allem für die Darstellung von Rotationen nützlich sein kann. Um dies zu erreichen muss die Methode `setFixedTreeHeight` aufgerufen werden. Bei den Parametern muss angegeben werden, was die maximale Höhe des linken Teilbaums (Höhe ab dem linken Kind der Root + 1, da die Root auch dazugezählt wird) und die maximale Höhe des rechten Teilbaumes ist.

`BinaryTreeModule(` `  String sessionName)`	Dem Konstruktor muss ein Session-Name übergeben werden. Dieser dient als Name für die Ansicht im UI.

`BinaryTreeModule(`
`  ADVBinaryTreeNode<T> root,`
`  String sessionName)`
	Zusätzlich kann im Konstruktor auch die Root des Trees übergeben werden.

`void setRoot(` `  ADVBinaryTreeNode<T> newRoot)`	Setzen einer neuen Root.
`void setShowArray(` `  boolean showArray)`	Setzen, ob Array-Darstellung im UI angezeigt werden soll.

`void setFixedTreeHeight(`
`  in maxLeftHeight,`
`  int maxRightHeight)`
	Methode zum Setzen der maximalen linken Höhe und der maximalen rechten Höhe des Baumes, um die Nodes im UI fixieren zu können.

`void clearFixedTreeHeight()`	Methode, um die Werte für die maximale linke und rechte Seite zu löschen, damit im UI die Nodes nicht mehr fixiert werden.

`void setPosition(` `  ModulePosition position)`	Setzen der Modul-Position im UI. Mögliche Werte sind: DEFAULT, LEFT, TOP, RIGHT, BOTTOM

Tabelle C.4.: Methoden BinaryTreeModule

Interface ADVBinaryTreeNode<T>

Alle Nodes eines Binary Trees, welche im ADV dargestellt werden sollen, müssen das Interface ADVBinaryTreeNode<T> implementieren.

`ADVStyle getStyle()`	Gibt den Style der Node zurück, mit dem die Node im UI dargestellt werden soll.
`T getContent()`	Gibt den Inhalt der Node zurück, welcher in der Visualisierung dargestellt werden soll.
`ADVBinaryTreeNode<T>` `  getLeftChild()`	Gibt das Kind zurück, das links unter der Node dargestellt werden soll.
`ADVBinaryTreeNode<T>` `  getRightChild()`	Gibt das Kind zurück, das rechts unter der Node dargestellt werden soll.

Tabelle C.5.: Methoden ADVBinaryTreeNode<T>

C.5.2. Beispiel

Im folgenden Beispiel wird ein einfacher Binary Tree mit einer Root und zwei Kindern erstellt. Das rechte Kind wird dabei farblich hervorgehoben. Darauf wird zum Tree auch das Array visualisiert.

```
public class SimpleBinaryTreeNode implements ADVBinaryTreeNode<String> {

   private SimpleBinaryTreeNode leftChild;
   private SimpleBinaryTreeNode rightChild;
   private String content;
   private ADVStyle style;

   public SimpleBinaryTreeNode(String content) {
      this.content = content;
   }

   /* Getters and Setters */
}
```

Auflistung C.3: Beispiel-Implementation ADVBinaryTreeNode<T>

```
public class SimpleBinaryTreeProgram {

   public static void main(String[] args) throws ADVException {
      // connect to (and optionally start) the UI
      ADV.launch(null);

      BinaryTreeModule module = new BinaryTreeModule("Simple Binary Tree");

      // create nodes and set relations
      SimpleBinaryTreeNode root = new SimpleBinaryTreeNode("ROOT");

      SimpleBinaryTreeNode nodeA = new SimpleBinaryTreeNode("A");
      root.setLeftChild(nodeA);

      SimpleBinaryTreeNode nodeB = new SimpleBinaryTreeNode("B");
      nodeB.setStyle(new ADVSuccessStyle());
      root.setRightChild(nodeB);

      module.setRoot(root);

      // set the maximum heights that the nodes are fixed in the UI
      module.setFixedTreeHeight(1, 1);

      // send current state of graph to the UI to be displayed
      ADV.snapshot(module);

      // show array for binary tree
      module.setShowArray(true);
      ADV.snapshot(module);
   }

}
```

Auflistung C.4: Beispiel-Darstellung eines Binary Trees

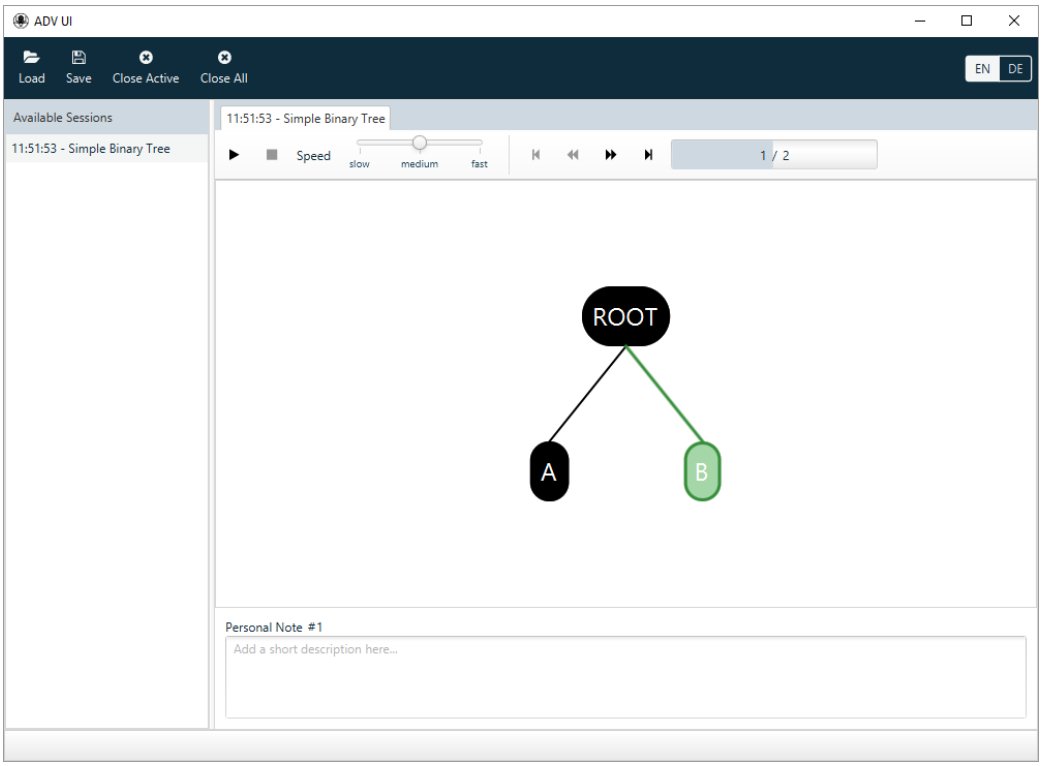


Abbildung C.3.: Beispiel-Darstellung eines Binary Trees

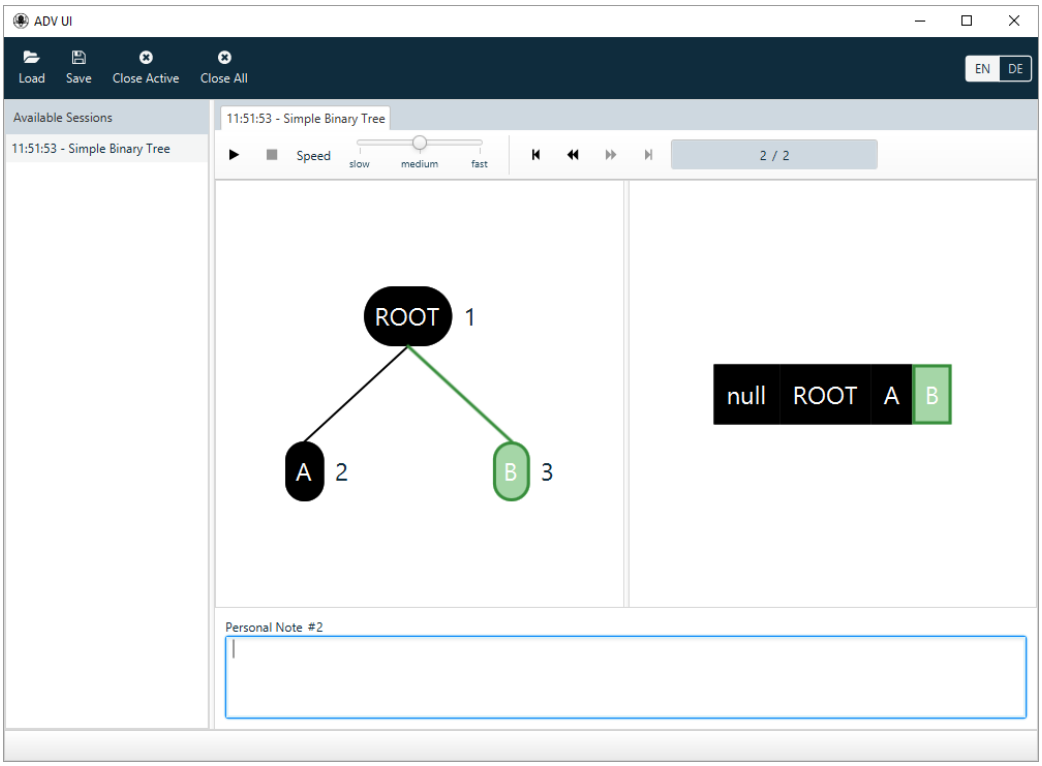


Abbildung C.4.: Darstellung des Binary Trees mit Array

C.6. Binary Array Tree Module

Das Binary Array Tree Module bildet das Gegenstück zum Binary Tree Module. Auch hier können nur Binary Trees dargestellt werden, jedoch wird anstatt einer Root ein Array bzw. eine ArrayList an das Modul übergeben.

C.6.1. Übersicht

BinaryArrayTreeModule<T>

Das BinaryArrayTreeModule<T> kapselt die Daten für einen Binary Tree. Für die Darstellung muss der Binary Tree in Array- bzw. ArrayList-Form an das Modul übergeben werden. Das Modul berechnet daraus die Beziehungen zwischen den Nodes und kann den Tree im UI entsprechend darstellen.
Damit die Trees im UI korrekt dargestellt werden können, muss die Länge des Arrays bzw. der ArrayList immer einen Wert einer zweier-Potenz betragen. Array-Einträge die keine Nodes beinhalten haben den Wert `null` .
Beim Übergeben eines Arrays bzw. einer ArrayList erstellt das Modul bei sich eine Kopie davon. Dies hat die Konsequenz, dass Änderungen am Array bzw. an der ArrayList nicht automatisch im Modul übernommen werden. Damit die Änderungen auch im UI sichtbar werden, muss vor dem Erstellen eines neuen Snapshots das aktuelle Array bzw. ArrayList erneut übergeben werden.
Wie beim Binary Tree Module kann auch beim Binary Array Tree Module das Array neben dem Baum dargestellt werden. Es wird empfohlen, das Array nur bei kleinen Trees darstellen zu lassen, da das Array bei grossen Trees unlesbar wird.
Das Modul bietet die Möglichkeit die Knoten im UI in einer fixierten Position darzustellen. Genauere Informationen lassen sich im Kapitel zum «BinaryTreeModule» finden.

`BinaryArrayTreeModule(`
`  T[] nodeArray,`
`  String sessionName)`
	Dem Konstruktor muss ein Array sowie ein Session-Name übergeben werden. Der Session-Name dient als Bezeichnung für die Ansicht im UI.

`BinaryArrayTreeModule(`
`  ArrayList<T> nodeList,`
`  String sessionName)`
	Alternativ kann statt dem Array eine ArrayList mit den Werten übergeben werden.

`void setArray(T[] nodeArray)`	Ersetzen des Arrays.
`void setArray(` `  ArrayList<T> nodeList)`	Ersetzen des Arrays durch Übergeben einer ArrayList.
`void setShowArray(` `  boolean showArray)`	Setzen, ob Array-Darstellung im UI angezeigt werden soll.

`void setFixedTreeHeight(`
`  in maxLeftHeight,`
`  int maxRightHeight)`
	Methode zum Setzen der maximalen linken Höhe und der maximalen rechten Höhe des Baumes, um die Nodes im UI fixieren zu können.

`void clearFixedTreeHeight()`	Methode, um die Werte für die maximale linke und rechte Seite zu löschen, damit im UI die Nodes nicht mehr fixiert werden.
`void setPosition(` `  ModulePosition position)`	Setzen der Modul-Position im UI. Mögliche Werte sind: DEFAULT, LEFT, TOP, RIGHT, BOTTOM

Tabelle C.6.: Methoden BinaryArrayTreeModule<T>

C.6.2. Beispiel

Der folgende Code visualisiert zuerst eine ArrayList eines Binary Trees im UI. Danach werden die Knoten C und F aus dem Tree entfernt.

```
public class SimpleBinaryArrayTreeProgram {

   public static void main(String[] args) throws ADVException {
      // connect to (and optionally start) the UI
      ADV.launch(null);

      ArrayList<String> binaryTreeArrayList = new ArrayList<>(
            Arrays.asList(null, "A", "B", "C", "D", null, "F", null));

      BinaryArrayTreeModule<String> module = new BinaryArrayTreeModule<>(
            binaryTreeArrayList, "Simple Binary Array Tree");

      // set the maximum heights that the nodes are fixed in the UI
      module.setFixedTreeHeight(2, 2);

      // send current state of graph to the UI to be displayed
      ADV.snapshot(module);

      // remove nodes C and F from tree
      binaryTreeArrayList.set(6, null);
      binaryTreeArrayList.set(3, null);
      module.setArray(binaryTreeArrayList);
      ADV.snapshot(module);
   }

}
```

Auflistung C.5: Beispiel-Darstellung eines Binary Array Trees

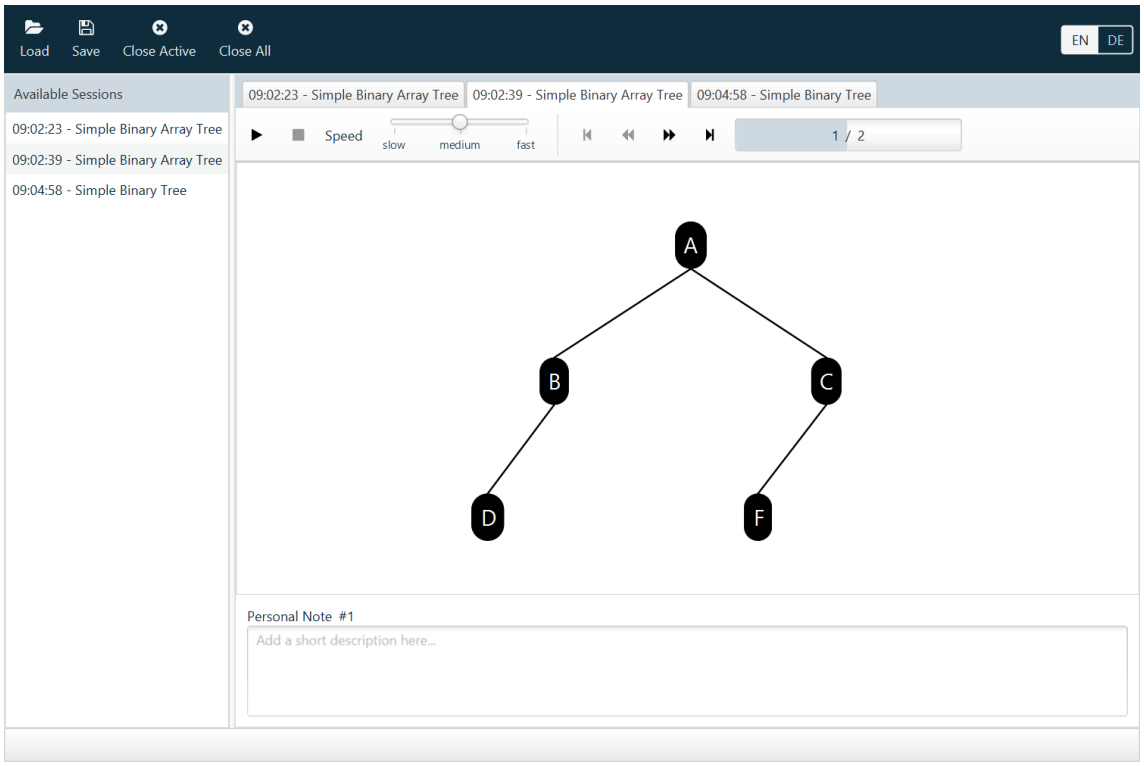


Abbildung C.5.: Beispiel-Darstellung eines Binary Array Trees

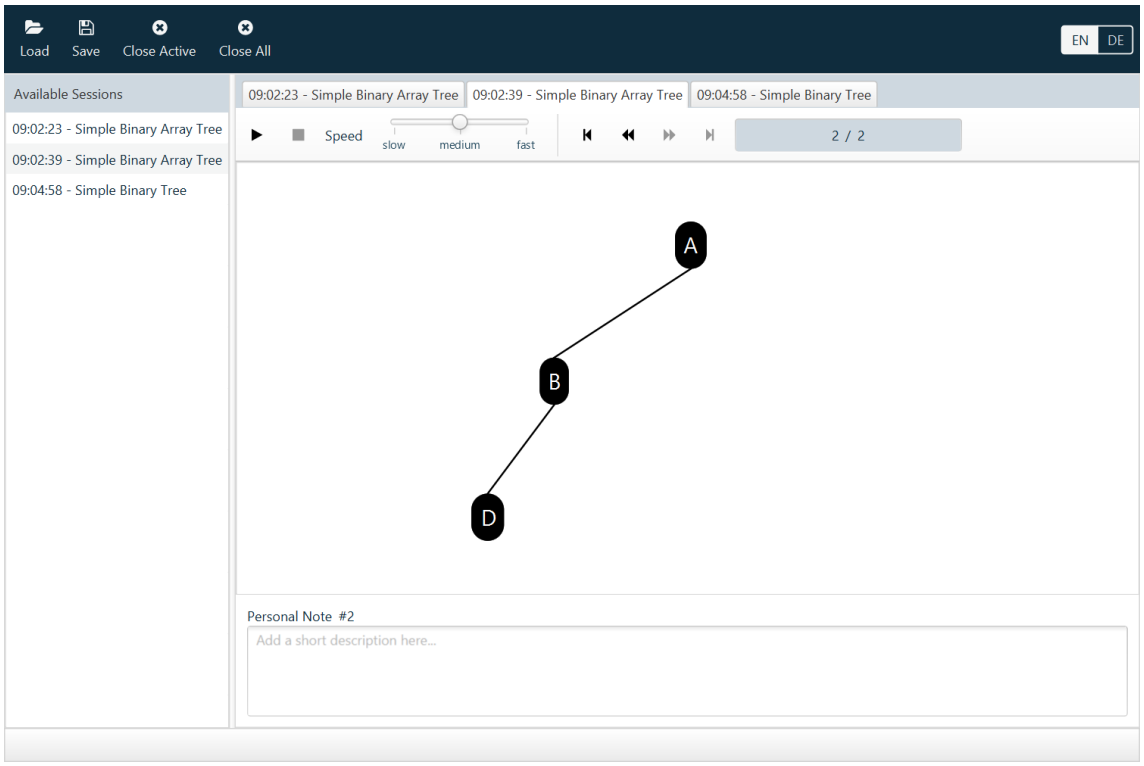


Abbildung C.6.: Darstellung des Trees nach dem Entfernen der Knoten C und F

C.7. Collection Tree Module

Das Collection Tree Module dient sowohl zur Darstellung von allgemeinen Trees, als auch von Forests. Dem Modul müssen jedoch immer alle Nodes hinzugefügt werden, die im UI dargestellt werden sollen.

C.7.1. Übersicht

CollectionTreeModule<T>

Das CollectionTreeModule<T> kapselt die Daten für einen Collection Tree. Alle Nodes, die im UI dargestellt werden sollen, müssen an das Modul übergeben werden. Dies kann einzeln, über ein Array oder über eine Collection gemacht werden.

`CollectionTreeModule( String sessionName)`	Dem Konstruktor muss ein Session-Name übergeben werden. Dieser dient als Name für die Ansicht im UI.
`void add( ADVGeneralTreeNode<T> node)`	Hinzufügen einer einzelnen Node, falls sie noch nicht enthalten ist.
`void add(Collection<? extends ADVGeneralTreeNode<T>> nodes)`	Hinzufügen mehrerer Nodes mit einer Collection, falls sie noch nicht enthalten sind.
`void add( ADVGeneralTreeNode<T>[] nodes)`	Hinzufügen mehrerer Nodes mit einem Array, falls sie noch nicht enthalten sind.
`void remove( ADVGeneralTreeNode<T> node)`	Entfernen einer Node, falls sie enthalten ist.
`void remove(Collection<? extends ADVGeneralTreeNode<T>> nodes)`	Entfernen mehrerer Nodes mit einer Collection, falls sie enthalten sind.
`void remove( ADVGeneralTreeNode<T>[] nodes)`	Entfernen mehrerer Nodes mit einem Array, falls sie enthalten sind.
`void addRoot( ADVGeneralTreeNode<T> root)`	Hinzufügen eines Root-Knotens und seiner Kind-Knoten.
`void removeRoot( ADVGeneralTreeNode<T> root)`	Löschen eines Root-Knotens und seiner Kind-Knoten, wenn diese vorhanden sind.
`void setPosition( ModulePosition position)`	Setzen der Modul-Position im UI. Mögliche Werte sind: DEFAULT, LEFT, TOP, RIGHT, BOTTOM

Tabelle C.7.: Methoden CollectionTreeModule

Interface ADVGeneralTreeNode<T>

Alle Nodes eines Collection Trees, welche im ADV dargestellt werden sollen, müssen das Interface ADVGeneralTreeNode<T> implementieren. Die Dokumentation des Interfaces ist dem Abschnitt «Interface ADVGeneralTreeNode<T>» im Kapitel «General Tree Module» zu entnehmen.

C.7.2. Beispiel

Das Beispiel des Collection Trees zeigt einen Forest mit drei Trees. Durch das Hinzufügen einer neuen Root werden die einzelnen Trees zu einem grossen Tree.
Die Implementierung der SimpleGeneralTreeNode wurde aus dem Beispiel im Kapitel «General Tree Module» übernommen.

```
public class SimpleCollectionTreeProgram {

   public static void main(String[] args) throws ADVException {
      // connect to (and optionally start) the UI
      ADV.launch(null);

      // create nodes and set relations
      SimpleGeneralTreeNode nodeA = new SimpleGeneralTreeNode("A");
      SimpleGeneralTreeNode nodeB = new SimpleGeneralTreeNode("B");
      nodeA.addChild(nodeB);
      SimpleGeneralTreeNode nodeC = new SimpleGeneralTreeNode("C");
      nodeA.addChild(nodeC);
      SimpleGeneralTreeNode nodeD = new SimpleGeneralTreeNode("D");
      SimpleGeneralTreeNode nodeE = new SimpleGeneralTreeNode("E");
      SimpleGeneralTreeNode nodeF = new SimpleGeneralTreeNode("F");
      nodeE.addChild(nodeF);
      SimpleGeneralTreeNode nodeG = new SimpleGeneralTreeNode("G");
      nodeE.addChild(nodeG);

      CollectionTreeModule<String> module =
            new CollectionTreeModule<>("Simple Collection Tree");
      module.add(new SimpleGeneralTreeNode[]{nodeA, nodeB, nodeC, nodeD,
            nodeE, nodeF, nodeG});

      // send current state of graph to the UI to be displayed
      ADV.snapshot(module);

      // add new root
      SimpleGeneralTreeNode root = new SimpleGeneralTreeNode("NEW ROOT");
      root.addChild(nodeA);
      root.addChild(nodeD);
      root.addChild(nodeE);
      module.add(root);
      ADV.snapshot(module);
   }

}
```

Auflistung C.6: Beispiel-Darstellung eines Collection Trees

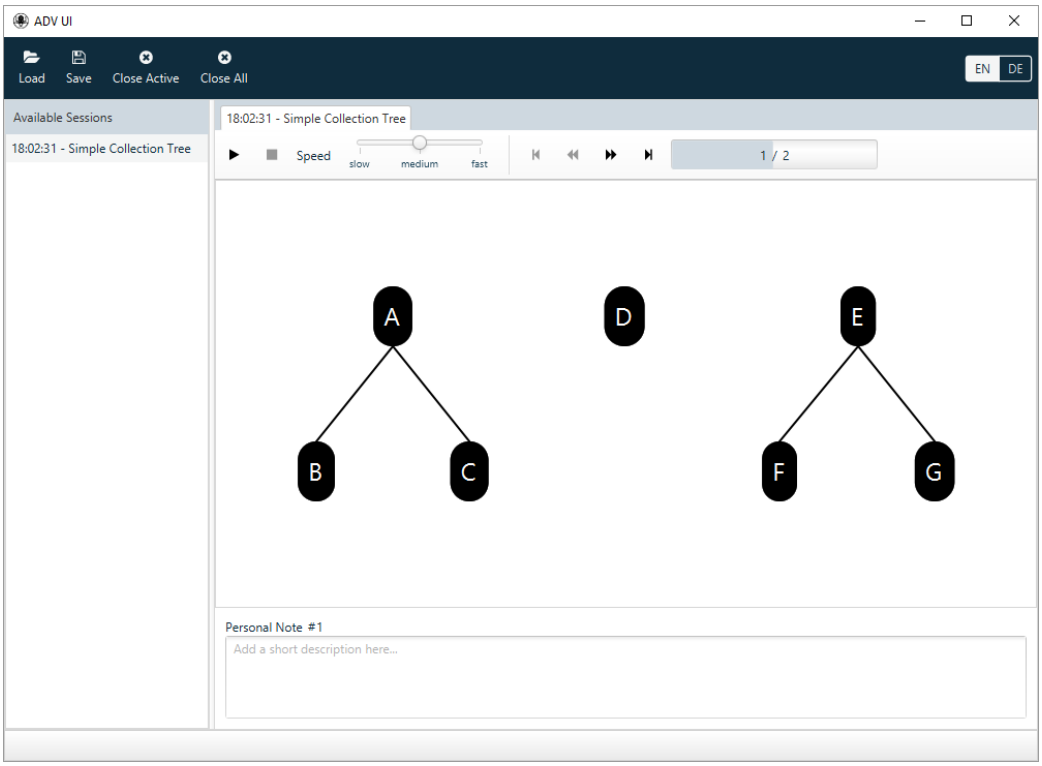


Abbildung C.7.: Beispiel-Darstellung eines Forests

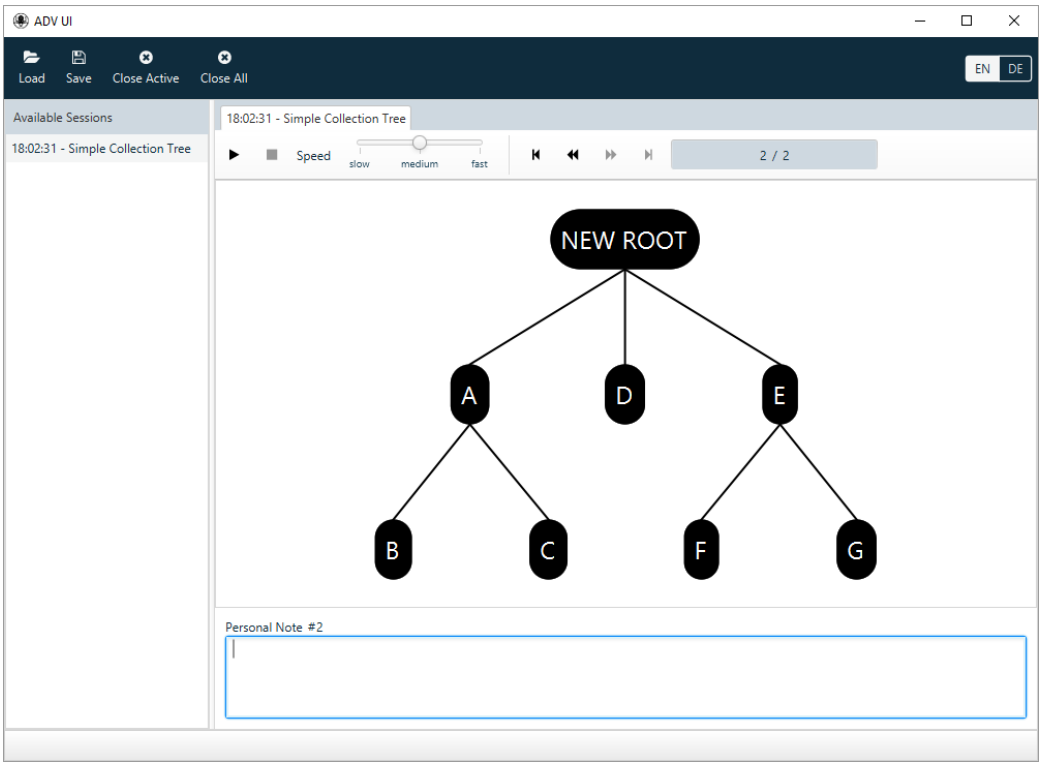


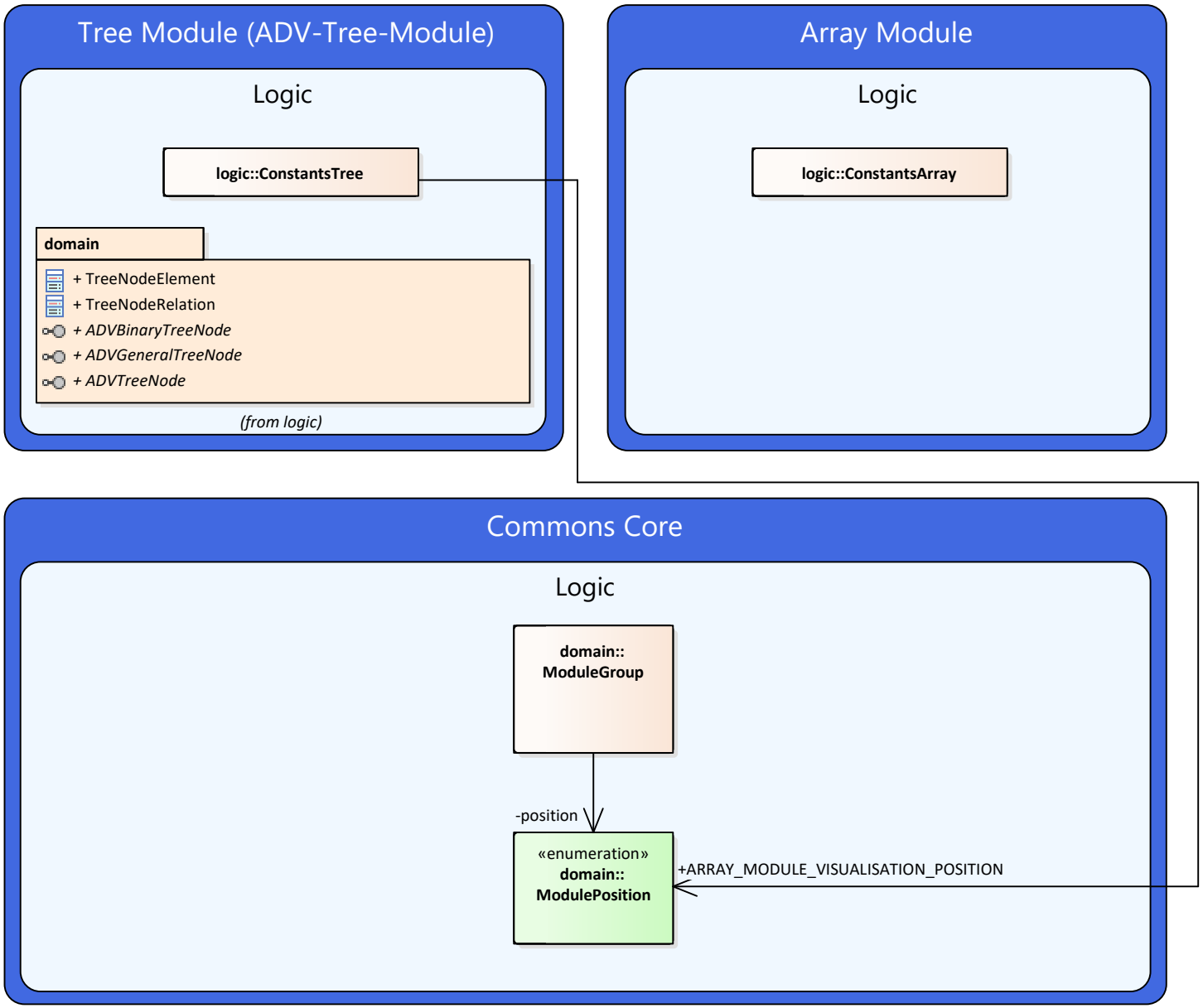
Abbildung C.8.: Darstellung des Forests nach dem Hinzufügen einer neuen Root

ANHANG D

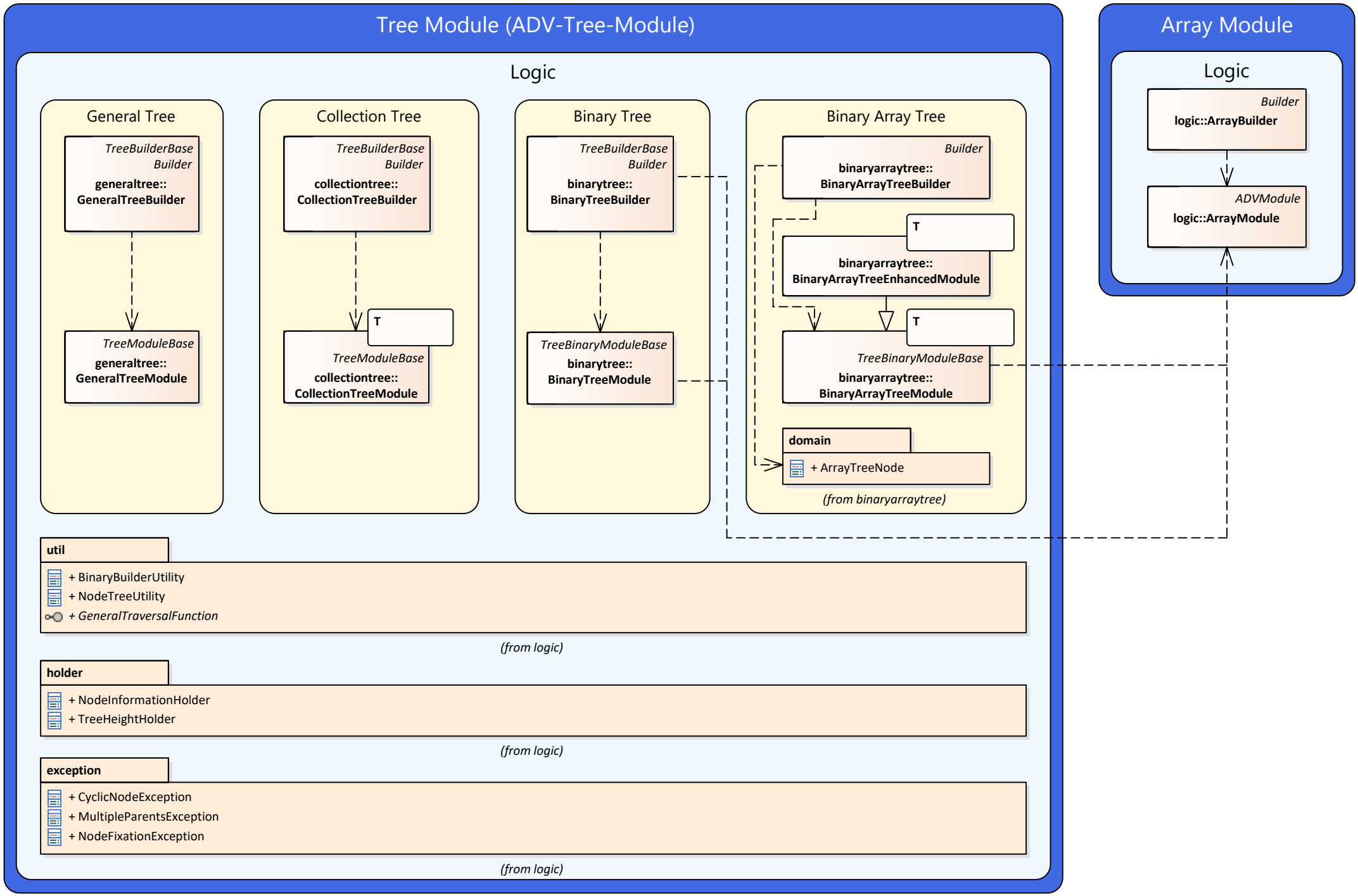
Klassendiagramme

In diesem Kapitel befinden sich Klassendiagramme, welche die wichtigsten Klassen und ihre Layer zeigen. Zur Verringerung der Komplexität sind die Diagramme in die drei Projekte Commons, Lib und UI aufgeteilt.

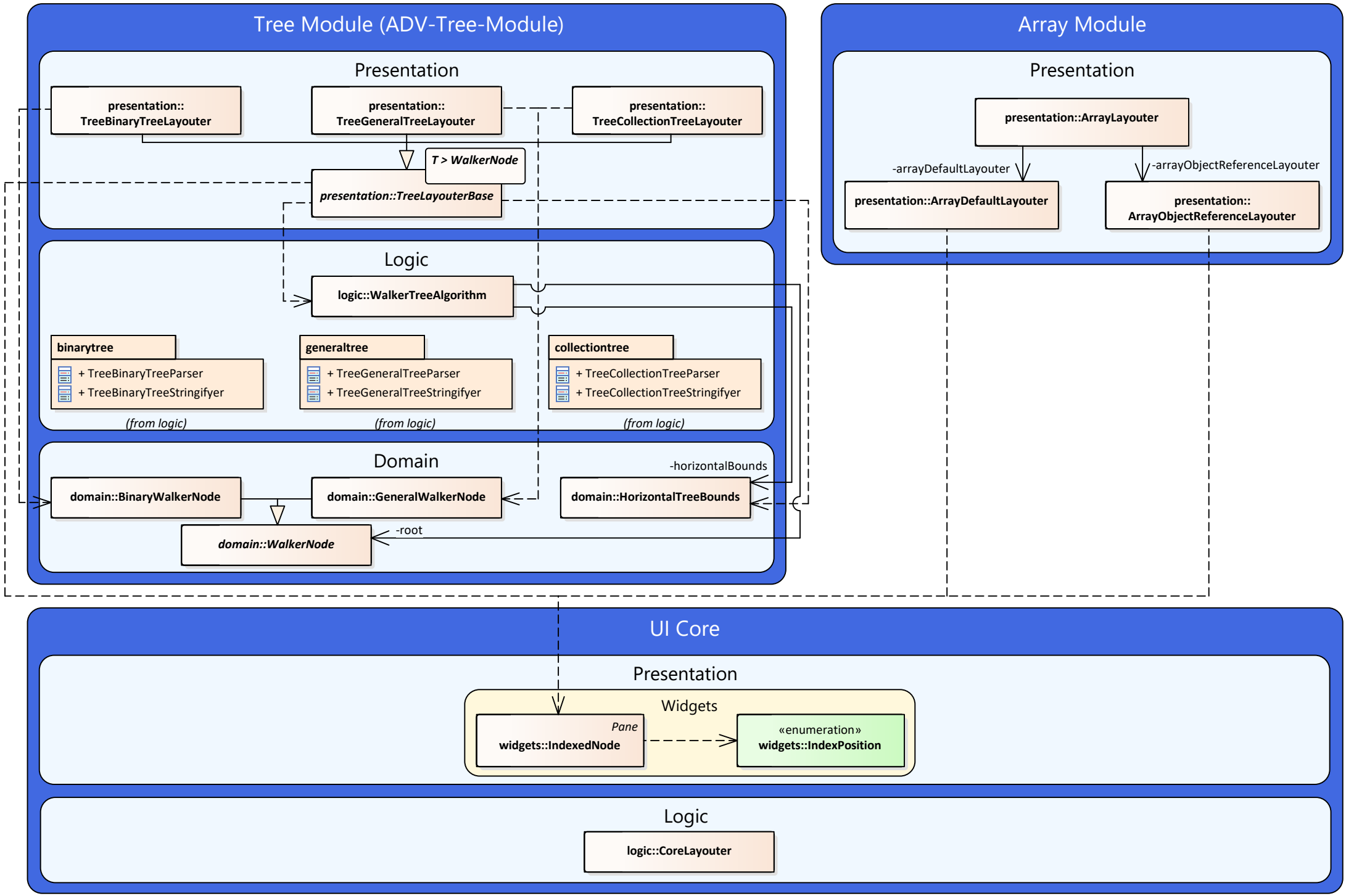
Klassendiagramm ADV-Commons



Klassendiagramm ADV-Lib



Klassendiagramm ADV-UI



ANHANG E

Usability-Tests

Zur Überprüfung der Verständlichkeit des ADV-Tree-Modules wurden einmal Usability-Tests mit vier Studenten, welche das Unterrichts-Modul AD2 besuchen, durchgeführt.

E.1. Testszenarien Verständnis

In den Unterrichtsmodulen AD1 und AD2 hast du bereits einiges zur Datenstruktur Baum gelernt. Nun hast du erfahren, dass es ein neues Programm gibt, den ADVisualizer (ADV). Mit diesem Programm können unter anderem auch Bäume dargestellt werden. Um das Gelernte besser zu verstehen, hast du dich entschieden, den ADV auszuprobieren. Bitte versuche nun folgende Szenarien zu lösen.

E.1.1. Szenario 1: Korrekte Darstellung

Wir haben dir ein Programm bereitgestellt, mit dem ein Baum im ADV dargestellt wird. Führe nun das Programm aus, warte bis das Fenster und der Baum dargestellt wird und benenne dann die wichtigsten Elemente des Baumes, die du erkennst.

E.1.2. Szenario 2: Binärbaum im Array

Vom AD-Unterricht weisst du, dass Binärbäume auch in einem Array gespeichert werden können. Das bereitgestellte Programm visualisiert den Zusammenhang zwischen dem Baum und dem Array. Führe das Programm aus und ordne die Knoten des Baumes den entsprechenden Array-Einträgen zu.

E.1.3. Szenario 3: Forest

Der ADV hat auch die Möglichkeit Forests darzustellen. Führe das Programm aus und versuche die verschiedenen Bäume auseinanderhalten. Zeige dazu mit dem Finger die Trennlinien zwischen den Bäumen.
Sieh dir auch den Code an. Kannst du erkennen, wie der Forest zustande kommt?

E.1.4. Szenario 4: Fehlende Kindknoten

Dieses Programm visualisiert einen Binärbaum, bei dem einige Knoten nur ein Kind haben. Führe das Programm aus, suche die Knoten, welche nur ein Kind haben und erkläre, welches der beiden Kinder vorhanden ist bzw. welches fehlt.

E.1.5. Szenario 5: Rotationen

Im AD-Unterricht hast du AVL-Bäume kennengelernt und weisst, dass sie Rotationen ausführen, um sich auszubalancieren. Wir haben dir ein Programm bereitgestellt, das einen AVL-Baum visualisiert. Klicke durch die Snapshots der Session und versuche die erlernte Theorie aus dem Unterricht in der Visualisierung zu erklären.

E.2. Testszenarien Programmieren

Du hast gesehen, dass der ADV verschiedenste Bäume darstellen kann. Nun möchtest du aber auch deine eigenen Baum-Implementationen visualisieren. Lies dazu die folgenden Szenarien durch und versuche den Code zu erstellen, um die Bäume korrekt darzustellen.

E.2.1. Szenario 6: Darstellung des Arrays

Wir haben dir ein Array bereitgestellt. Du erkennst in dem Array die Struktur eines Baumes wieder, bist dir aber nicht sicher, wie der Baum aussehen wird. Daher möchtest du den Baum im Array und das Array selbst im ADV darstellen. Lies dazu die Beschreibung des Moduls im Benutzerhandbuch, erstelle den notwendigen Code und führe diesen aus.

E.2.2. Szenario 7: Knoten hinzufügen

Wir haben dir nun ein Programm bereitgestellt, das einen anderen Array darstellt. Beim Ausführen des Programms erkennst du, dass der Baum darin ziemlich leer ist. Daher möchtest du dem Array einen weiteren Knoten hinzufügen. Füge dem Root-Knoten ein linkes Kind mit der Bezeichnung «left-child» hinzu und stelle den neuen Baum in einem zweiten Snapshot dar. Benutze bei Unklarheiten die bereitgestellte Interface-Beschreibung im Benutzerhandbuch.

E.2.3. Szenario 8: Problem mit dem Array

Nun möchtest du einen anderen Array-Baum im ADV visualisieren. Das Programm scheint jedoch fehlerhaft zu sein. Starte das Programm, um die Fehler herauszufinden und korrigiere sie anschliessend.

E.2.4. Szenario 9: Array vergrössern

Du bekommst von uns einen Array mit einem vollständig befüllten Baum. Erweitere nun das Array, sodass Blatt «C» ein rechtes Kind «D» erhält. Visualisiere das Ergebnis in einem zweiten Snapshot. Zur Hilfe benutzt du die Interface-Beschreibung im Benutzerhandbuch.

E.2.5. Szenario 10: Problem im Baum

Zuletzt möchtest du einen allgemeinen Baum visualisieren. Den notwendigen Code hast du bereits geschrieben. Allerdings hat sich im Programm ein Fehler eingeschlichen. Bitte führe das Programm aus und versuche die Ursache des Problems herauszufinden und zu korrigieren.

Usability-Test: ADV-Tree-Module

Studiengang: I
Semester: 3. Semester
Datum 26.11.2018 10:30 - 12:00

Bewertungsskala

4	Tester hat keinen Ansatz und muss die Hilfe der Betreuer in Anspruch nehmen
3	Tester hat Probleme, kommt aber selbständig auf ein Ergebnis (gut oder schlecht)
2	Es gibt Unklarheiten, die aber schnell geklärt werden können, danach kann der Tester ohne Probleme weiterarbeiten
1	Keine Probleme

#	Szenario	Feedback der Testperson	Bemerkung der Betreuer	Bewertung
1	Korrekte Darstellung			1
2	Binärbaum im Array	ev. Farben einsetzen damit das Mapping klar wird	Index wurde nicht gebraucht für das Mapping	2
3	Forests	es ist mühsam die Knoten zweimal hinzuzufügen; Inkonsistenz zu anderen Modulen		1
4	Fehlende Kindknoten	Falls ein Knoten fehlt, könnte ein leerer Knoten dargestellt werden (aber das wäre wahrscheinlich eine Überladung des UI)		1
5	Rotationen	Einfügen und Prüfen auf die Höhe sind zwei Teilschritte; gelbe Markierung ist nicht klar; x, y, z wären hilfreiche Labels; Label hinzufügen: Node eingefügt, Node wird rotiert, etc.; Farben nach Rotation auch noch anzeigen		3
6	Darstellung des Arrays		Nicht klar, dass es ein Binary Tree ist; BinaryTreeModule und BinaryArrayTreeModule verwechselt	2
7	Knoten hinzufügen			1
8	Problem mit dem Array	Fehlermeldung nicht klar (müsste 2^(height+1) sein)		3
9	Array vergrössern	es ist mühsam die Null Knoten aufzufüllen; ev. Automatisch machen	Testperson hat addChild-Methode auf dem Modul gesucht	3
10	Problem im Baum	Bezeichnung "Rang" war nicht mehr so präsent; ev. Content ausgeben		1

Anmerkungen des Testers

-Nach Einführung was es für Module gibt und dass es eine snapshot-Methode gibt, wäre die Benutzung klar

-bei getchildren: Blätter sollen null oder leere Liste zurückgeben

-Platz wäre besser ausgenutzt, wenn das Array unten dargestellt würde

-Beim Inhaltsverzeichnis öffnen, wenn Benutzerhandbuch gebraucht wird

Anmerkungen der Betreuer

-GVS nie gross gebraucht

Usability-Test: ADV-Tree-Module

Studiengang: I
Semester: 5.Semester
Datum 26.11.2018 12:15 - 13:00

Bewertungsskala

4	Tester hat keinen Ansatz und muss die Hilfe der Betreuer in Anspruch nehmen
3	Tester hat Probleme, kommt aber selbständig auf ein Ergebnis (gut oder schlecht)
2	Es gibt Unklarheiten, die aber schnell geklärt werden können, danach kann der Tester ohne Probleme weiterarbeiten
1	Keine Probleme

#	Szenario	Feedback der Testperson	Bemerkung der Betreuer	Bewertung
1	Korrekte Darstellung			1
2	Binärbaum im Array	Indizes über dem Array wären hilfreich	Index wurde nicht gebraucht für das Mapping	2
3	Forests	Verrwirrend, dass die Knoten auch zum Modul hinzugefügt werden müssen; Roots hinzufügen würde helfen		1
4	Fehlende Kindknoten			1
5	Rotationen	x, y und z hinzufügen für mehr Klarheit		1
6	Darstellung des Arrays		BinaryTreeModule und BinaryArrayTreeModule verwechselt	2
7	Knoten hinzufügen			1
8	Problem mit dem Array	es müsste 2^(height + 1) sein	Höhe automatisch auffüllen	2
9	Array vergrössern			1
10	Problem im Baum			1

Anmerkungen des Testers
-bei getchildren: Blätter sollen null oder leere Liste zurückgeben
-Array neben dem Baum fällt mehr auf, anstatt unterhalb des Baumes

Anmerkungen der Betreuer
-GVS nicht häufig genutzt

Usability-Test: ADV-Tree-Module

Studiengang: I
Semester: 3. Semester
Datum 30.11.2018 11.15 - 11:45

Bewertungsskala

4	Tester hat keinen Ansatz und muss die Hilfe der Betreuer in Anspruch nehmen
3	Tester hat Probleme, kommt aber selbständig auf ein Ergebnis (gut oder schlecht)
2	Es gibt Unklarheiten, die aber schnell geklärt werden können, danach kann der Tester ohne Probleme weiterarbeiten
1	Keine Probleme

#	Szenario	Feedback der Testperson	Bemerkung der Betreuer	Bewertung
1	Korrekte Darstellung			1
2	Binärbaum im Array	Indizes unter dem Array wären hilfreich		1
3	Forests	Hilfreich wäre es, nur die roots hinzuzufügen und dieKindknoten werden automatisch dazugeneriert		1
4	Fehlende Kindknoten			1
5	Rotationen	Nach Rotation eventuell noch einmal anzeigen, was rotiert wurde		1
6	Darstellung des Arrays		Die Testperson hat das Szenario mit Hilfe des Beispielcodes gelöst; Tester hat zuerst setArray anstatt showArray genommen	2
7	Knoten hinzufügen		Der Tester hat das BinaryTreeModule mit dem BinaryArrayTreeModule verwechselt	2
8	Problem mit dem Array		Die Betreuer haben den Hinweis auf die korrekte Formel 2^(height + 1) gegeben	1
9	Array vergrössern			2
10	Problem im Baum			1

Anmerkungen des Testers

-bei getchildren: Blätter sollen null oder leere Liste zurückgeben
-Arraydarstellung unterhalb des Baumes wäre sinnvoller

Anmerkungen der Betreuer

-Der GVS wurde benutzt und war ok bis auf den Style und den Bug

Usability-Test: ADV-Tree-Module

Studiengang: I
Semester: 3. Semester
Datum 30.11.2018 13:30 - 14:15

Bewertungsskala

4	Tester hat keinen Ansatz und muss die Hilfe der Betreuer in Anspruch nehmen
3	Tester hat Probleme, kommt aber selbständig auf ein Ergebnis (gut oder schlecht)
2	Es gibt Unklarheiten, die aber schnell geklärt werden können, danach kann der Tester ohne Probleme weiterarbeiten
1	Keine Probleme

#	Szenario	Feedback der Testperson	Bemerkung der Betreuer	Bewertung
1	Korrekte Darstellung			1
2	Binärbaum im Array			1
3	Forests		Vorschlag mit addRoot Methode fand der Tester gut	1
4	Fehlende Kindknoten			1
5	Rotationen	Die Farben sind hilfreich		1
6	Darstellung des Arrays			1
7	Knoten hinzufügen	Add Methode auf dem Modul mit Index vom Parent und Inhalt wäre bequem	Wahrscheinlich ist der Lerneffekt kleiner mit einer Methode	1
8	Problem mit dem Array			1
9	Array vergrössern	Gleich wie bei Szenario 7, eine Add Methode wäre bequem	Die Testperson hat vergessen setArray aufzurufen	2
10	Problem im Baum	In der Fehlermeldung wäre vielleicht die Höhe auf der der Fehler aufgetreten ist sinnvoller als der Rank		1

Anmerkungen des Testers
-bei getchildren: Blätter sollen null zurückgeben
-Es wäre vielleicht hilfreich die Indizes mit eckigen Klammern einzuklammern

-Die Arraydarstellung nach unten zu verlagern würde das Verständnis nicht wesentlich erhöhen
-Hilfreich wäre es auf einen Array-Eintrag zu klicken und der entsprechende Knoten im Baum würde markiert werden

Anmerkungen der Betreuer
-Der Tester hat den GVS in den Übungen gebraucht

ANHANG F

Systemtests

Tests mit dem UI, welche nicht automatisiert werden konnten, wurden in Form von Systemtestdefinitionen aufgeschrieben und zweimal während der Studienarbeit ausgeführt.

ADV Systemtests

System under Test	https://github.com/ADVisualizer
Test File Verzeichnis	https://github.com/ADVisualizer/ADV-User_Codebase/tree/develop/src/main/java/ch/hsr
Datum	30.11.2018
Testergebnis	100%

Legende	
+	Bestanden
-	Nicht bestanden

ADV-Tree-Module (/tree)

Binary Tree Module

#	Test File	Test Beschreibung	Erwartetes Resultat	Ergebnis	Bemerkungen
1	BinaryTraversal.java	Binary Tree Darstellung	Der Binärbaum wird ohne Array visualisiert. Sämtliche Kinder werden links bzw. rechts unter ihrem Parent dargestellt.	+	setShowArray sollte auskommentiert werden.
2	BinaryTraversal.java	Binary Tree mit Array-Darstellung	Der Binärbaum und das dazugehörige Array werden nebeneinander dargestellt.	+	
3	BinaryTraversal.java	Binary Tree mit Styles	Der Binärbaum wird dargestellt. Beim Durchklicken der Snapshots werden die jeweils traversierten Nodes durch Styles hervorgehoben.	+	
4	BinaryTraversal.java	Binary Tree mit Indizes	Neben den Nodes werden die Indizes entsprechend dem zugehörigen Array-Eintrag dargestellt.	+	
5	DisplayableAVLTree.java	Binary Tree mit Rotationen	Ein Baum zeigt bei einer Rotationen die korrekten Zwischenstufen und zum Schluss den ausbalancierten Baum an	+	

Binary Array Tree Module

#	Test File	Test Beschreibung	Erwartetes Resultat	Ergebnis	Bemerkungen
6	BinaryArrayTreeShowcase.java	Binary Tree ohne Array-Darstellung	Der Binärbaum wird ohne Array-Darstellung visualisiert	+	
7	BinaryArrayTreeShowcase.java	Binary Tree mit Array-Darstellung	Der Binärbaum und die zugehörige Array-Darstellung werden nebeneinander visualisiert	+	Durchgehen der Snapshots (4. Snapshot)
8	BinaryArrayTreeShowcase.java	Hinzufügen eines neuen Knotens	Der neue Knoten wird im Baum und im Array an der korrekten Stelle angezeigt	+	setShowArray sollte nach Hinzufügen eingefügt werden
9	BinaryArrayTreeShowcase.java	Entfernen eines Knotens	Der Knoten erscheint nicht mehr im Baum und im Array und die Abstände zwischen den Knoten bleiben gleich wie vorher	+	
10	BinaryArrayTreeShowcase.java	Entfernen eines Elternknotens	Der Eltern- und die Kindknoten werden im Baum nicht mehr angezeigt; im Array werden die Kindknoten noch dargestellt	+	Vor dem Löschen des Elternknoten musste manuell ein Kindknoten hinzugefügt werden

General Tree Module					
#	**Test File**	**Test Beschreibung**	**Erwartetes Resultat**	**Ergebnis**	**Bemerkungen**
11	GeneralTraversal.java	General Tree Darstellung	Der Baum wird visualisiert. Die Kinder befinden sich von links nach rechts in derselben Reihenfolge, in der sie den Nodes hinzugefügt wurden	+	
12	GeneralTraversal.java	General Tree mit Styles	Beim Durchklicken der Snapshots werden die bereits traversierten Nodes mit Styles hervorgehoben.	+	
13	GeneralTreeModification.java	Hinzufügen eines neuen Knotens	Der neue Knoten wird im Baum an der korrekten Stelle angezeigt	+	
14	GeneralTreeModification.java	Entfernen eines Knotens	Der Knoten erscheint nicht mehr im Baum.	+	
15	GeneralTreeModification.java	Entfernen eines Elternknotens	Der Eltern- und die Kindknoten werden im Baum nicht mehr angezeigt	+	

Collection Tree Module					
#	**Test File**	**Test Beschreibung**	**Erwartetes Resultat**	**Ergebnis**	**Bemerkungen**
16	CollectionTreeModification.java	Forest Darstellung	Mehrere Bäume werden nebeneinander dargestellt.	+	
17	CollectionTreeModification.java	Collection Tree mit Styles	Die Nodes in den Bäumen werden mit den korrekten Styles visualisiert	+	Ein Style muss manuell hinzugefügt werden.
18	CollectionTreeModification.java	Kindknoten wird nicht hinzugefügt	Der Kindknoten wird nicht angezeigt. Alle Kinder des Kindknotens werden als Root-Knoten dargestellt	+	

ADV Systemtests

System under Test	https://github.com/ADVisualizer
Test File Verzeichnis	https://github.com/ADVisualizer/ADV-User_Codebase/tree/develop/src/main/java/ch/hsr
Datum	06.12.2018
Testergebnis	100%

Legende	
+	Bestanden
-	Nicht bestanden

ADV-Tree-Module (/tree)

Binary Tree Module

#	Test File	Test Beschreibung	Erwartetes Resultat	Ergebnis	Bemerkungen
1	BinaryTraversal.java	Binary Tree Darstellung	Der Binärbaum wird ohne Array visualisiert. Sämtliche Kinder werden links bzw. rechts unter ihrem Parent dargestellt.	+	
2	BinaryTraversal.java	Binary Tree mit Array-Darstellung	Der Binärbaum und das dazugehörige Array werden untereinander dargestellt.	+	
3	BinaryTraversal.java	Binary Tree mit Styles	Der Binärbaum wird dargestellt. Beim Durchklicken der Snapshots werden die jeweils traversierten Nodes durch Styles hervorgehoben.	+	
4	BinaryTraversal.java	Binary Tree mit Indizes	Neben den Nodes und über dem Array werden die Indizes entsprechend dem zugehörigen Array-Eintrag dargestellt.	+	
5	DisplayableAVLTree.java	Binary Tree mit Rotationen	Ein Baum zeigt bei einer Rotation die korrekten Zwischenstufen und zum Schluss den ausbalancierten Baum an.	+	
6	DisplayableAVLTree.java	Nodes sind fixiert	Alle Nodes, welche nicht von einer Rotation betroffen sind, bleiben an derselben Position.	+	

Binary Array Tree Module

#	Test File	Test Beschreibung	Erwartetes Resultat	Ergebnis	Bemerkungen
7	BinaryArrayTreeShowcase.java	Binary Tree ohne Array-Darstellung	Der Binärbaum wird ohne Array-Darstellung visualisiert	+	
8	BinaryArrayTreeShowcase.java	Binary Tree mit Array-Darstellung	Der Binärbaum und die zugehörige Array-Darstellung werden untereinander visualisiert	+	
9	BinaryArrayTreeShowcase.java	Hinzufügen eines neuen Knoten	Der neue Knoten wird im Baum und im Array an der korrekten Stelle angezeigt	+	
10	BinaryArrayTreeShowcase.java	Entfernen eines Knotens	Der Knoten erscheint nicht mehr im Baum und im Array und die Abstände zwischen den Knoten bleiben gleich wie vorher	+	
11	BinaryArrayTreeShowcase.java	Entfernen eines Elternknoten	Der Eltern- und die Kindknoten werden im Baum nicht mehr angezeigt; im Array werden die Kindknoten noch dargestellt	+	
12	BinaryArrayTreeShowcase.java	Entfernen der Root	Nach dem Entfernen der Root, steht im Root-Eintrag des Arrays null. Beim Baum wird eine leere Fläche dargestellt	+	
13	BinaryArrayTreeShowcase.java	Binary Tree mit Styles	Nachdem ein Snapshot mit einem markierten Array-Eintrag ans UI gesendet wurde, wird der Array-Eintrag und die zugehörige Node im Baum entsprechend hervorgehoben	+	

General Tree Module					
#	***Test File***	***Test Beschreibung***	***Erwartetes Resultat***	***Ergebnis***	***Bemerkungen***
14	GeneralTraversal.java	General Tree Darstellung	Der Baum wird visualisiert. Die Kinder befinden sich von links nach rechts in derselben Reihenfolge, in der sie den Nodes hinzugefügt wurden	+	
15	GeneralTraversal.java	General Tree mit Styles	Beim Durchklicken der Snapshots werden die bereits traversierten Nodes mit Styles hervorgehoben.	+	
16	GeneralTreeModification.java	Hinzufügen eines neuen Knoten	Der neue Knoten wird im Baum an der korrekten Stelle angezeigt	+	
17	GeneralTreeModification.java	Entfernen eines Knotens	Der Knoten erscheint nicht mehr im Baum.	+	
18	GeneralTreeModification.java	Entfernen eines Elternknotens	Der Eltern- und die Kindknoten werden im Baum nicht mehr angezeigt	+	

Collection Tree Module					
#	***Test File***	***Test Beschreibung***	***Erwartetes Resultat***	***Ergebnis***	***Bemerkungen***
19	CollectionTreeModification.java	Forest Darstellung	Mehrere Bäume werden nebeneinander dargestellt.	+	
20	CollectionTreeModification.java	Collection Tree mit Styles	Die Nodes in den Bäumen werden mit den korrekten Styles visualisiert	+	
21	CollectionTreeModification.java	Kindknoten wird nicht hinzugefügt	Der Kindknoten wird nicht angezeigt. Alle Kinder des Kindknotens werden als Root-Knoten dargestellt	+	
22	CollectionTreeModification.java	Root Knoten (addRoot-Methode) wird zum Modul hinzugefügt	der ganze Baum wird korrekt im UI dargestellt und ist unterteilt von den anderen Bäumen	+	
23	CollectionTreeModification.java	Root Knoten (removeRoot-Methode) wird vom Modul entfernt	der Baum wird nicht mehr im UI angezeigt	+	

ADV-Mixed-Modules (/multi)					
Module Positioning					
#	***Test File***	***Test Beschreibung***	***Erwartetes Resultat***	***Ergebnis***	***Bemerkungen***
24	MultiPositionModule.java	Modul mit vier Child-Modulen mit jeweils anderer Position (left, right, bottom, top)	Modul ist in der Mitte und alle Child-Module sind links, rechts, unten und oben darum verteilt	+	
25	MultiPositionModule.java	Modul mit zwei Child-Modulen, die die gleiche Position haben (links)	Die zwei Child-Module werden links vom Modul dargestellt, wobei das auf der linken Seite dem zuletzt hinzugefügten Child-Modul entspricht	+	

ANHANG G

Metriken

G.1. Metriken

G.1.1. Spotbugs und Checkstyle

Spotbugs und Checkstyle melden keine Fehler oder Warnungen für alle drei Projekte.

G.1.2. Codacy Code-Quality

Codacy prüft die Projekte auf verschiedene Eigenschaften, um die Code Qualität zu bewerten. Unter anderem beurteilt es Conditional Complexity und Code-Duplizierung und führt auch eine statische Code-Analyse durch. Der Status kann auch online abgerufen werden: `https://app.codacy.com/organization/ADV`.

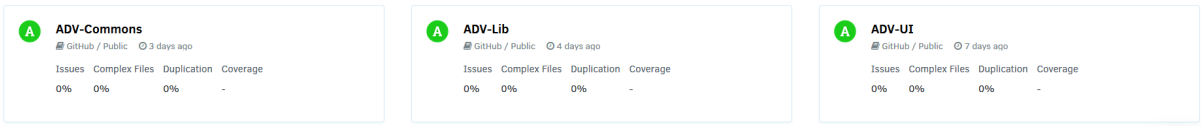


Abbildung G.1.: Bewertung der Code Qualität von Codacy

Laut Codacy ist die Qualität des ADV sehr gut. Es gibt keine offenen Issues, die bearbeitet werden müssen.

G.1.3. Build-Status

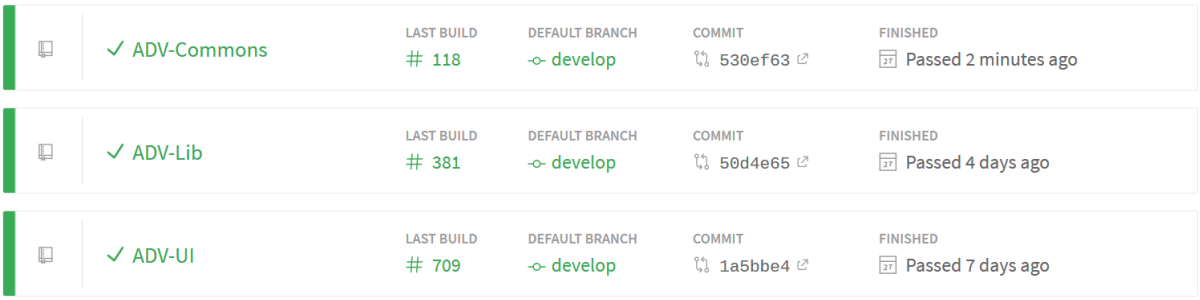


Abbildung G.2.: Build-Status der drei Projekte

Alle drei Projekte haben einen grünen Build Status. Dieser kann auch online abgerufen werden: `https://travis-ci.org/ADVisualizer`.

G.1.4. Code Coverage

Codecov erfasst die Coverage für die Projekte ADV Lib und ADV UI. ADV Commons enthält keine Tests, da es sich dort entweder um Interfaces, Datenklassen oder Enums handelt.

Files	≡	●	●	●	Coverage
binaryarraytree	128	120	4	4	93.75%
binarytree	93	91	1	1	97.85%
collectiontree	74	72	1	1	97.30%
exception	6	6	0	0	100.00%
generaltree	31	24	2	5	77.42%
holder	26	18	3	5	69.23%
util	40	34	1	5	85.00%
TreeBinaryModuleBase.java	22	18	2	2	81.82%
TreeBuilderBase.java	10	10	0	0	100.00%
TreeModuleBase.java	12	9	0	3	75.00%
Folder Totals (10 files)	442	402	14	26	90.95%
Project Totals (56 files)	972	687	27	258	70.68%

Abbildung G.3.: Code Coverage im des ADV-Tree-Module im ADV Lib

Files	≡	●	●	●	Coverage
domain/BinaryWalkerNode.java	20	19	1	0	95.00%
domain/GeneralWalkerNode.java	7	7	0	0	100.00%
domain/HorizontalTreeBounds.java	11	11	0	0	100.00%
domain/WalkerNode.java	35	35	0	0	100.00%
logic/TreeParser.java	11	11	0	0	100.00%
logic/TreeStringifyer.java	8	8	0	0	100.00%
logic/WalkerTreeAlgorithm.java	114	104	5	5	91.23%
logic/binarytree/TreeBinaryTreeParser.java	2	2	0	0	100.00%
logic/binarytree/TreeBinaryTreeStringifyer.java	2	2	0	0	100.00%
logic/collectiontree/TreeCollectionTreeParser.java	2	2	0	0	100.00%
logic/collectiontree/TreeCollectionTreeStringifyer.java	2	2	0	0	100.00%
logic/generaltree/TreeGeneralTreeParser.java	2	2	0	0	100.00%
logic/generaltree/TreeGeneralTreeStringifyer.java	2	2	0	0	100.00%
presentation/TreeBinaryTreeLayouter.java	109	104	2	3	95.41%
presentation/TreeCollectionTreeLayouter.java	22	22	0	0	100.00%
presentation/TreeGeneralTreeLayouter.java	16	16	0	0	100.00%
presentation/TreeLayouterBase.java	93	93	0	0	100.00%
Folder Totals (17 files)	458	442	8	8	96.51%
Project Totals (75 files)	2,514	1,458	39	1,017	58.00%

Abbildung G.4.: Code Coverage im des ADV-Tree-Module im ADV UI

Insgesamt erreicht das ADV-Tree-Module eine sehr hohe Coverage im ADV Lib (ca. 91%) und im ADV UI (ca. 96%). Im ADV Lib gibt es Getter- und Setter-Methoden, die nicht getestet wurden, da dies unnötige Testfälle nach sich ziehen würde.
Das ADV-Tree-Module erreicht diese hohe Coverage, da der ADV viele GUI-spezifische Details kapselt und vor dem Modul versteckt. Daher kann effektiver getestet werden, ohne dass Tests mit aufwändigen Mocks geschrieben werden müssen. Genauere Informationen, welche Zeilen abgedeckt sind oder Details zu den Dateien, können online abgerufen werden: `https://codecov.io/gh/ADVisualizer`.

ANHANG H

Zeitauswertung

H.1. Zeitauswertung nach Meilensteinen

Genauere Beschreibungen zu den einzelnen Meilensteinen sind dem Projektplan zu entnehmen.

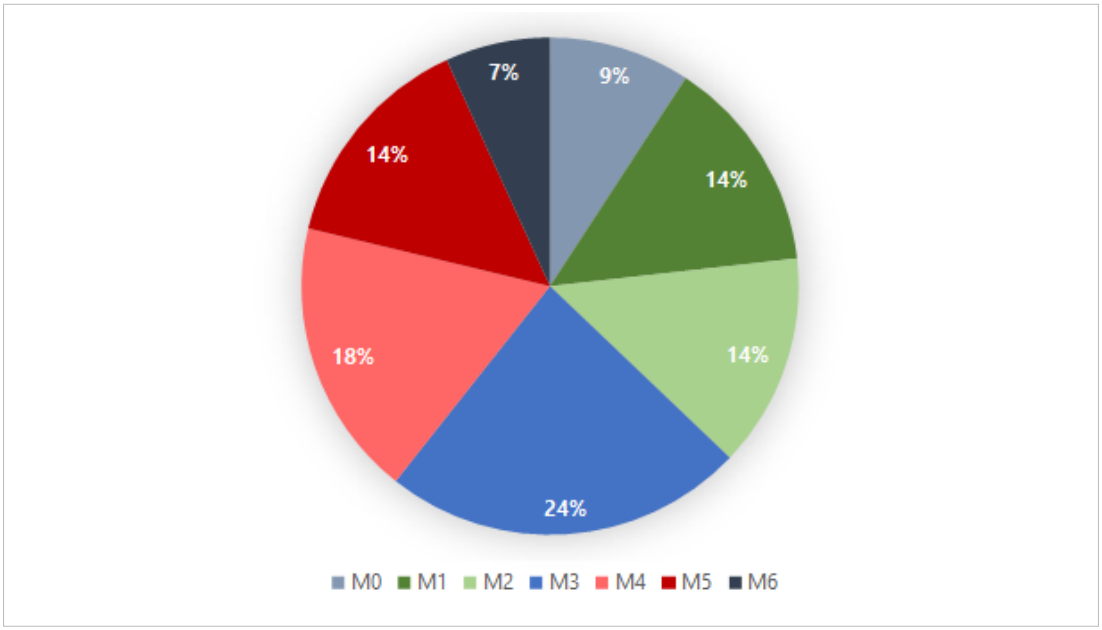


Abbildung H.1.: Zeitauswertung nach Meilensteinen

H.2. Zeitauswertung nach Aktivitäten

Anforderungen
Analyse und Erfassung der Anforderungen

Abschlussarbeiten
Schreiben der Dokumente für die Schlussabgabe

Umgebung/Werkzeuge
Aufsetzen der Entwicklungsumgebung und Werkzeuge

Projektmanagement
Projektspezifische Arbeiten wie Backlog-Management und Wochenbesprechungen

Review & Testing
Durchführungen von Code Reviews, Systemtests und Usability-Tests

Entwicklung
Schreiben von produktivem Code und Unit-Tests

Architektur & Design
Dokumentation der geplanten Architektur und des schliesslich angewandten Designs

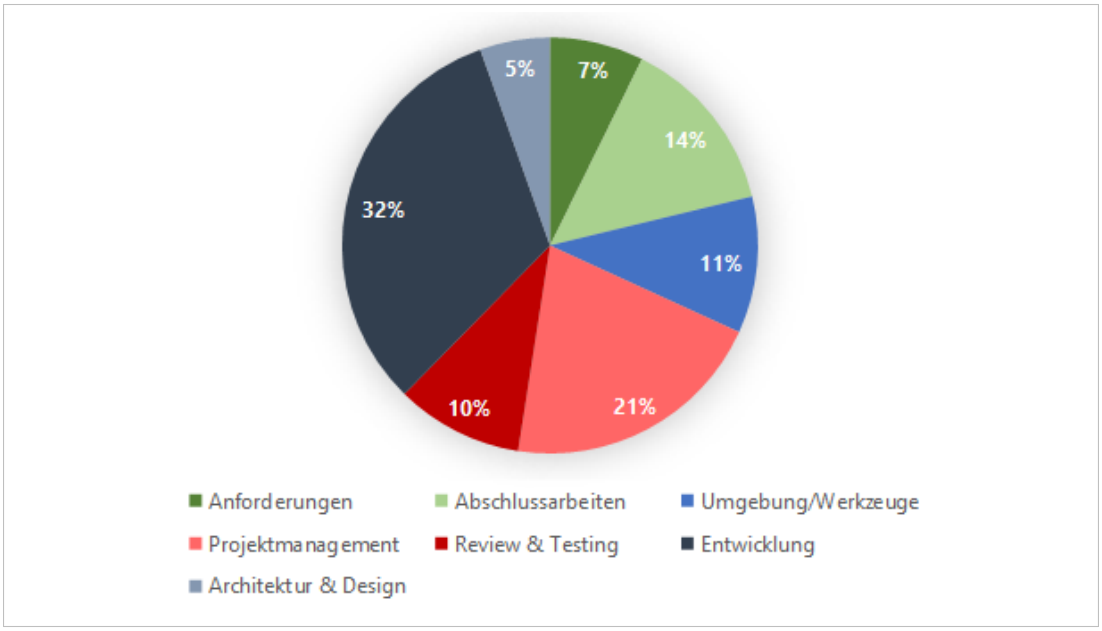


Abbildung H.2.: Zeitauswertung nach Aktivitäten

H.3. Zeitauswertung nach Teammitgliedern

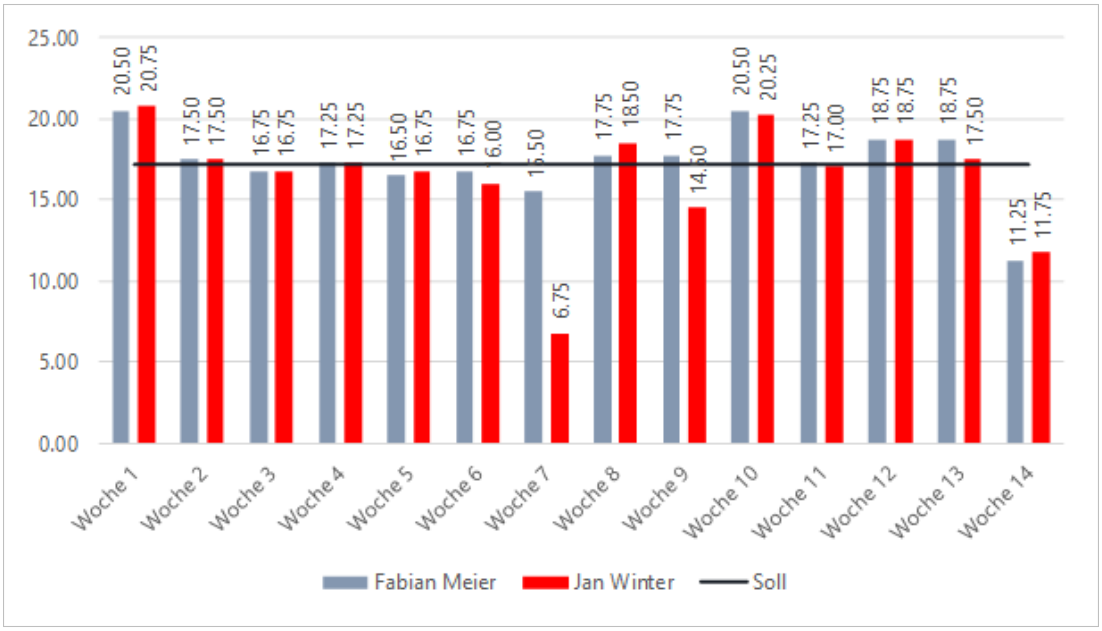


Abbildung H.3.: Zeitauswertung nach Teammitgliedern

H.4. Soll-Ist-Vergleich

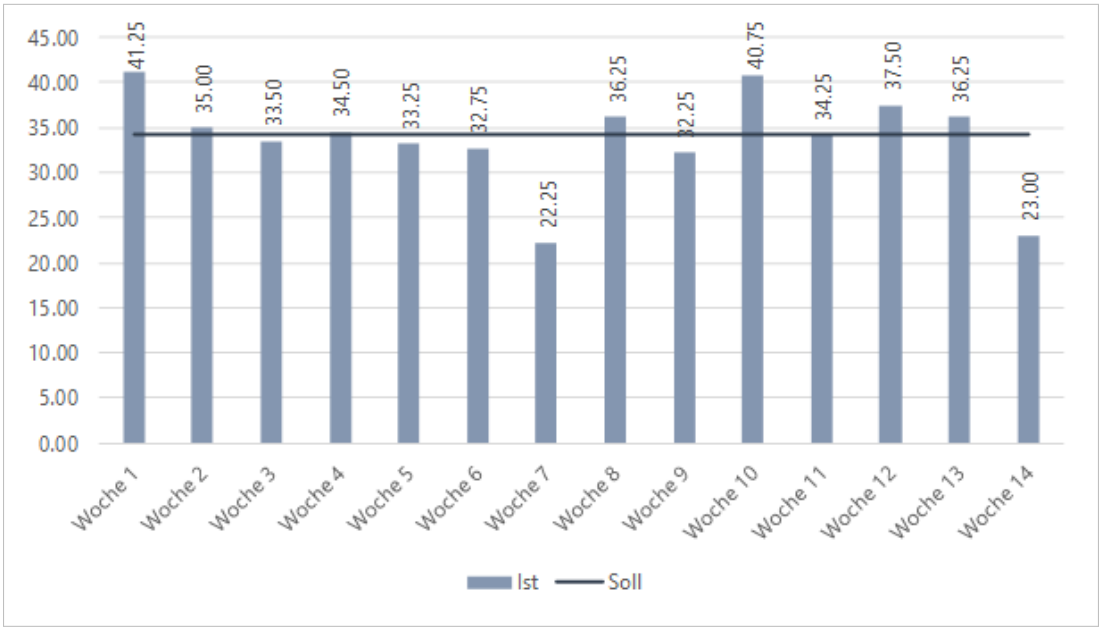


Abbildung H.4.: Soll-Ist-Vergleich

ANHANG I

Literaturverzeichnis

[1] Open-jdk. https://openjdk.java.net/, Oktober 2018.

[2] Thomas Letsch. Aufgabenstellung adv-tree-module.

[3] Thomas Letsch. Generelle richtlinien für studien- und bachelorarbeiten.

[4] Murièle Trentini und Michael Wieland. Graphs-visualization-service gvs 2.0, 2017.

[5] Murièle Trentini und Michael Wieland. Framework zur visualisierung von algorithmen und datenstrukturen, 2018.

[6] Frank Buschmann und Regine Meunier und Hans Rohnert und Peter Sommerlad und Michael Stal. *Pattern-oriented Software Architecture*. John Wiley & Sons Ltd, The Atrium, Southern Gate, Chichester, 1996.

ANHANG J

Auflistungsverzeichnis