HSR
HOCHSCHULE FÜR TECHNIK
RAPPERSWIL
FHO Fachhochschule Ostschweiz

Bachelorarbeit

Framework zur Visualisierung von Algorithmen und Datenstrukturen

Hochschule für Technik Rapperswil
Abteilung für Informatik

Zeitraum: 19.02.2018 - 15.06.2018

Autoren Murièle Trentini
Michael Wieland

Betreuer Thomas Letsch
Experte Prof. Dr. Martin Zimmermann
Gegenleser Manuel Bauer

Abstract

An der Hochschule für Technik Rapperswil (HSR) werden in den Modulen *Algorithmen und Datenstrukturen 1 & 2* diverse Algorithmen unterrichtet, deren Funktionsweise schwierig nachzuvollziehen ist. Infolgedessen bekunden viele Studenten Schwierigkeiten beim Erlernen dieser abstrakten Konzepte. Abhilfe schafft der Graphs-Visualization-Service (GVS), welcher die Datenstrukturen *Graph* und *Tree* sowie ausgewählte darauf anwendbare Algorithmen visualisiert. Es hat sich gezeigt, dass die Visualisierung der Konzepte den Lernerfolg der Studenten positiv beeinflusst. Der GVS ist in seinem Funktionsumfang auf die genannten Datenstrukturen limitiert. Zudem wurde dessen Architektur nie für grössere Erweiterungen konzipiert. Ziel dieser Bachelorarbeit ist es, einen erweiterbaren Nachfolger zu entwickeln, der den GVS in Zukunft vollständig ablösen kann und weitere Datenstrukturen, wie beispielsweise eine Linked-List, in Form von Modulen unterstützt.

Unter Berücksichtigung des Funktionsumfangs des GVS wurden die Anforderungen an den neuen Algorithm & Data Structure Visualizer (ADV) spezifiziert. Zusätzlich lieferte eine Benutzerumfrage wertvolle Inputs, welche insbesondere auf die Benutzeroberfläche und das Feature-Set Einfluss hatten. Bei der Planung der Architektur lag der Fokus auf der Erweiterbarkeit der Software sowie einer intuitiven Benutzeroberfläche. Damit die Umsetzbarkeit der Designentscheide überprüft werden konnte, wurden diese bereits früh mit Prototypen verifiziert. Der ADV wurde in zwei Teilkomponenten aufgeteilt. Beim Benutzer wird eine Library eingebunden, welche Klassen zur Visualisierung von Datenstrukturen zur Verfügung stellt. Zudem wurde eine JavaFX Applikation entwickelt, welche sich um die Visualisierung kümmert. Zentraler Baustein der Visualisierungs-Komponente ist ein modulares Framework, das wiederverwendbare Widgets, wie Knoten oder Pfeile, zur Verwendung in den Modulen bereitstellt. Zur Unterstützung künftiger Erweiterungen der Software wurde ein umfassender Buildprozess spezifiziert, der die hohen Qualitätsstandards der Software auch in Zukunft verlässlich überprüfen soll.

Als Resultat dieser Bachelorarbeit entstand eine produktiv nutzbare Software, die komplexe Algorithmen anhand von schrittweise nachvollziehbaren Schnappschüssen einer Datenstruktur visualisiert. Das Framework ist für zukünftige Entwickler unter anderem durch den Einsatz des Strategy Pattern flexibel erweiterbar und dank einer durchgehend modularen Architektur klar gekapselt. Der Endanwender des ADV profitiert von einer intuitiven Benutzeroberfläche und kann sich so vollumfänglich auf die Lerninhalte fokussieren.

Management Summary

Ausgangslage

In den Unterrichts-Modulen *Algorithmen und Datenstrukturen 1 & 2* an der HSR werden verschiedene Datenstrukturen und Informatik-Konzepte vermittelt. Einigen Studenten fällt es schwer, diese Konzepte zu fassen und anzuwenden. Um den Lernerfolg der Studenten positiv zu beeinflussen, haben sich zwei Methoden bewährt. Einerseits müssen die Studenten die Algorithmen selbständig implementieren und andererseits lassen sich die abstrakten Konzepte durch Visualisierung einfacher aufnehmen.

Die bisher eingesetzte Software (GVS) ist auf die Visualisierung der Datenstrukturen *Graph* und *Tree* beschränkt. Dennoch demonstriert sie bereits gut, welche Funktionalitäten eine solche Lernsoftware bereitstellen sollte. Da die Architektur des GVS keine Erweiterungen von Datenstrukturen vorsieht, ist für die Abdeckung des ganzen Unterrichts-Stoffes eine neue Software nötig.

Vorgehen, Technologien

Die Anforderungen an den ADV werden durch verschiedene Einflüsse geprägt:

- Eine Nutzerumfrage unter Anwendern des GVS hat gezeigt, dass die Applikation zwar für den Lernerfolg hilfreich ist, die Usability und die *Feature Awareness* aber tief sind. Somit ergab sich für den ADV einen Schwerpunkt auf der Usability.
- Die vorhandenen Funktionalitäten des GVS UI sollten vom ADV UI ebenfalls unterstützt werden.
- Der Umfang des Frameworks wurde mit der Erstellung von Modulidee-Konzepten abgesteckt. Die Modulideen können zudem als Vorschläge für zukünftige Modulerweiterungen verwendet werden. Die Anforderungen dieser potentiellen Module beeinflussten die Anforderungen an den ADV.

In einer zweiten Phase wurde die Architektur und das Design der Applikation geplant. Der ADV besteht aus drei Containern: zwei Java Libraries und ein Java Framework mit JavaFX Benutzeroberfläche. Um die Software modular zu halten, wurde eine übersichtliche API realisiert. Die Core Components des Frameworks werden mit mehreren Modulen erweitert.

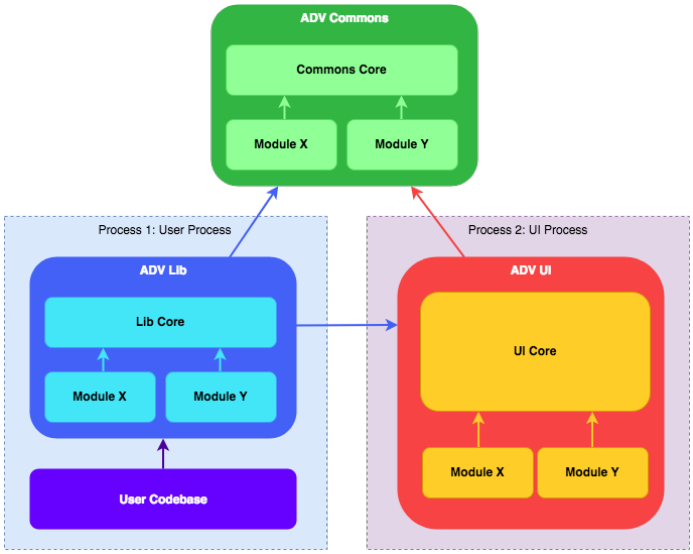


Abbildung 1: Übersicht der Container und Komponenten des ADV

Die umgesetzten Module wurden bewusst gewählt, um einen möglichst grosse Diversität abzudecken und die Grenzen des Frameworks zu testen. Umgesetzt wurden die Module Array, Graph, Queue und Stack.

Ergebnisse

Im Rahmen der vorliegenden Arbeit wurde eine modular erweiterbare Software entwickelt, welche zukünftig mit mehreren Modulen erweitert werden kann, wobei ein Modul jeweils eine Datenstruktur beinhaltet. Im Projektumfang sind ein UI Framework, zwei Libraries sowie vier Module enthalten.

Durch das Einführen der ADV Commons Library konnten Code Duplikationen weitestgehend vermieden werden, was zu einer besseren Wartbarkeit führt. Die Entwicklung neuer *Modules* wird durch einen automatisierten Build-Prozess unterstützt, welcher den Sourcecode auf Qualitätsattribute überprüft.

Die laut der Nutzerumfrage gewünschte Usability konnte durch die Durchführung von Usabilitytests erreicht werden.

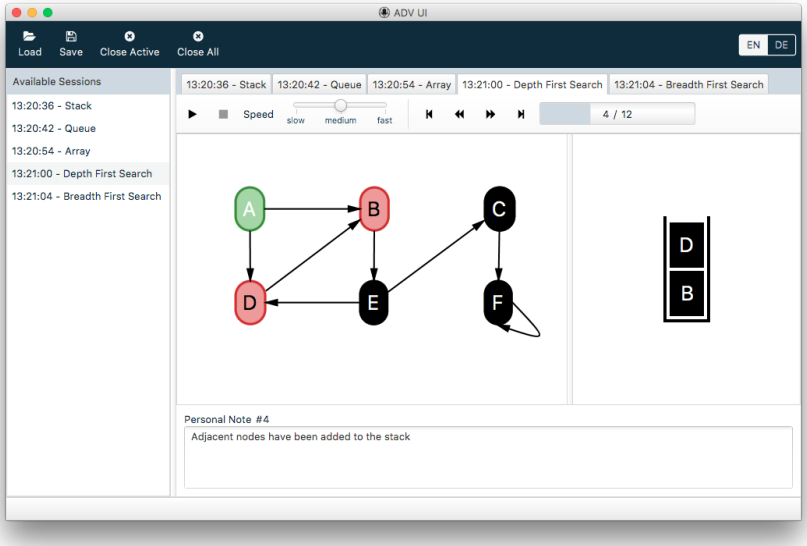


Abbildung 2: Finale Benutzeroberfläche des ADV UI

Ausblick

Der ADV ist speziell für Erweiterbarkeit entwickelt worden. Zukünftig können im Rahmen von Studienarbeiten weitere *Modules* programmiert und integriert werden. Für zukünftige Entwickler liegt eine Entwickler-Benutzeranleitung vor (siehe Anhang I).

Danksagungen

Wir danken folgenden Personen für Ihre Unterstützung während unserer Bachelorarbeit:

- Unserem Betreuer Thomas Letsch für die angenehme und unterstützende Zusammenarbeit während der gesamten Studien- und Bachelorarbeit
- Unserem Experten Prof. Dr. Martin Zimmermann
- Unserem Gegenleser Manuel Bauer
- Unseren fünf Testpersonen, die an den Usability Tests teilgenommen haben
- Stefano Trentini für die wertvollen Inputs beim Korrekturlesen

Inhaltsverzeichnis

Glossar und Abkürzungsverzeichnis

ADV Algorithm & Data Structure Visualizer. i, ii, 2, 3, 5, 8, 16, 17, 43–45, 47, 49, 51, 53, 55, 59, 63, 75–78, 116, 141, 142, 144, 151, *siehe* Algorithm & Data Structure Visualizer

ADV Commons Library die von der ADV Lib und dem ADV UI verwendet wird und die Gemeinsamkeiten der beiden Container kapselt. 43, 49, 54, 112, 142, 143, 151, 155, 158

ADV Lib API welche alle unterstützten Datenstrukturen beinhaltet, die im ADV UI dargestellt werden können. Die Library muss im Projekt des Studenten eingebunden werden. Sie ist für die Übertragung der Daten an das ADV UI zuständig. xvi, 16, 33, 43, 48, 49, 53, 54, 59, 62, 64, 76, 78, 112, 116, 117, 128, 131, 134, 137, 142, 143, 151, 155, 158, 172

ADV UI Eigenständig laufende JavaFX Applikation welche die Daten von der ADV Lib empfängt und visualisiert. xvi, 3, 16, 39, 43, 45, 48, 49, 53–57, 59, 60, 64, 76, 78, 112, 115–117, 125, 142, 143, 147, 151, 155, 158, 167, 172

Algorithm & Data Structure Visualizer Produkttitel der vorliegenden Bachelorarbeit. i

API Application Programming Interface. 17, 28, *siehe* Application Programming Interface

Bellman Ford Algorithmus zum Finden des kürzesten Pfades von einem Start Node zu einem End Node innerhalb eines Graphen. Im Gegensatz zum Dijkstra erlaubt dieser Algorithmus auch negativ gewichtete Kanten. 23

BFS Breadth First Search. 23, 31, 65, *siehe* Breadth First Search

Breadth First Search Algorithmus zum Suchen von Knoten in einem Graphen. Die Reihenfolge der zu besuchenden Knoten ist durch eine FIFO Queue gegeben [**bfs**]. 23

C4 Model Model zur Darstellung und Kommunikation von Software Architektur. Umfasst 4 Modelle mit unterschiedlichen Detaillierungsgraden: Context, Container, Component, Class. 41

Code Behind Das Code Behind entspricht der Controller Klasse, welche die Logik einer FXML (JavaFX XML)-View enthält. Der Begriff wird im Model View ViewModel (MVVM) Pattern verwendet und ist besonders bei Microsoft Technologien (ASP.NET, WPF) verbreitet. 48

Component Logischer Baustein, der innerhalb eines Containers lebt. Siehe C4 Model. ii, 12, 46, 68

Container Ein Container ist eine unabhängig laufende Einheit, welche Code ausführt oder Daten speichert wie beispielsweise eine Web Applikation, eine Mobile App oder eine Datenbank. Siehe C4 Model. ii, xiii, 4, 12, 43, 46, 47, 49, 54, 68, 142

Continuous Integration CI ist zuständig für das fortlaufendes Zusammenführen von Softwarebausteinen sowie die automatische Ausführung des Builds und der Tests. 5, 67, 75

DAG Directed Acyclic Graph. 32, *siehe* Directed Acyclic Graph

Dependency Injection Abhängigkeiten von Objekten werden zum Zeitpunkt der Instanzierung übergeben und nicht vom Objekt selber instanziert. Dies führt zu tieferer Kopplung und besserer Testbarkeit. 47

Depth First Search Algorithmus zum Suchen von Knoten in einem Graphen. Die Reihenfolge der zu besuchenden Knoten ist durch einen Stack gegeben [**dfs**]. 22

DFS Depth First Search. 22, 31, 65, 137, *siehe* Depth First Search

DI Dependency Injection. 47, 61, *siehe* Dependency Injection

Dijkstra Algorithmus zum Finden des kürzesten Pfades von einem Start Node zu einem End Node innerhalb eines Graphen mit positiv gewichteten Kanten [**dijkstra**]. 23, 137

Directed Acyclic Graph Gerichteter Graph ohne Zyklen welcher zudem über eine topologische Ordnung verfügt. 32

Element Element, welches vom ADV UI dargestellt wird. Zum Beispiel die Nodes eines Graphen. 28

Extensible Markup Language Markup Language zur plattformunabhängigen Übertragung von hierarchisch strukturierten Daten. Die Struktur wird durch ein Schema beschrieben. 51

FIFO First in First Out. 23, 33, *siehe* First in First Out

First in First Out Elemente werden in genau dieser Reihenfolge von der Datenstruktur entfernt, wie sie auch hinzugefügt werden. Ein klassischer Anwendungsfall ist die Datenstruktur Queue. 23

Framework Ein Framework legt fest, wie Klasseninstanzen zur Laufzeit miteinander zusammenarbeiten dürfen. Es implementiert das Hollywood Principle rsp. Inversion of Control. 59

FXML XML basierte UI Markup Language für JavaFX Programme. 48

GNU Privacy Guard Kryptographie-Suite zum Ver- und Entschlüsseln von Files sowie zum Erzeugen und Prüfen von elektronischer Signaturen. 152

GPG GNU Privacy Guard. 152, *siehe* GNU Privacy Guard

Gradle Groovy basiertes Build System. 115, 143, 144

Graphs-Visualization-Service Diplomarbeit (GVSv1 [2]) und Studienarbeit (GVSv2 [9]) welche als Grundlage dieser Bachelorarbeit dient. i

GUI Graphical User Interface. 12, 33, 37, 39, 55–57, 63, 68, 116, 124, 159, 169, *siehe* Graphical User Interface

Guice Dependency Injection (DI) Framework von Google [34]. 61

GVS Graphs-Visualization-Service. i, ii, 2, 3, 8, 15–17, 56, 92, *siehe* Graphs-Visualization-Service

Hochschule für Technik Rapperswil Fachhochschule in Rapperswil an welcher die Bachelorarbeit durchgeführt wird. i

HSR Hochschule für Technik Rapperswil. i, ii, 2, 44, 51, 55, 63, *siehe* Hochschule für Technik Rapperswil

i18n Internationalization. 166, 167, *siehe* Internationalization

IDE Integrated Development Environment. 71, 147, *siehe* Integrated Development Environment

Integrated Development Environment Tools zur effizienten Entwicklung von Software Produkten. Integriert unter Anderem einen Compiler sowie nützliche Refactoring Werkzeuge. 71

Internationalization Prozess, welcher Software in verschiedenen Sprachen nutzbar macht. 166

Interprozesskommunikation Verschiedene Verfahren des Informationsaustausches zwischen den Prozessen eines Systems. 44

IPC Interprozesskommunikation. 44, 49, 51, *siehe* Interprozesskommunikation

JAR Java Archive. 43, 71, *siehe* Java Archive

Java Archive Container für ausführbare Java Files. 43

JavaFX JavaFX ist ein Framework zur Erstellung plattformübergreifender Java-Applikationen. ii, 3, 6, 47, 48, 55, 56, 60, 62, 78, 169

JavaScript Object Notation Kompaktes, menschen-lesbares Datenformat für den Datenaustauschs zwischen Anwendungen. 51

JSON JavaScript Object Notation. 51, 78, *siehe* JavaScript Object Notation

Jukito Testing Framework. Vereint Guice und Mockito. 169

Last in First Out Elemente werden in umgekehrter Reihenfolge von der Datenstruktur entfernt, als das sie zuvor hinzugefügt wurden. Ein klassischer Anwendungsfall ist die Datenstruktur Stack. 21

LIFO Last in First Out. 21, 22, *siehe* Last in First Out

Model View Controller Verbreitetes Softwaremuster zur Trennung der Aufgabenbereiche von Software Komponenten die das UI steuern. 47

Model View ViewModel Entwurfsmuster für die Darstellung von modernen UI-Plattformen mittels Databinding. MVVM ist eine Variante von Model View Controller (MVC). 47

Module Ein Modul erweitert die Funktionalität des Core Frameworks oder der Core Library. Es umfasst eine spezifische Datenstrukturen. 2, 17, 18, 27, 29, 30, 38, 39, 45, 53, 59, 60, 62–65, 69, 77, 125, 138, 142, 143, 147, 158

ModuleGroup Kapselt den modul-spezifischen Inhalt einer Übertragung. Ein Snapshot kann eine oder mehrere *ModuleGroups* enthalten. 53, 62, 64

MVC Model View Controller. 47, *siehe* Model View Controller

MVVM Model View ViewModel. 47, *siehe* Model View ViewModel

NFR Non-functional requirement. 11, 38, 68, *siehe* Non-functional requirement

Non-functional requirement Anforderungen an eine Software, welche die qualitativen Eigenschaften eines Systems beschreiben. 11

Plain Old Java Object Triviales Java Objekt mit wenigen oder keinen externen Abhängigkeiten. 39

POJO Plain Old Java Object. 39, 48, *siehe* Plain Old Java Object

Property Change Event Event welcher von einer Source ausgelöst wird, wenn sich ihre Daten ändern. Alle registrierten Subscriptions werden durch das Event über die Änderungen informiert. Diese Mechanik löst die gut bekannten Observer-Observable Klassen ab, welche in Java 9 *deprecated* sind. 47, 63

Rational Unified Process Iteratives Vorgehensmodell für die Entwicklung von Software, dass in Phasen organisiert ist. 67

Reflection Das Reflection-API dient dazu, die Bestandteile von Klassen (Datenelemente, Konstruktoren und Methoden) dynamisch zur Laufzeit zu analysieren und zu benutzen. 5, 24, 61

RUP Rational Unified Process. 67, *siehe* Rational Unified Process

SCRUM Verbreitetes Vorgehensmodell zur agilen Softwareentwicklung. Umfasst ein itertives Vorgehen mit stetigem Kundenfeedback. 67

Session Eine ADV Session umfasst ein bis mehrere Snapshots die zwischen dem ADV Lib und ADV UI übertragen werden. 3, 33, 49, 52–54, 56, 60

Snapshot Ein Snapshot repräsentiert den Status einer Datenstruktur aus der User Codebase zu einem bestimmten Zeitpunkt. Er enthält alle Informationen die zur Darstellung notwendig sind. Ein Snapshot kann ein oder mehrere ModuleGroups enthalten. 33, 52, 53, 56, 58, 60

Style Ein Style dient der Darstellung von Algorithmen. Er kann auf Elements und Relations angewendet werden. Umfasst folgende Attribute: Hintergrundfarbe, Randfarbe, Randbreite, Randstil (durchgezogen, gestrichelt, ...). 28, 31, 125, 162

Tangle Abhängigkeiten von Packages aus tiefen Schichten auf Packages in höher gelegenen Schichten. Tangles können mit dem Architektur-Werkzeug *Structure101* erkannt und beispielsweise mit dem Observer Pattern behoben werden. 147

Ubiquitous Language Entwickler und Anwender einer Applikation nutzen das gleiche Vokabular. Damit können Missverständnisse auf ein Minimum reduziert werden, da alle Projektmitglieder Dinge mit den selben

Namen benennen. Die Ubiquitous Language wurde von Eric Evans in *Domain Driven Design* eingeführt. 12, 68

User Codebase Programmcode der vom ADV User geschrieben wird. Die Usercodebase erweitert die API-Klassen aus der ADV Lib. 33, 53, 55, 60, 64

Widget Ein Widget ist eine wiederverwendbare grafische Komponente. 63

Windows Presentation Foundation Klassenbibliothek zur Gestaltung von grafischen Benutzeroberflächen in .NET. 47

WPF Windows Presentation Foundation. 47, *siehe* Windows Presentation Foundation

XML Extensible Markup Language. 51, *siehe* Extensible Markup Language

YAML YAML Ain't Markup Language. 51, *siehe* YAML Ain't Markup Language

YAML Ain't Markup Language Markup Language zur Datenserialisierung (ursprünglich Yet Another Markup Language). 51

Abbildungsverzeichnis

Tabellenverzeichnis

Teil I

Technischer Bericht

Kapitel 1

Technischer Bericht

1.1 Ausgangslage und Problembeschreibung

Im Rahmen dieser Bachelorarbeit soll eine Software entwickelt werden, welche Algorithmen und Datenstrukturen visualisieren kann. Grundlage dafür bieten zwei Vorgängerarbeiten, welche im Rahmen einer Diplomarbeit und Studienarbeit unter dem Projekttitel GVS durchgeführt wurden. Der GVS wird erfolgreich und aktiv im Unterricht der HSR eingesetzt. Er ist jedoch auf die Visualisierung von Graphen und Trees beschränkt und seine Architektur sieht keine Erweiterungen für weitere Datenstrukturen vor.

Mit dem Ziel, ein modulares Framework zu entwickeln, wurden die Funktionalitäten des GVS analysiert und eine Lösung kreiert, welche den GVS in Zukunft vollständig ablösen soll. Das Hauptaugenmerk lag dabei auf einer verbesserten Benutzbarkeit für den Endanwender sowie einer gut dokumentierten Schnittstelle für jene Entwickler, die den ADV zukünftig erweitern werden.

Laut der Aufgabenstellung sind im Umfang dieses Projektes eine erweiterbare Software zur Visualisierung von Datenstrukturen sowie mindestens zwei implementierte Erweiterungen, sogenannte Modules, enthalten.

Die Software soll den Lernerfolg der Studenten positiv beeinflussen. Zukünftig soll ein möglichst breites Angebot an Modulen zur Verfügung gestellt werden, damit sich ein Student nur mit einer Software auseinandersetzen muss und sich so vollumfänglich auf die Lerninhalte konzentrieren kann.

1.1.1 System Kontext

Die Software wird den Studenten als Lernsoftware zur Verfügung gestellt. Anstatt vorgefertigte Algorithmen darzustellen, visualisiert der ADV im Gegensatz zu vergleichbaren Konkurrenzprodukten den Programmcode des

Studenten. Programmiert der Student einen Fehler, kann er diesen daher auch visuell nachvollziehen.

Sämtliche Artefakte des ADV sind in *Java* programmiert. Für die Erstellung der Desktop-Anwendung wurde JavaFX verwendet. JavaFX ist die Standardlösung zur Erstellung von grafischen Benutzeroberflächen unter Java.

1.2 Lösungskonzept

Zu Beginn des Projekts wurden die Nutzer des GVS nach ihrer Meinung zu der Benutzbarkeit des GVS befragt. Zusammen mit den Anforderungen an ein erweiterbares Framework, bilden diese Inputs die Funktionalen Requirements an den ADV. Der Fokus lag dabei insbesondere bei der Benutzbarkeit der Applikation, um den Lernerfolg des Studenten nicht zu behindern.

Da der Umfang des Unterrichtstoffes bereits klar spezifiziert ist, konnte gut abgeschätzt werden, welche Datenstrukturen in Zukunft im ADV integriert werden sollen. Die Software musste also so designet werden, dass alle zur Zeit dieser Arbeit bekannten Modulideen im Framework umsetzbar sind.

Das Lösungskonzept sah vor, dass sich die Applikation in zwei unabhängige Container aufteilt. Der erste Container umfasst die Library, welche im Code des Studenten eingebunden wird. Der zweite Container kümmert sich um die Visualisierung der Klassen aus der Library mittels JavaFX. Sobald der Student seine Applikation ausführt, werden alle Momentaufnahmen der Datenstruktur in einer Session gekapselt und an das ADV UI übermittelt. Die Visualisierung der Datenstruktur besteht dann aus wenigen, einfach verständlichen Schritten. Das Framework gibt diese Schritte vor und delegiert ausgewählte Aufgaben an die Module.

Abbildung 1.1: Teilschritte zur Visualisierung einer Datenstruktur

Um den Installationsaufwand für den Studenten zu vereinfachen, werden beide Container in einem öffentlichen Maven-Repository zur Verfügung gestellt.

1.3 Umsetzung

Für die ersten Prototypen wurden vergleichbare Produkte analysiert, um eine Grundidee für die gänzlich neue Aufgabe, der Entwicklung eines Frameworks, zu erhalten. Dies hat den Einstieg bestimmt vereinfacht, schlussendlich wurde aber eine komplett eigene Lösung umgesetzt.

Während der Entwicklung der Module offenbarten sich weitere Anforderungen an das Framework. In der endgültigen Version sind anstatt einem einzelnen Modul auch mehrere Module parallel darstellbar. Somit erhält der Student auch Einblicke in die verwendeten Hilfsdatenstrukturen eines Algorithmus. Darüber hinaus wurden die Gemeinsamkeiten der beiden Container in einem Commons-Projekt vereint. Damit konnten Code-Duplikationen weitestgehend vermieden und die Wartbarkeit der Software verbessert werden. Zudem vereinfacht die endgültige Version die Integration neuer Module, indem annotierte Modul-Services automatisch im Framework registriert werden.

Ein grosser Kritikpunkt des GVS war die Benutzbarkeit der Software und die *Feature Awareness* der Nutzer. Um diesem Problem entgegenzuwirken wurden in mehreren Projektstadien Usability-Tests durchgeführt, welche die Nutzerfreundlichkeit der Applikation überprüften. Die Erkenntnisse aus diesen Tests, wie beispielsweise das Bedürfnisse für Key Shortcuts, fliessen in das Endprodukt ein.

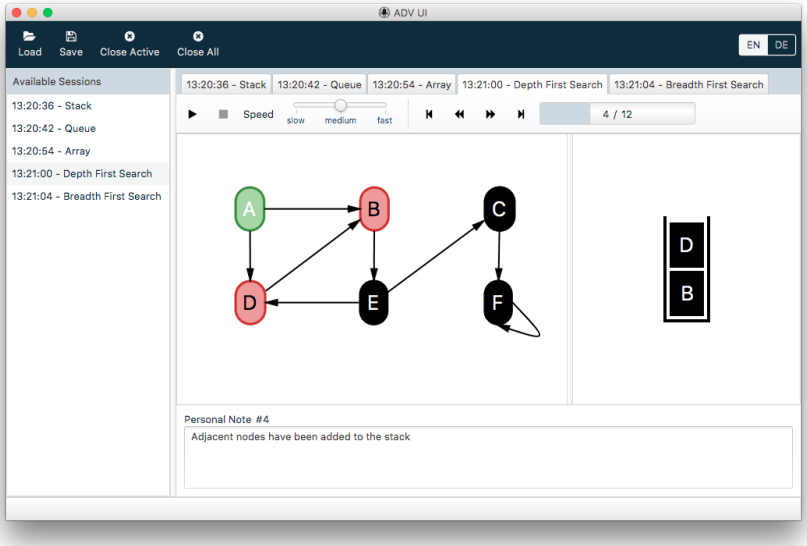


Abbildung 1.2: Benutzeroberfläche des ADV UI

Um die Qualität des Sourcecodes hoch zu halten, wurden eine Reihe an Qualitätsprüfungen eingeführt. Codefragmente werden einer statischen Codeanalyse unterzogen und auf die Konformität mit den Coding Richtlinien überprüft. Zusätzlich existieren für die wichtigen Code-Abschnitte Unit Tests. Änderungen an der Software werden automatisch auf einem CI (Continuous Integration) Server gebildet und publiziert.

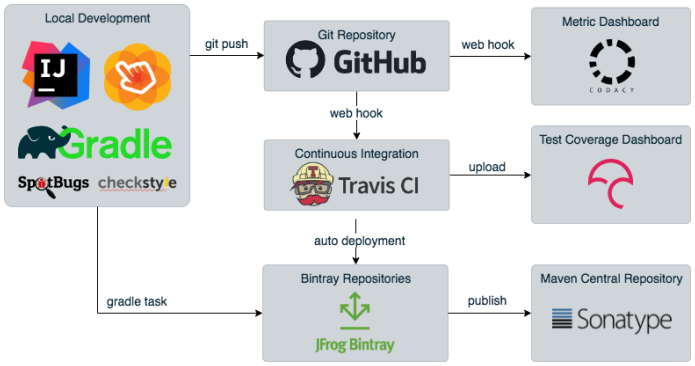


Abbildung 1.3: Übersicht des Continuous Integration und den Qualitätsmassnahmen

1.4 Ergebnisdiskussion

Im Rahmen dieser Arbeit wurde eine funktionsfähige, modular erweiterbare Software entwickelt, die im Unterricht an der HSR eingesetzt werden kann. Die gesetzten Ziele konnten erreicht und in bestimmten Bereichen sogar übertroffen werden. So wurden statt den geforderten 2 Modulen, 4 Module umgesetzt.

Der ADV unterstützt den Studenten aktiv beim Erlernen von schwierig zu verstehenden Konzepten und beeinflusst dessen Lernerfolg positiv. Die Benutzeroberfläche des ADV wurde stark durch die durchgeführten Usability-Tests beeinflusst. Die wichtigsten Funktionalitäten sind auch ohne Zuhilfenahme des Benutzermanuals ersichtlich und nutzbar.

Auch die technologischen Aspekte wurden nicht ausser Acht gelassen. Durch den Einsatz von Annotations und Reflection können neue Module dynamisch ausgelesen und im Framework geladen werden. Obschon der grundlegende Ablauf durch das Framework vorgegeben wird, ist der Modulentwickler aufgrund mehreren Designentscheiden relativ frei, seine Module zu programmieren. Einerseits wird das schemalose Dateiformat *JSON* zur Beschreibung der übertragenen Daten verwendet und andererseits bietet das Framework Events an, auf welche sich interessierte Module registrieren können.

Des weiteren profitiert der Modulentwickler von wiederverwendbaren Widget-Komponenten, die für eine Vielzahl an Anwendungsfällen verwendet werden können.

Aufgrund des umfangreichen Buildprozesses werden Modulentwickler gezwungen, die hohen Qualitätsstandards auch in zukünftigen Modulen weiter zu pflegen. Unterstützt wird der Entwickler durch eine klare Schichten-Architektur und eine ausführliche Dokumentation, die den Einstieg in das Framework erleichtern soll.

Für die Nutzung der Library im Studenten-Code konnten keine Usability-Tests durchgeführt werden, da diese Test-Durchführungen deutlich aufwändiger wären als die durchgeführten Usability Tests des UIs.

Dank Code-Reviews und Refactorings wurde die Qualität der Software laufend überprüft und beide Teammitglieder profitierten von einem Wissensaustausch.

Das Projektteam ist mit dem Endergebnis sehr zufrieden. Trotzdem sieht das Projektteam noch Potential in der Benutzbarkeit der Library, da der Fokus mehrheitlich auf das Framework sowie die Benutzeroberfläche gesetzt wurde.

1.4.1 Eingetretene Risiken

Von den antizipierten Risiken aus der Risikoanalyse (siehe 4.9), trat einzig das Risiko 3 ein, da der Testcoverage Uploader des Metrik Dashboards Codacy [19] keinen Support für die eingesetzte Java Version hatte. Dieses Problem konnte aber, gemäss geplanter Massnahmen, durch die Nutzung eines alternativen Testabdeckung-Dashboards behoben werden.

Generell mussten immer wieder kleinere Probleme mit Java und JavaFX gelöst werden. So gab es bei JavaFX im Rahmen des Projekts Jigsaw [54] (Modularisierung in Java 9) und JEP-253 [53] insbesondere bei der Sichtbarkeit von JavaFX-Internas Änderungen. Zudem wurden die weit verbreiteten Java Observer/Observable Klassen als überholt annotiert. Bei der Entwicklung von Widgets kam es zu Problemen, da bestimmte Change-Listener nicht wie erwartet aufgerufen wurden. Dies könnte an der verfrühten Garbage Collection liegen, da JavaFX intern mit Weak-Listener arbeitet [67]. Nach einem grösseren Zeitaufwand konnte das Problem schlussendlich mit entsprechenden Workarounds behoben werden.

1.4.2 Offene Features

Einige tief priorisierte User Stories aus den Funktionalen Anforderungen (siehe 2.9.1) konnten im Scope dieses Projektes nicht umgesetzt werden. Bei diesen User Stories handelt es sich nicht um Kern-Features der Applikation, sondern um Features, welche die Usability verbessern würden.

- US9: Theme
- US13: Tutorial
- US23: Modulbeschreibung

Ausserdem sind während dem Entwicklungsprozess weitere Feature-Ideen aufgetaucht, welche in zukünftigen Arbeiten umgesetzt werden könnten:

Framework Features

Kontext Menü
Bei einem Rechtsklick auf die Session Liste oder auf ein Tab erscheint ein Kontext Menü, welches Funktionen wie das Speichern und Schliessen der Session oder das Öffnen des Tabs in einem neuen Fenster ermöglicht.

Closing Tabs
Abgetrennte Tabs können per Drag-and-Drop wieder ins Hauptfenster eingebunden werden (vgl. Funktionalität von Browser Tabs).

Drag Support
Das Verschieben von UI Elementen ist eine Modul-übergreifende Anforderung. Das Framework sollte daher eine Util Klasse anbieten, welche Drag Support ermöglicht.

Modal Support
Das Framework zeigt für bestimmte User-Interaktionen ein Modal oder Overlay an. Somit wird der User zum Beispiel beim Löschen einer Session aufgefordert, die Aktion zu bestätigen.

Konfiguration der Styles über das UI
Die vorgefertigten Styles sowie die Farbe der Labels von Nodes und Edges können über einen Einstellungs-Button im UI verändert werden.

Verbessertes Öffnen von Files
*.adv Files sollen per Drag-and-Drop in das ADV UI oder per Doppelklick im File Explorer geöffnet werden können.

***Module* Features**

Graph Layouting Algorithm
Nicht fix positionierte Elemente sollen mit einem Layouting Algorithmus im *Graph Module* ansprechend positioniert werden.

Array Index
Für ein besseres Verständnis zeigt das *Array Module* zu allen Inhalten den entsprechenden Array Index an (siehe Abbildung 1.4).

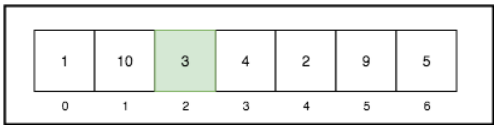


Abbildung 1.4: Array Module mit Indizes

Anordnung *Child Modules*
Je nach Anwendungsfall macht eine unterschiedliche Anordnung der *Child Modules* Sinn. Durch ein Flag soll diese konfigurierbar sein (z.B. "grid", "vertical", "horizontal").

1.4.3 Ausblick

In Zukunft werden diverse Erweiterungen, insbesondere in der Form von weiteren Modulen erwartet. Es wird sich zeigen, ob die angedachte Architektur des Frameworks, neuen innovativen Ideen standhalten wird. Das Projektteam ist überzeugt, eine skalierbare Grundlage geschaffen zu haben, die im Zweifelsfall auch einfach erweitert werden kann. Zusammen mit der umfassenden Dokumentation und dem stark spezifizierten Entwicklungsprozess sollte die Qualität des Framework auch keinen Schaden nehmen.

Als nächster Schritt soll insbesondere der Umfang der Module erweitert, sowie die Benutzbarkeit der API verbessert werden. Wünschenswert wäre die Implementation eines *Tree Modules*, damit der GVS vollumfänglich durch den ADV abgelöst werden kann.

Teil II

Projektdokumentation

Kapitel 2

Anforderungsspezifikation

Datum	Version	Änderungen	Autor
22.02.2018	0.01	Grundsätze und Terminologie beschrieben	mwieland
23.02.2018	0.02	Stakeholder beschrieben	mtrentini
26.02.2018	0.03	User Stories beschrieben	mtrentini
26.02.2018	0.04	Ubiquitous Language beschrieben	mtrentini
27.02.2018	0.05	Referenzarchikturen beschrieben	mwieland
27.02.2018	0.06	Technische Anforderungen beschrieben	mwieland
27.02.2018	0.07	NFR beschrieben	mwieland
02.03.2018	0.08	Modulideen Array und Linked List beschrieben	mtrentini
05.03.2018	0.09	User Analyse beschrieben	mtrentini
05.03.2018	0.1	Modulideen und Ubiquitous Language erweitert	mwieland
04.04.2018	0.2	Review Dokument	mwieland, mtrentini
29.05.2018	0.21	Anpassung Ubiquitous Language, Abbildung Session	mtrentini
11.06.2018	1.0	Review Dokument	mwieland, mtrenini

Tabelle 2.1: Versionshistory Anforderungsspezifikation

2.1 Einleitung

Das folgende Kapitel steckt den Systemkontext ab und definiert die funktionalen Anforderungen, die sich von den Bedürfnissen der Stakeholder ableiten lassen. Die Non-functional requirements (NFRs) geben einen Überblick über die Qualitätsanforderungen der Applikation. Eine allgegenwärtige Domänensprache wird im Abschnitt 2.3 definiert.

Die Anforderungsspezifikation soll während dem Projektverlauf periodisch konsultiert werden, um sicherzustellen, dass das richtige System entwickelt wird.

2.2 Terminologie

Um den vorliegenden Auftrag besser bewerten zu können, muss zuerst Klarheit über die Terminologie geschaffen werden.

Library Eine Library ist eine Sammlung von Code, die eine Reihe von verwandten Funktionen zur Verfügung stellt.

Framework Ein Framework legt fest, wie Klasseninstanzen zur Laufzeit miteinander zusammenarbeiten dürfen. Es bietet wiederverwendbare Klassen mit wohldefinierten Schnittstellen. Ein Framework fungiert als Gerüst, das bestimmte Implementationsentscheide den Subklassen überlässt. Die wichtigsten Designentscheide werden im Framework gekapselt, was zu einer besseren Wartbarkeit führt. Ein Framework wächst über die Zeit und wird über mehrere Zyklen verfeinert [1].

Inversion of Control Einer der Hauptunterschiede zwischen Frameworks und Libraries ist die Inversion of Control.
Eine Library umfasst eine Reihe von Funktionen, die aufgerufen werden können. Jeder Aufruf erledigt eine bestimmte Aufgabe und gibt die Kontrolle wieder an den Client zurück.
Im Gegensatz dazu, bleibt die Kontrolle über den Programmfluss bei einem Framework. Dieses ruft zu gegebener Zeit die Implementierungen der Applikation auf und bekommt die Kontrolle danach wieder zurück. Dieses Verhalten ist auch unter dem Namen «Hollywood Principle» bekannt. [25]

Wie in der Abbildung 2.1 gut zu erkennen ist, kann das Framework nur zusammen mit der Applikation genutzt werden, wohingegen eine Library ausserhalb des Applikations-Kontexts definiert ist.

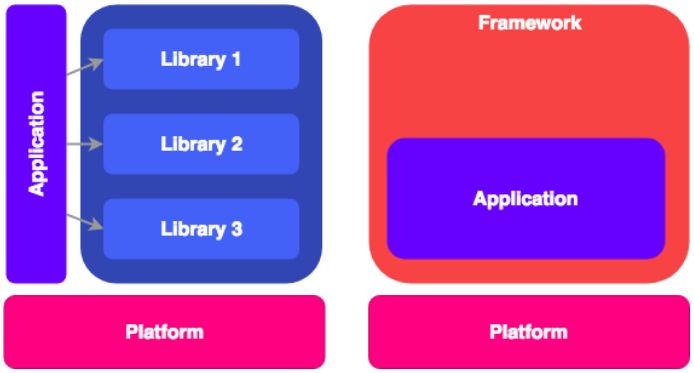


Abbildung 2.1: Vergleich Library und Framework

2.3 Ubiquitous Language

Damit sich das gesamte Projektteam klarer verständigen kann, sind in der Tabelle 2.2 jene Begriffe definiert, die im vorliegenden Kontext verwendet werden (siehe Ubiquitous Language). Alle Begriffe sind zur besseren Integration in den Source Code in Englisch gehalten.

Container	
ADV UI	Umfasst das GUI. Empfängt Daten von der *ADV Lib* und stellt diese dar.
ADV Lib	Library die im Projekt des *ADV Lib Users* eingebunden wird. Sendet Daten an das *ADV UI*.
ADV Commons	Library die gemeinsame Klassen der *ADV Lib* und des *ADV UI* kapselt und in beiden Projekten eingebunden ist.
User Codebase	Programmcode der vom *ADV Lib User* geschrieben wird und die Klassen aus der *ADV Lib* verwendet.
Components	
UI Core	Core Framework Komponente, welche den Kontrollfluss für die Visualisierung definiert.
Lib Core	Enthält generische Klassen der *ADV Lib* (z.B Socket Verbindung)
Commons Core	Kapselt von Lib Core und UI Core gemeinsam verwendete Klassen (z.B. Styles).
Module	Erweitert die Cores um modulspezifische Klassen und Funktionalitäten.

Classes/Elements	
Session	Gruppiert mehrere Snapshots
Snapshot	Repräsentiert den Status einer Datenstruktur in der *User Codebase* zu einem bestimmten Zeitpunkt. Wird im *ADV UI* dargestellt.
ModuleGroup	Gruppiert modul-spezifische Elemente innerhalb eines Snapshots
Element	Element, welches vom ADV UI dargestellt wird
Relation	Definiert Abhängigkeiten, Zugehörigkeiten oder Ordnungen zwischen mehreren Elementen.
Style	Kann auf Elemente und Relationen angewendet werden. Umfasst folgende Attribute: Hintergrundfarbe, Randfarbe, Randbreite, Randstil (durchgezogen, gestrichelt, ...)
Funktionalitäten	
Stepping, *Steppen*	Bezeichnet das Navigieren durch die Snapshots einer Session
Replay	Automatisches steppen durch alle Snapshots einer Session.

Tabelle 2.2: Ubiquitous Language

Die Abbildungen 2.3 und 2.2 geben einen Überblick über die Begriffe der Ubiquitous Language und wie sie zusammenhängen. Hierbei handelt es sich nicht um eine Architekturübersicht. Detaillierte Kontext-, und Containerdiagramme sind der *Architektur und Design Spezifikation* im Kapitel 3 zu entnehmen.

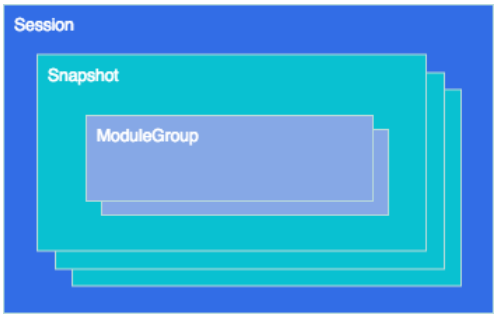


Abbildung 2.2: Verschachtlung von Session, Snapshot und ModuleGroup

Abbildung 2.3: Zuordnung der Begriffe aus der Ubiquitous Language zu den Containern

2.4 Analyse: Konkurrenzprodukte und vergleichbare Architekturen

Der Entwurf von skalierbaren Frameworks benötigt viel Erfahrung. Damit Anfängerfehler minimiert werden können, werden vergleichbare Architekturen analysiert, die sich in diesem Feld durchgesetzt haben. Die Tabelle 2.3 bietet einen Überblick über Produkte, die ähnliche Aufgaben erledigen oder eine Architektur implementieren die zur Inspiration verwendet werden kann. Somit kann von den «Lessons Learned» der Konkurrenzprodukte profitiert werden.

Produkt	Beschreibung
GVS	Visualisiert vom User programmierte Graphen und Trees. Mit Styles können Algorithmen auf diesen Strukturen abgebildet werden.

Algorithm Visualizer [11]	Visualisiert eine Vielzahl an Algorithm und Datenstrukturen im Browser. Bietet ein HTML/JS Interface um den Source Code zu manipulieren.
VisuAlgo [69]	Visualisiert eine Vielzahl an Algorithmen und Datenstrukturen. Bietet einfache Interaktionsmöglichkeiten ohne den Sourcecode direkt verändern zu können.
Data Structure Visualizations [49]	Visualisiert eine grosse Anzahl an Algorithmen und Datenstrukturen ohne spezielle Interaktionsmöglichkeiten.
GraphStream [32]	Modulare OpenSource Library mit vielen Extensions zum Darstellen von Algorithmen aus der Graphentheorie.
JUNG [43]	OpenSource Library zum Darstellen von Algorithmen aus der Graphentheorie. Spezialisiert auf sehr grosse Graphen.
FXGL [26]	JavaFX Superset zum einfachen Erstellen von 2D Games

Tabelle 2.3: Referenzarchitekturen und Konkurrenzprodukte

Für die Entwicklung des Prototyps fliessen hauptsächlich Erkenntnisse aus dem GVS und dem FXGL Projekt ein. Bei FXGL wird die Schnittstelle zwischen Framework und Client Code analysiert. Beim GVS wird hauptsächlich der Funktionsumfang untersucht. Die restlichen Konkurrenzprodukte liefern Inspirationen für die Modulideen.

2.5 Aktoren

Aktoren sind Benutzer, welche die Applikation direkt verwenden. Neben den Aktoren gibt es weitere Interessengruppen die Einfluss auf die Software haben. Diese sind im Abschnitt 2.6 genauer beschrieben.

Der Aktor *ADV-User* kann zwei verschiedene Rollen einnehmen, wobei auch beide Rollen gleichzeitig eingenommen werden können.

#	Rolle	Beschreibung	Stakeholder
1	ADV UI User	Nutzt nur das ADV UI	Dozent
2	ADV Lib User	Implementiert die Interfaces der ADV Lib in der *User Codebase*	Student

Tabelle 2.4: Relevante Aktoren

2.6 Stakeholder

Stakeholder repräsentieren Personen, die ein Interesse an der vorliegenden Software haben (Anspruchsgruppen/Interessensgruppen). Für den ADV können drei relevante Stakeholder ermittelt werden. Während der Umsetzung wird überprüft, ob die Erwartungen der Stakeholder erfüllt werden können.

#	Rolle	Ziel/Intention	Erwartungshaltung
1	Student	Möchte den ADV unter Zuhilfenahme der Benutzeranleitung unkompliziert installieren und benutzen können.	Möchte seinen Code durch die verschiedenen Modules dargestellt bekommen. Der Lerneffekt steht im Vordergrund.
2	Modulentwickler	Möchte mit Hilfe von gut dokumentierten APIs weitere Modules für den ADV entwickeln.	Erwartet eine kohärente Umsetzung des ADV und der Dokumentation. Verbreitete Software Patterns sind im Source Code zu erkennen.
3	Dozent	Möchte den ADV im Unterricht einsetzen, um die Unterrichtsinhalte zu veranschaulichen.	Erwartet eine intuitive Integration des ADV in bestehende Übungsaufgaben.

Tabelle 2.5: Relevante Stakeholder

2.7 Benutzer-Umfrage

Die vorliegende Bachelorarbeit ist eine Folgearbeit des *GVS 1.0* [2] und *GVS 2.0* [9]. Die Benutzer dieser beiden Applikationen sind vergleichbar mit den

zukünftigen Anwendern des ADV. Um das Nutzungsverhalten und die Bedürfnisse der Benutzer besser verstehen zu können, wird eine Umfrage an die Studenten des unteren Semesters versendet. Die Ergebnisse der Umfrage sind dem Anhang E zu entnehmen.

Zusammenfassend können aus der Umfrage folgende Erkenntnisse gewonnen werden:

- Einer klaren Mehrheit der Studenten hilft der GVS, die vermittelten Konzepte besser zu verstehen
- Nur knapp 40% der Befragten fanden die Inbetriebnahme des GVS einfach und intuitiv.
- Die Exportfunktion ist unter den Studenten weitgehend unbekannt oder wird nicht benutzt.
- Eine Minderheit der Studenten nutzt das Wechseln zwischen Sessions. Fast 45% kennen diese Funktion nicht.
- Die Navigationsmöglichkeiten durch die Session werden gut genutzt (Stepping, Replay)
- Lernschwierigkeiten bieten vor allem folgende Konzepte: Graphentheorie (65%), Sortieralgorithmen (45%), Rekursion (35%)

Unter den Wünschen für ein neues Framework wurden vor allem eine einfachere Installation und Handhabung sowie Unterstützung eines Build-Tools wie Maven genannt.

2.7.1 Einfluss auf Anforderungen

Die Erkenntnisse aus der User Analyse fliessen in die User Stories in Abschnitt 2.9.1 ein.

- Die Installation und Nutzung des ADV muss intuitiv und unkompliziert sein (vgl. User Stories US1, US2, US3).
- Die Import/Export Funktion sowie das Session Switching muss für den Benutzer klar ersichtlich und gut dokumentiert sein (vgl. User Story US13).

2.8 Modulideen

Die Anforderungen an das Framework werden massgeblich durch die potentiellen Anforderungen der Modules beeinflusst. Eine umfassende Analyse der Übungsunterlagen aus AD1 und AD2 führte zu folgenden Modulideen. Sie sind nicht als abgeschlossene Liste zu betrachten, sondern zeigen nur

die vorhanden Ideen zum Zeitpunkt dieser Bachelorarbeit. Aus den Modulideen abgeleitete Anforderungen an das Framework sind in Abschnitt 2.9 aufgeführt.

2.8.1 Array

Das Array-Modul soll dem Studenten eine Basis-Datenstruktur visualisieren. Beispielsweise zeigt ein Snapshot direkt nach der Initialisierung der Datenstruktur einen Array mit Default-Values. Bei Arrays die mit primitiven Typen gefüllt sind, ist ersichtlich, dass die Values direkt im Array gespeichert werden (Abbildung 2.4).

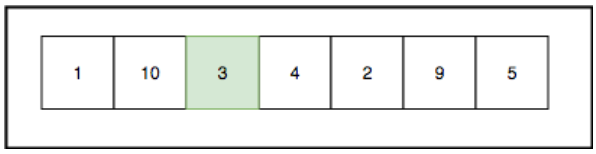


Abbildung 2.4: Design Mock: Modulidee Array mit Value Types

Bei Arrays mit Objekten, zeigt sich das Array mit Objekt-Referenzen gefüllt (Abbildung 2.5).

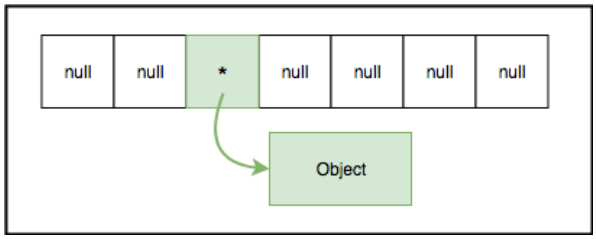


Abbildung 2.5: Design Mock: Modulidee Array mit Objekt Referenzen

Linear Search

Der *Linear Search* Algorithmus kann sehr einfach visualisiert werden, indem das aktuelle Element in einer Schleife farblich hervorgehoben wird. Zusätzlich soll das gefundene Element speziell angezeigt werden.

Binary Search

Beim *Binary Search* im Array muss jeweils jene Hälfte des Arrays farblich gekennzeichnet werden, in welcher die Suche fortgesetzt wird. Damit der Algorithmus angewendet werden kann, muss das Array vorsortiert sein. Das *Module* soll deshalb überprüfen, ob das übergebene Array sortiert ist.

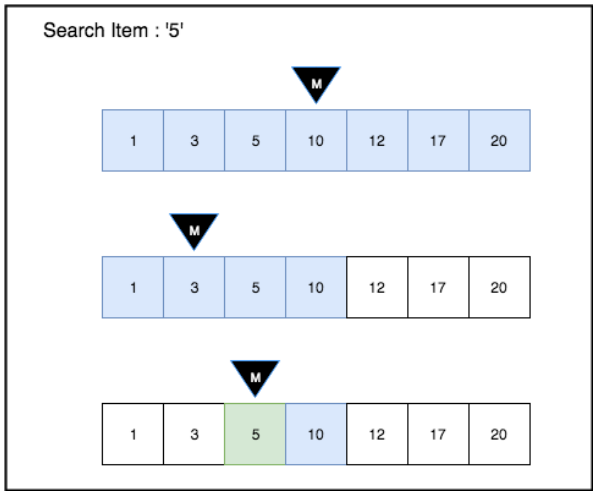


Abbildung 2.6: Design Mock: Modulidee Binary Search

Bubblesort

Beim *Bubblesort* müssen jeweils die zwei Elemente markiert sein, die potentiell vertauscht werden. Zusätzlich könnte das aktuelle Maximum des Arrays speziell markiert werden, welches nach jeder vollständigen Iteration um eins dekrementiert wird.

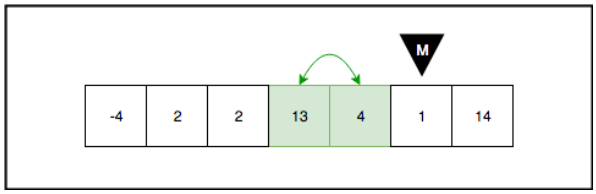


Abbildung 2.7: Design Mock: Modulidee Bubblesort

Quicksort und Mergesort

Beim *Quicksort* und *Mergesort* ist die Darstellung komplexer, da zum besseren Verständnis alle rekursiven Aufrufe dargestellt werden sollen. Zusätzlich müssen temporäre Arrays übergeben und angezeigt werden können.

Ebenfalls muss das Pivot Element speziell markiert werden können. Problematisch könnte die Tatsache sein, dass die beiden Algorithmen oft rekursiv implementiert werden. Das bedeutet, dass das Framework vorerst nur Informationen über den linken Teilbaum zur Verfügung hat, bevor der rechte Teilbaum überhaupt bearbeitet wird. Dabei muss sich das Framework sämtliche Schritte merken und fortlaufend mit neuen Daten ergänzen.

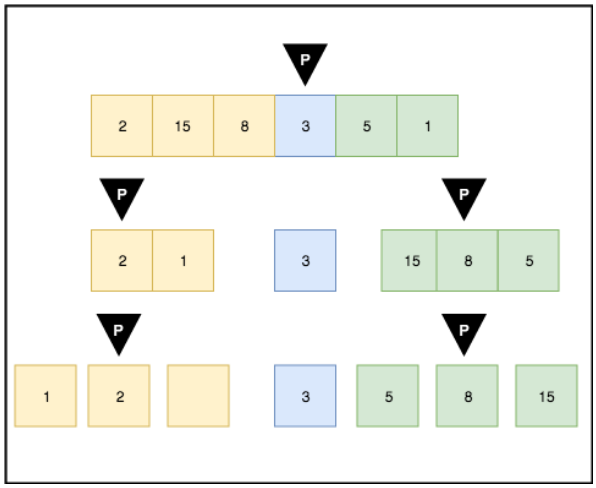


Abbildung 2.8: Design Mock: Modulidee Quicksort und Mergesort

2.8.2 Linked-List

Das Linked-List Modul soll hauptsächlich die zweiseitige Verlinkung der List-Nodes visualisieren. Durch die Visualisierung ist direkt ersichtlich, ob die Objekte korrekt miteinander verlinkt sind (Abbildung 2.9) oder nicht (Abbildung 2.10). Dazu benötigt der ADV Kenntnisse über den Head und den Tail der Liste.

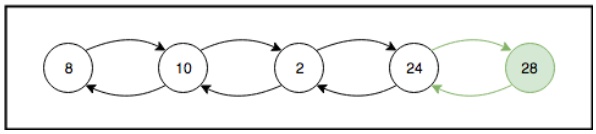


Abbildung 2.9: Design Mock: Modulidee Linked List von korrekter List

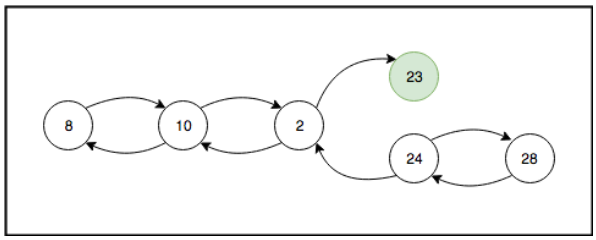


Abbildung 2.10: Design Mock: Modulidee Linked List von inkorrekter List

2.8.3 Stack

Das Stack Modul soll das Last in First Out (LIFO) Verhalten veranschaulichen. Dabei spielt es keine Rolle, ob der Stack als *Array* oder *Linked List* implementiert wurde. Die Darstellung ist bei beiden Varianten die Selbe. Jeder Snapshot zeigt den aktuellen Inhalt des Stacks an.

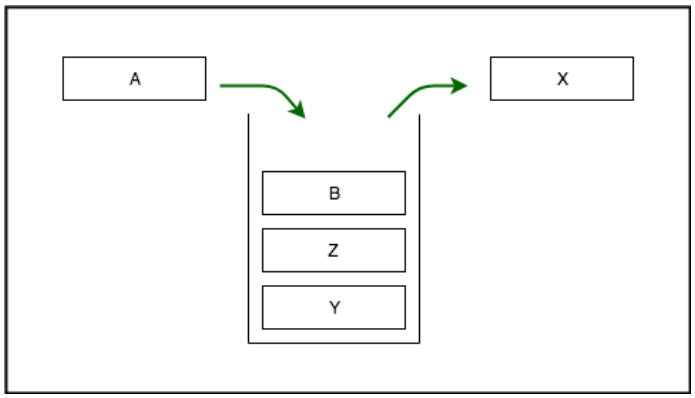


Abbildung 2.11: Design Mock: Modulidee Stack

2.8.4 Graph

Das Graph Modul visualisiert ungerichtete und gerichtet Graphen. Das Graph Modul beinhaltet mehrere Algorithmen die auf der Datenstruktur *Graph* angewendet werden können. Zum Zeitpunkt der Arbeit werden folgende Algorithmen in den Übungen unterrichtet.

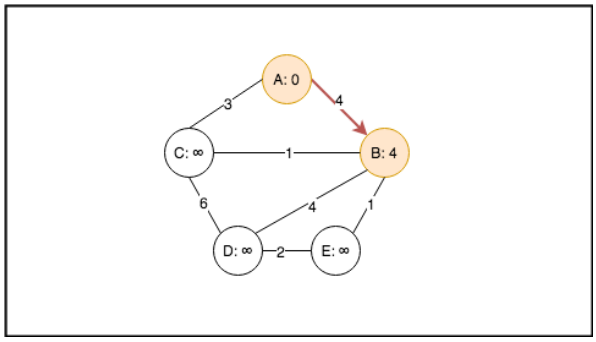


Abbildung 2.12: Design Mock: Modulidee Graph

Depth First Search (DFS)

Bei der *Tiefensuche* müssen die entdeckten Knoten farblich gekennzeichnet werden. Zusätzlich soll die Besuchsreihenfolge in den Edge-Label abgespeichert werden. Ein LIFO Stack gibt Auskunft, welche Knoten als nächstes bearbeitet werden. Der LIFO Stack wird mit den Elementen der Adjazenzliste befüllt. Als optionales Feature könnte ebenfalls die Adjazenzliste dargestellt werden, welche Auskunft über die Nachbarknoten jedes einzelnen Knoten gibt.

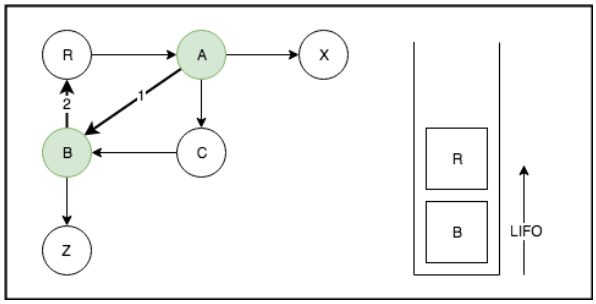


Abbildung 2.13: Design Mock: Modulidee Tiefensuche (DFS)

Breadth First Search (BFS)

Die *Breitensuche* hat die selben Anforderungen wie die *Tiefensuche*. Der BFS verwendet jedoch eine First in First Out (FIFO) Queue, welche das jeweils nächste zu verarbeitende Element darstellt.

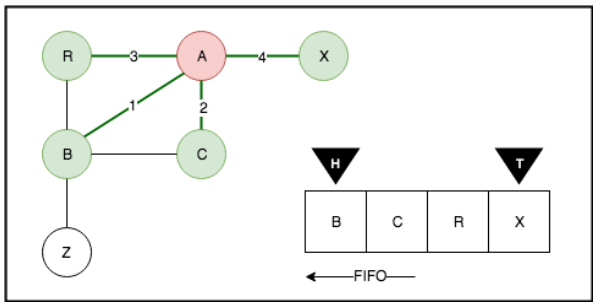


Abbildung 2.14: Design Mock: Modulidee Breitensuche (BFS)

Dijkstra Shortest Path

Für den *Dijkstra* müssen die positiven Gewichte in die Edge Label geschrieben werden können. Ebenfalls müssen die jeweils günstigsten Kosten in das Node Label gespeichert werden können. Auch hier müssen die besuchten und aktiven Nodes und Edges farblich gekennzeichnet werden können.

Bellman Ford Shortest Path

Der *Bellman Ford* Algorithmus stellt die selben Anforderungen wie der Dijkstra, mit dem Unterschied, dass auch negative Gewichte gespeichert werden können.

2.8.5 Tree

Das Modul *Tree* visualisiert n-ary Trees von beliebiger Tiefe. Das Tree Modul visualisiert den Auf-, und Abbau von Tree Strukturen sowie die Binäre Suche.

Für eine ansprechendes Layout der Tree-Nodes kann der Algorithmus von Buchheim, Jünger und Leipert [3] verwendet werden. Dies läuft in linearer Zeit und kann auch für n-ary Trees verwendet werden.

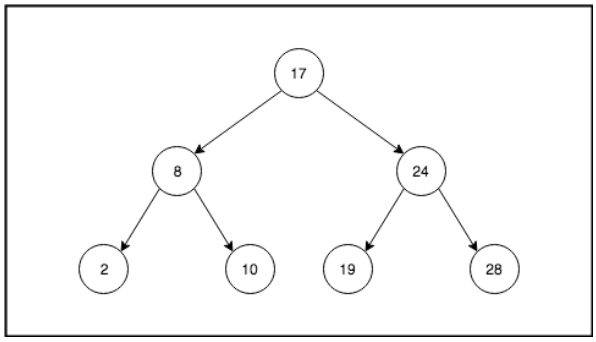


Abbildung 2.15: Design Mock: Modulidee Tree

Tries

Tries oder Präfixbäume verhalten sich wie n-ary Trees. Zusätzlich haben sie eine besondere Anforderung an die Breite der Nodes. Da längere Wörter in die Nodes abgespeichert werden können, sollte der Trie-Node am besten als Rechteck mit abgerundeten Ecken dargestellt werden.

2.8.6 Rekursion

Das Rekursion-Modul soll den Auf- und Abbau des Call-Stacks schrittweise visualisieren. Die Aufrufparameter und Rückgabewerte sowie ausgewählte lokale Variablen werden zur Laufzeit z.B mit Reflection ausgelesen und dargestellt. Sobald der Call-Stack vollständig aufgebaut ist, werden die einzelnen Methodenaufrufe entfernt und die Rückgabewerte abgefüllt. Dies wird solange wiederholt, bis das Resultat im letzten Stack Eintrag dargestellt wird.

Im Unterschied zu den restlichen Modulideen ist dieses Modul selbständig für die Erstellung der Snapshots zuständig. Der *ADV Lib User* programmiert zwar die rekursive Methode, ruft jedoch die `snapshot()` Methode nie selbständig auf. Das Modul analysiert die Rekursive-Methode und erstellt die nötigen Snapshots automatisch. Nach der Analyse können die Snapshots einzeln betrachtet werden.

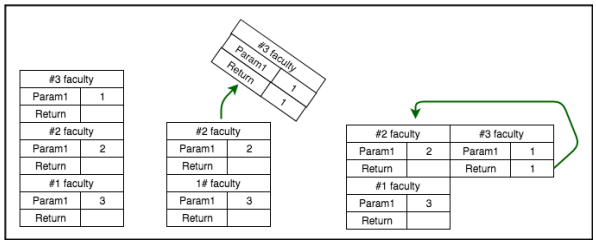


Abbildung 2.16: Design Mock: Modulidee Recursion

2.8.7 Tower of Hanoi

Tower of Hanoi wird oft genutzt um Rekursion einzuführen. Das zu lösende Problem besteht darin, eine bestimmte Anzahl an unterschiedlich grossen Scheiben vom Anfangs-Stab auf den End-Stab zu verschieben. Die Scheiben sind zu Beginn der Grösse nach sortiert, mit der kleinsten Scheibe zuoberst. Zum Ende müssen die Scheiben wieder in der gleichen Reihenfolge, jedoch auf einem anderen Stab angeordnet sein. Es darf immer nur eine Scheibe aufs Mal verschoben werden und grössere Scheiben dürfen nie auf kleineren Scheiben zu liegen kommen.

Da immer nur die oberste Scheibe von einem Stab entfernt werden darf, eignet sich für die Umsetzung ein Stack. Dies bedeutet, dass der Tower von Hanoi mit mehreren *Stack Modules* oder visuell etwas ansprechender mit einem extra *Tower of Hanoi Module* umsetzbar ist. Letzteres ist in Abbildung 2.17 dargestellt.

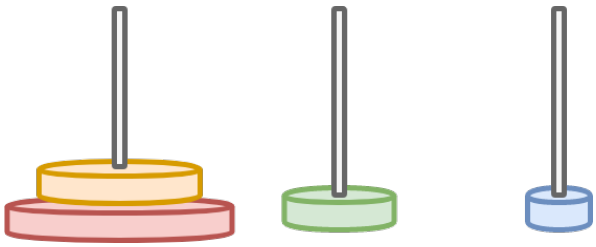


Abbildung 2.17: Design Mock: Modulidee Tower of Hanoi

2.8.8 Labyrinth

Um Rekursion und Back-Tracking zu erklären eignet sich das Labyrinth Problem. Dabei muss in einem zwei-dimensionalen Labyrinth ein Pfad von einem Startpunkt zu einem Endpunkt gefunden und markiert werden. Es darf sich immer nur ein Feld aufs Mal in eine der vier Himmelsrichtungen bewegt werden. Dabei dürfen nur Felder begangen werden, auf denen keine Hindernisse stehen.

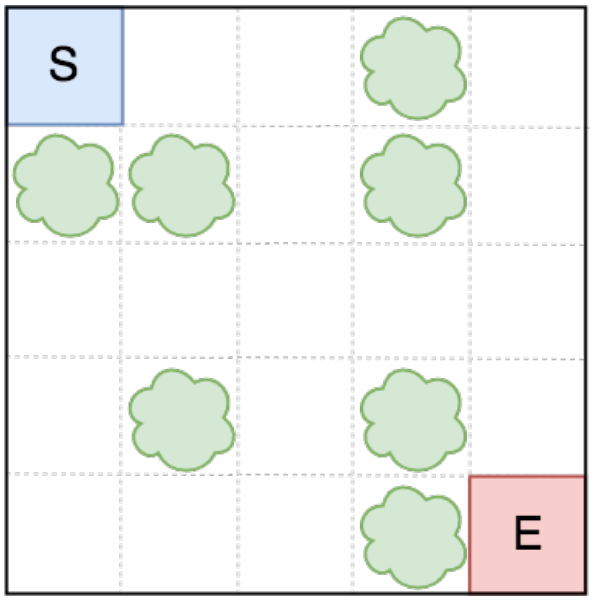


Abbildung 2.18: Design Mock: Modulidee Labyrinth

2.9 Funktionale Anforderungen

Funktionale Anforderungen beschreiben, welche Funktionalität umgesetzt werden muss. Sie werden hauptsächlich aus den Interessen der Anspruchsgruppen abgeleitet. Im folgenden Abschnitt sind die Anforderungen durch User Stories (siehe 2.9.1) und Use Cases beschrieben (siehe 2.9.4).

2.9.1 User Stories: Core

Die User Stories sind pro Stakeholder aus dem Abschnitt 2.6 gruppiert. Zudem zeigt die Priorität an, wie wichtig die Umsetzung vom Projektteam eingestuft wird. Dabei wird eine dreistufige Skala verwendet:

Hoch ≡, Mittel =, Tief _ [1].

Student

US 1: Einfache Nutzung	Priorität: =
Als Student möchte ich bei der Benutzung der ADV Library wenig Boilerplate Code benötigen, um mich bei der Programmierung auf den AD-Unterrichtsstoff zu konzentrieren.	
US 2: Standalone Application	Priorität: ≡
Als Student möchte ich die visualisierten Daten unabhängig vom Status meiner eigenen Applikation betrachten können. Sobald ich einmal Daten an das *ADV UI* gesendet habe, soll sich dieses nicht mehr beenden. Dies ermöglicht mir zwischen dem Source Code und der Visualisierung hin und her zu springen.	
US 3: Einfache Integration	Priorität: =
Als Student möchte ich die beiden ADV Container (Lib und UI) in meinem Buildtool als Dependency hinterlegen können, um wenig Zeit beim Setup des Projekts zu verlieren.	
US 4: Snapshot Stepping	Priorität: ≡
Als Student möchte ich durch die einzelnen Snapshots einer Session navigieren können, um die Abläufe in meinem persönlichen Lerntempo betrachten zu können.	
US 5: Session Replay	Priorität: ≡
Als Student möchte ich eine Session mit einer konfigurierbaren Geschwindigkeit abspielen lassen, um mir einen Überblick über den gesamten Ablauf einer Session zu machen.	
US 6: Session Switching	Priorität: ≡
Als Student möchte ich zwischen verschiedenen Sessions wechseln können, um meine Lösung mit älteren Versionen oder mit der Musterlösung zu vergleichen.	

[1] Tief priorisierte User Stories werden nur bei genügend Zeit umgesetzt.

US 7: Element Positioning	Priorität: ≡
Als Student möchte ich von den Layoutern der Modules profitieren können, welche mir eine anschauliche Positionierung meiner Elemente berechnet. Alternativ möchte ich, wo sinnvoll, die Möglichkeit haben, eine fixe Positionierung von Elementen angeben zu können, um meine Ergebnisse mit anderen Benutzern zu vergleichen.	
US 8: Application Style	Priorität: =
Als Student möchte ich, dass der ADV unabhängig vom aktiven Modul einheitlich aussieht, damit ich die Applikation als Ganzes und nicht als Komposition vieler Teilapplikationen wahrnehme.	
US 9: Theme	Priorität: =
Als Student möchte ich zwischen einem Light und einem Dark Theme wechseln können, damit ich die Benutzeroberfläche dem Umgebungslicht anpassen kann.	
US 10: Debugging	Priorität: ≡
Als Student möchte ich meinen Sourcecode debuggen ohne unerwartete Seiteneffekte im UI feststellen zu müssen.	
US 11: Default Element Style	Priorität: ≡
Als Student möchte ich für meine Elements vorgefertigte Styles nutzen können, um Implementationsaufwand zu sparen.	
US 12: Individual Element Style	Priorität: =
Als Student möchte ich über ein low level API meine eigenen Styles zusammenstellen und verwenden, um die Darstellung möglichst flexibel zu gestalten.	
US 13: Tutorial	Priorität: =
Als Student möchte ich mir ein kleines Tutorial anzeigen lassen, welches mir die grundlegenden Funktionen des ADV erklärt (Stepping, Replay, Session Switching, Import/Export)	

Tabelle 2.6: User Stories: Student

Dozent

US 14: Session Management	Priorität: ☰
Als Dozent möchte ich Sessions speichern und laden können, damit ich diese in der Vorlesung zeigen kann. Ebenfalls möchte ich den Studierenden eine Musterlösung zur Verfügung stellen können.	
US 15: Snapshot Description	Priorität: ☰
Als Dozent möchte ich einem Snapshot eine Beschreibung hinzufügen können, um den Studierenden die Vorgänge erklären zu können.	
US 16: Darstellung auf dem Beamer	Priorität: ☰
Als Dozent möchte ich die Grösse des ADV-Windows beliebig verändern können, damit der ADV auch auf dem Beamer bei kleinen Auflösungen (z.B. 1024x768) nutzbar ist.	

Tabelle 2.7: User Stories: Dozent

Modulentwickler

US 17: Framwork Blackbox	Priorität: ☰
Als Modulentwickler möchte ich mich bei der Entwicklung eines Modules nicht mit dem internen Code des Framework befassen müssen.	
US 18: Routing	Priorität: ☰
Als Modulentwickler möchte ich, dass die darzustellenden Daten vom Framework an das richtige Modul geleitet werden.	
US 19: Layouter	Priorität: ☰
Als Modulentwickler möchte ich die Elemente meines Modules selbständig positionieren können.	
US 20: Zoomable Pane	Priorität: ☰
Als Modulentwickler möchte ich eine vom Framework bereitgestellte Zoomable Pane nutzen können, damit meine Inhalte zentriert und grössen-angepasst dargestellt werden. Für speziellere Anwendungsfälle möchte ich auch eine beliebige andere Pane verwenden können.	

US 21: Style	Priorität: ☰
Als Modulentwickler möchte ich, dass das Framework einen Default-Style vorgibt, den ich bei Bedarf überschreiben kann, um gewisse Modulelemente optisch hervorzuheben.	

US 22: Modulbeschreibung	Priorität: ▁
Als Modulentwickler möchte ich die Möglichkeit haben, für mein Module eine Beschreibung zu hinterlegen, welche dem User eine kleine Hilfe oder Erklärung liefert. Wenn die Beschreibung nicht genutzt wird, soll das entsprechende UI Element ausgeblendet werden.	

US 23: Visualisierungselemente	Priorität: ☰
Als Modulentwickler möchte ich, dass mir das Framework generische Bauelemente bereitstellt, um Implementationsarbeit zu sparen (z.B Directed Edge, Labeled Node, Labeled Edge).	

US 24: Session Navigation	Priorität: ≡
Als Modulentwickler möchte ich, dass das Framework das Durchsteppen der Snapshots, die Replay Funktion und das Session Switching übernimmt.	

US 25: Session Routing	Priorität: ≡
Als Modulentwickler möchte ich, dass das Speichern und Laden von Sessions (gemäss User Story 14: Session Management) vom Framework an das entsprechende Modul delegiert wird, um die nötigen Parse-Schritte durchzuführen.	

US 26: Verbindungsaufbau	Priorität: ≡
Als Modulentwickler möchte ich den Verbindungsaufbau zwischen *ADV Lib* und *ADV UI* dem Framework überlassen. Zudem möchte ich das verwendete Übertragungsprotokoll erweitern können, um spezielle Ansprüche meines Modules abzudecken.	

US 27: Hooks	Priorität: ≡
Als Modulentwickler möchte ich den Programmfluss über Hook-Methoden beeinflussen können, um modulspezifische Erweiterungen bei den wichtigen Schritten anfügen zu können (z.B Animation bei Stepping).	

US 28: Menschenlesbares Dateiformat	Priorität: ☰
Als Modulentwickler möchte ich, dass die Sessions in einem menschenlesbaren Format persistiert werden, damit das Erstellen einer Testsession mit wenig Aufwand verbunden ist.	

Tabelle 2.8: User Stories: Modulentwickler

2.9.2 User Stories: Array Module

Die User Stories im *Array Module* richten sich hauptsächlich an die Anforderungen des Stakeholder *Student* und *Modulentwickler*.

US 29: Darstellung von Array	Priorität: ☰
Als Student möchte ich einen Array mit fixer Grösse und konfigurierbarer Typinformation visualisieren können.	
US 30: Darstellung Objektreferenzen	Priorität: ☰
Als Student möchte ich Objektreferenzen optional speziell darstellen lassen, damit ich mir ein besseres Bild über die effektive Repräsentation im Speicher machen kann.	
US 31: Algorithmus Visualisierung	Priorität: ☰
Als Student möchte ich ausgewählte Elemente visuell hervorheben, um Array Algorithmen (z.B Bubblesort) besser verstehen zu können. Ich möchte dazu vorgefertigte Style-Klassen verwenden, die ich nach meinem persönlichen Geschmack anpassen kann.	
US 32: Unterschiedliche Layouts	Priorität: ☰
Als Modulentwickler möchte ich unterschiedliche Layouts für Arrays erstellen können, um spezifische Ausprägungen meines Moduls unterschiedlich darstellen zu können.	

Tabelle 2.9: User Stories *Array Module*

2.9.3 User Stories: DFS/BFS Module

Die beiden Algorithmen DFS und BFS arbeiten beide auf einem Graphen und sind daher sehr ähnlich. Unterschiedlich ist neben der eigentlichen Implementierung die verwendete Hilfsdatenstruktur. Der DFS Algorithmus arbeitet mit einem Stack und der BFS Algorithmus mit einer Queue. Die

Hilfsdatenstrukturen werden dazu verwendet, um Adjazenz-Knoten in der gewünschten Reihenfolge abzuarbeiten.

US 33: Multimodule	Priorität: ☰
Als Modulentwickler möchte ich bestehende *Modules* wiederverwenden können, um Code Duplikationen so weit wie möglich vermeiden zu können.	
US 34: Hilfsdatenstrukturen	Priorität: ☰
Als Student möchte ich die vom Algorithmus verwendete Hilfsdatenstruktur (*Stack* oder *Queue*) parallel zur Hauptdatenstruktur (*Graph*) darstellen lassen.	
US 35: Vordefinierte Styles	Priorität: ☰
Als Student möchte ich auf vordefinierte Styles für die Kanten-Typen *DISCOVERY*, *BACK*, *FORWARD*, *CROSS* zurückgreifen können.	

Tabelle 2.10: User Stories *DFS/BFS Generell*

Graph-Module

US 36: Darstellung von Graphen	Priorität: ☰
Als Student möchte ich einen Graphen darstellen können, auf welchem die beiden Algorithmen arbeiten können	
US 37: Node Label	Priorität: ☰
Als Student möchte ich Knoten mit einem Label versehen können, um die berechneten Pfadkosten oder die Anzahl inzidenter Kanten eines Knotens darstellen zu können.	
US 38: Node Styling	Priorität: ☰
Als Student möchte ich den Zustand eines Graph-Nodes grafisch kennzeichnen können, um besuchte Knoten von nicht besuchten Knoten unterscheiden zu können.	
US 39: Gewichtete Kanten	Priorität: ☰
Als Student möchte ich das Gewicht einer Kante darstellen können.	
US 40: Gerichtete Kanten	Priorität: ☰
Als Student möchte ich die Richtung von Kanten konfigurieren können, damit ich z.B einen Directed Acyclic Graph (DAG) darstellen kann.	

US 41: Edge Styling	Priorität: ≡
Als Student möchte ich den Zustand einer Graph-Kante grafisch kennzeichnen können, um z.B den direktesten Pfad beim BFS Algorithmus darstellen zu können.	

Tabelle 2.11: User Stories *Graph Module*

Stack-Module

US 42: Darstellung von Stacks	Priorität: ≡
Als Student möchte ich einen Stack darstellen können, welcher das LIFO Prinzip veranschaulicht.	

Tabelle 2.12: User Stories *Stack Module*

Queue-Module

US 43: Darstellung von Stacks	Priorität: ≡
Als Student möchte ich eine Queue darstellen können, welcher das FIFO Prinzip veranschaulicht.	

Tabelle 2.13: User Stories *Queue Module*

2.9.4 Use Cases

User Stories mit einer hohen funktionalen Komponente, welche Interaktionen mit dem GUI umfassen, sind in diesem Abschnitt als Use Cases ausformuliert. Auf ein Use Case Diagramm wird bewusst verzichtet, da es pro Use Case nur einen einzelnen relevanten Aktor gibt (siehe Abschnitt 2.5).

UC1: Snapshot anzeigen

Name	UC-1: Snapshots anzeigen
Kurzbeschreibung	Ein Student hat eine Datenstruktur oder ein Algorithmus ausprogrammiert und möchte diese nun visualisieren.
User Stories	US 4
Aktoren	ADV UI User
Trigger	Aufruf einer speziellen Methode aus der ADV Lib, welche einen Snapshot erzeugt. Der Aufruf erfolgt in der User Codebase
Vorbedingung	*ADV Lib* ist im Projekt eingebunden und ein entsprechendes Modul existiert.
Standardablauf	1. Implementation der Datenstruktur oder Algorithmus durch den ADV User. 2. Aufruf einer Library Methode, welche einen Snapshot erstellt und diesen an das *ADV UI* sendet. 3. Der aktuelle Stand des Codes wird von der *ADV Lib* an das *ADV UI* übertragen und dargestellt. 4. Beliebige Wiederholung von Punkt 2 und 3 5. Alle Snapshots innerhalb der Lebensdauer des Aktor-Codes werden im *ADV UI* als Session zusammengefasst. 6. Gemäss UC3 in Tabelle 2.16 kann nun durch die aktive Session navigiert werden
Erweiterungen	1.1 Die Datenstrukturen werden unter Verwendung von Interfaces aus der ADV Library ausprogrammiert.
Nachbedingung	keine

Tabelle 2.14: UC1: Snapshots anzeigen

UC2: Laden und Speichern einer Session

Name	UC2: Laden und Speichern einer Session
Kurzbeschreibung	Eine dargestellte Session wird auf dem Dateisystem persistent abgespeichert oder von dort geladen. • Der Dozent speichert eine Session als Musterlösung ab • Der Student speichert seine persönliche Lösung für die Prüfungsvorbereitung ab
User Stories	US 14
Aktoren	ADV User
Trigger	Aktor wählt Import/Export Funktion aus
Vorbedingung	Import: das geladene File ist ein *.adv File. Export: eine Session mit mindestens einem Snapshot existiert.
Standardablauf	Export 1. Eine bestehende Session mit mindestens einem Snapshot wird im ADV angezeigt 2. Der Aktor klickt auf den *Export* Button 3. Der Aktor wählt den gewünschten Speicherort und benennt die Session. Standardmässig wird der Sessionname als Dateiname vorgeschlagen. 4. Die Session wird auf der Festplatte gemäss Angaben des Aktors gespeichert. Import 1. Der Aktor klickt auf den *Import* Button 2. Im erscheinenden File Browser navigiert der Aktor zu einem *.adv File und öffnet dieses. 3. Die gespeicherte Session wird geladen und angezeigt 4. Gemäss UC3 in Tabelle 2.16 kann nun durch die aktive Session navigiert werden
Nachbedingung	keine

Tabelle 2.15: UC2: Laden und Speichern einer Session

UC3: Session Navigation

Name	UC3: Session Navigation
Kurzbeschreibung	Innerhalb einer Session kann zwischen den einzelnen Snapshots navigiert werden.
User Stories	US 4, US 5
Aktoren	ADV User
Trigger	Klicken der entsprechenden Navigation-Buttons
Vorbedingung	eine Session mit min. einem Snapshot existiert.
Standardablauf	1. Die Snapshots einer Session können über vier Navigation-Buttons durchgesteppt werden (Navigation zum ersten, nächsten, vorherigen und letzten Snapshot einer Session) 2. Das Durchsteppen durch die Snapshots kann durch die Replay Funktion automatisiert ablaufen a) Das Betätigen des Start Buttons startet das Replay ausgehend vom aktuellen Snapshot. b) Der Pause Button pausiert das Replay. Gemäss Punkt 1 kann nun manuell navigiert werden oder das Replay kann durch den Resume Button fortgeführt werden. c) Der Cancel Button stoppt das Replay und setzt die Session auf den ersten Snapshot zurück.
Erweiterungen	1.1 Navigations Buttons sind nur aktiv, wenn die entsprechende Navigation ausführbar ist. Beispielsweise sind alle Buttons deaktiviert, wenn die Session nur einen Snapshot umfasst. 2.1 Die Abspielgeschwindigkeit des Replays kann nur verändert werden, wenn der Replay nicht läuft (gestoppt oder pausiert ist).
Nachbedingung	keine

Tabelle 2.16: UC3: Session Navigation

UC4: Session Switching

Name	UC4: Session Switching
Kurzbeschreibung	Zur Laufzeit kann zwischen der aktiven und zuvor übertragenen oder geladenen Sessions gewechselt werden. Dies ist zum Beispiel der Fall, wenn der Student seine Lösung mit der geladenen Musterlösung vergleichen will.
User Stories	US 6
Aktoren	ADV User
Trigger	Interaktion im GUI
Vorbedingung	Mindestens zwei Sessions existieren
Standardablauf	1. Ein beliebiger Snapshot aus einer aktiven Session wird angezeigt 2. Der Aktoren klickt auf eine andere Session in der Liste der aktiven Sessions 3. Die gewählte Session wird aktiv und der erste Snapshot der neuen Session wird angezeigt. 4. Gemäss UC3 in Tabelle 2.16 kann nun durch die aktive Session navigiert werden
Nachbedingung	keine

Tabelle 2.17: UC4: Session Switching

2.9.5 Technische Anforderungen

Logging

Die Richtlinien der Bachelorarbeit verlangen ein zweidimensionales Logging mit Laufzeit Konfiguration. In der ersten Dimension muss das Package und in der zweiten Dimension das Log Level ersichtlich sein. Das Log Level muss pro Package zur Laufzeit verändert werden können. Logging ist sofern sinnvoll, intensiv einzusetzen.

Persistenz

Die Sessions müssen persistent auf dem Dateisystem abgelegt werden können. Dazu ist eine Art Serialisierung der Objekte in einem verbreiteten Dateiformat notwendig. Das Dateiformat soll für Menschen lesbar abgelegt werden.

Parallelität

Das Empfangen und Verarbeiten der Daten muss in einem eigenständigen Thread passieren, damit das Empfangen von neuen Snapshots in Realtime ersichtlich ist und das UI nicht blockiert.

2.10 Nicht-funktionale Anforderungen

NFRs definieren Qualitätsattribute die von der Applikation erfüllt werden müssen.

SMART Akronym
Das SMART Akronym unterstützt den Requirement Engineer beim Definieren von überprüfbaren Qualitätsattributen. Das Akronym kennt in der Literatur verschiedenen Auslegungen. Im Rahmen dieser Arbeit wird folgende Spezifikation verwendet. Ein Schwerpunkt wird auf die Anforderungen Specific und Measurable gelegt.

Specific Welche Komponenten sind von der Anforderung betroffen?

Measurable Kann ein Tester/Stakeholder herausfinden ob die Anforderung erfüllt wird oder nicht?

Achievable Ist die Anforderung realistisch gewählt?

Relevant Gibt es ein Verlangen für die Anforderung?

Time-bound Die Anforderung hat eine zeitliche Komponente.

2.10.1 Performance

Das *ADV UI* muss unter normaler Systemauslastung innerhalb von 5 Sekunden nach Initialisierung der *ADV Lib* dargestellt werden. Dabei wird die benötigte Zeit für den Buildprozess nicht eingerechnet. Als Resultat läuft die Visualisierungs-Komponente (JavaFX) in einem eigenständigen Thread und ein Kommunikationskanal ist zwischen den beiden Prozessen geöffnet.

2.10.2 Erweiterbarkeit und Flexibilität

Das Protokoll muss soweit erweiterbar sein, dass die angedachten Modulideen (siehe Abschnitt 2.8), ohne Anpassungen am Framework umgesetzt werden können. Die Interpretation des Protokolls soll vollständig in der Verantwortung des Modulentwicklers liegen.

2.10.3 Robustheit und Fehlertoleranz

Im Falle eines auftretenden Fehlers innerhalb der User Codebase, soll der *ADV UI* Prozess ungehindert weiter laufen.

2.10.4 Skalierbarkeit

Das ADV Core Framework muss mindestens durch zwei Modules erweitert werden können, ohne das NFR «Performance» (Abschnitt 2.10.1) zu verletzen. Diese Anforderung ist als Minimalanforderung zu verstehen, da im Rahmen dieser Arbeit keine weiteren *Modules* programmiert werden können.

2.10.5 Testabdeckung

Die Schnittstellen der Core Komponenten müssen eine hohe aber vernünftige Testabdeckung erreichen. Unter vernünftig versteht sich eine Abdeckung der Business Logik von mindestens 75%. Davon ausgeschlossen sind Klassen zur Visualisierung sowie Plain Old Java Objects (POJOs).

2.10.6 Testbarkeit

Die Architektur soll so designet sein, dass Klassen isoliert getestet werden können. Das Konzept der hohen Kohäsion und tiefen Kopplung soll eingehalten werden.

2.10.7 Accessability

Der ADV muss auf gängigen Betriebssystemen (Windows, macOS, Linux) funktionsfähig und ausführbar sein.

2.10.8 Verständlichkeit und Benutzbarkeit

Der ADV muss unter Zuhilfenahme der Dokumentation für Modulentwickler innerhalb von einem Tag erweitert werden können. Dazu gehört nur die Anbindung an das Framework sowie die Verwendung der generischen Funktionalität. Modul-spezifische Arbeiten sind in diesem NFR ausgenommen.

2.10.9 Usability

Das GUI des ADV UI muss soweit selbsterklärend sein, dass die vier Use-Cases (siehe 2.9.4) ohne fremde Hilfe durchgeführt werden können. Es wird davon ausgegangen, dass die Benutzer mit der Bedeutung verbreiteter Icons vertraut sind.

2.10.10 API Dokumentation

Um die zukünftige Entwicklung von Modules zu erleichtern, muss das API dokumentiert und eine umfassende Benutzeranleitung speziell für Modul-Entwickler erstellt werden. Die gesamte Dokumentation muss aktuell sein, dem effektiven Code entsprechen und Anwendungsbeispiele enthalten.

Kapitel 3

Architektur und Design Spezifikationen

Datum	Version	Änderungen	Autor
20.02.2018	0.01	Dokument erstellt	mwieland
26.02.2018	0.02	C4 Model beschrieben, Context Diagramm eingefügt	mtrentini
05.03.2018	0.03	Container Diagramm eingefügt inkl. Y-Statement	mtrentini
12.03.2018	0.04	IPC Optionen verglichen inkl. Y-Statement	mtrentini
26.03.2018	0.05	Protokoll beschrieben	mwieland
02.04.2018	0.06	Klassendiagramme, Layering beschrieben	mtrentini
02.04.2018	0.07	Sequenzdiagramm Flow-Control beschrieben	mtrentini
03.04.2018	0.08	Qualitätsattribute und Benutzeroberfläche beschrieben	mwieland
03.04.2018	0.09	Threading, Logging, Exception Handling und Testing beschrieben	mwieland
04.04.2018	0.010	Events beschrieben	mwieland
04.04.2018	0.1	Review Dokument	mwieland, mtrentini

14.04.2018	0.11	Y-Statement für User Story Hooks geschrieben	mtrentini
14.04.2018	0.12	C4 Diagramme überarbeitet	mtrentini
20.05.2018	0.13	Draggable Tabs beschrieben	mtrentini
29.05.2018	0.14	ADV Commons integriert	mtrentini
29.05.2018	0.15	Modulwahl DFB/BFS Module	mtrentini
11.06.2018	1.0	Review Dokument	mwieland, mtrentini

Tabelle 3.1: Versionshistory Architektur und Design Spezifikationen

3.1 Einleitung

Die vorliegende *Architektur und Design Spezifikationen* beschreibt die Architekturentscheide welche für diese Arbeit getroffen wurden. Neben einer Übersicht auf vier Abstraktionsstufen (siehe C4 Model im Abschnitt 3.2), werden die wichtigsten Entscheide wie beispielsweise die Interprozesskommunikation genauer dokumentiert.

Architekturentscheide
Die Architekturentscheide in dieser Arbeit werden als (WH)Y-Statements verfasst. Dabei handelt es sich um ein Template, welches den Entwickler beim Definieren von nachvollziehbaren und strukturierten Entscheiden unterstützt. Ein Y-Statement enthält neben den positiven und negativen Eigenschaften einer Entscheidung auch die verworfenen Alternativen. [70]

3.2 C4 Model

Das C4 Model [14] ermöglicht es, die Struktur einer Software in verschiedenen Detaillierungsgraden und Abstraktionsebenen zu kommunizieren. Das Modell umfasst vier Diagramme: Context, Container, Component und Code. Die Abbildung 3.1 gibt einen Überblick über die vier Zoom-Stufen.

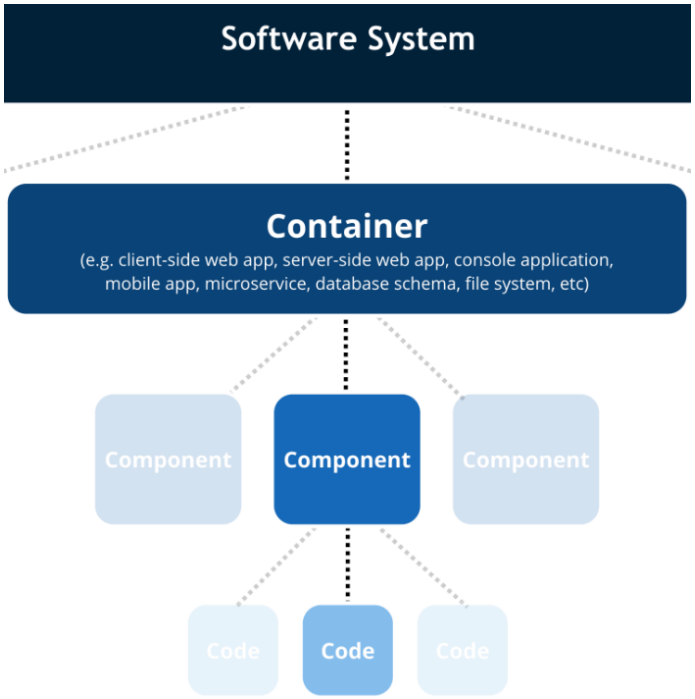


Abbildung 3.1: Offizielle C4 Übersicht [14]

Context
Zeigt das System als Blackbox mit externen Abhängigkeiten, umliegenden Systemen und Aktoren. Es zeigt, in welchem Kontext das System eingesetzt wird.

Container
Ein Container ist eine unabhängig laufende Einheit, welche Code ausführt oder Daten speichert wie beispielsweise eine Web Applikation, eine Mobile App oder eine Datenbank. Das Container Diagramm zeigt die *high-level* Form der Software Architektur, die Verantwortlichkeiten der Container und grundlegende Technologieentscheide.

Component
Jeder Container besteht aus einem oder mehreren *Components*, welche die hauptsächlichen Bausteine des Systems widerspiegeln. Diese Bausteine können logische Komponenten, Subsysteme, Layers und Workflows umfassen.

Code
Das Code Diagramm zeigt die detaillierteste Ansicht des Systems. Es entspricht einem klassischen UML Klassendiagramm und zeigt, wie eine Komponente im Code umgesetzt wird.

3.3 Kontextdiagramm

Das Kontextdiagramm in Abbildung 3.2 zeigt den ADV sowie das externe Software System *User Codebase*. Die beiden Rollen *ADV Lib User* und *ADV UI User* nutzen die Funktionalität des ADV. Im Normalfall werden beide Rollen von der selben Person eingenommen (siehe Abschnitt Aktoren 2.5).

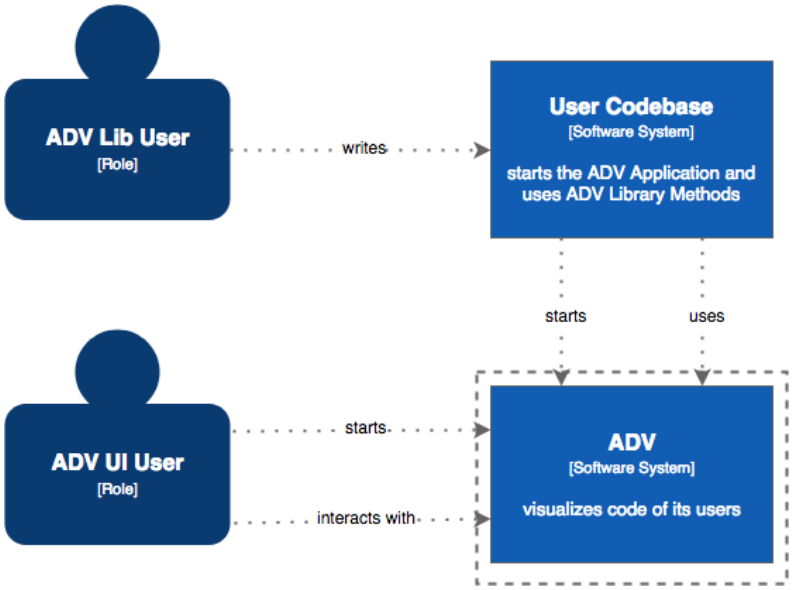


Abbildung 3.2: C4 System Context Diagram

3.4 Containerdiagramm

Auf der Abstraktionsstufe *Container* bestehen zwei grundsätzliche Möglichkeiten, das System zu realisieren:

1. Einen einzigen Container, welcher Klassen zur Visualisierung sowie Library-Klassen zur Erweiterung bereithält. Die gesamte Applikation wird als ein Java Archive (JAR) Library File ausgeliefert. Der Programmcode des Studenten und die Visualisierung läuft in demselben Java-Prozess.

2. Je ein Container für die ADV Lib und das ADV UI sowie ein Container für gemeinsam verwendete Klassen (ADV Commons). Die beiden Haupt-Container (Lib und UI) laufen in eigenständigen Prozessen. Die beiden Container implementieren das Client-Server Pattern [7], wobei die ADV Lib (Client) die Dienste des ADV UI (Server) in Anspruch nimmt. Die Kommunikation zwischen Client und Server erfolgt klassischerweise synchron.

(WH)Y Statement *Container Konzept*:
Im Kontext des ADV für den AD-Unterricht an der HSR, konfrontiert mit dem Bedarf eines möglichst hohen Lernerfolgs des Studenten, wird die oben aufgelistete *Variante 2* gewählt. Die Variante ermöglicht dem Studenten, den Zustand seiner Visualisierungen auch noch nach dem Terminierung des Studenten-Prozesses betrachten zu können. Zudem definiert es eine klarere Trennung zwischen Library und Visualisierung.

Verworfen wird die *Variante 1*, da nach dem Beenden des User-Prozesses sämtliche Visualisierungen verloren gehen. Zusätzlich lässt sich mit dieser Variante weniger gut gleichzeitig am Quellcode arbeiten und die Visualisierungen einsehen.

Mit diesem Entscheid nehmen wir in Kauf, dass ein leicht erhöhter Entwicklungsaufwand durch die Interprozesskommunikation entsteht und der Student zwei Abhängigkeiten in seinem Projekt definieren muss. Die Wahl des Client-Server-Patterns hat durch die klassischerweise synchrone Umsetzung Einfluss auf die Wahl der Interprozesskommunikation (IPC) Technologie (siehe Abschnitt Interprozesskommunikation 3.8).

Die aus diesem Architekturentscheid resultierende grobe Architektur ist im Container Diagramm in Abbildung 3.3 ersichtlich.

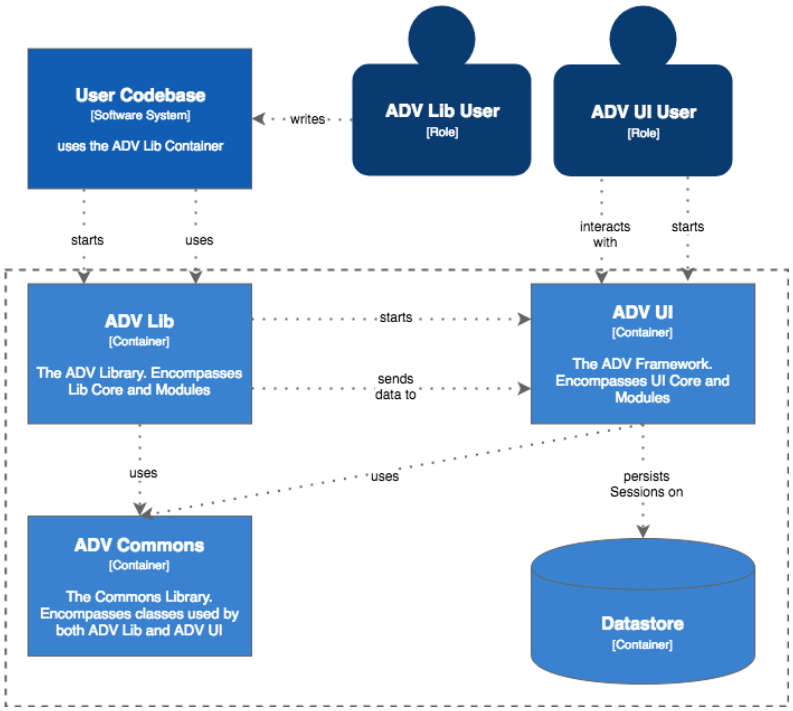


Abbildung 3.3: C4 Container Diagram

3.5 Komponentendiagramm

Alle drei Container des ADV werden in mehrere Komponenten unterteilt. Sie verfügen jeweils über eine Core Komponente sowie mehrere Module Komponenten. Der Core enthält wiederverwendbare Konzepte, die von den Modulen benutzt werden. Ausserdem ist er für die Client-Server Kommunikation zwischen der ADV Lib und dem ADV UI zuständig (siehe 3.8).

Die Core Komponenten des ADV-UI enthält zudem das Framework, welches den Prozessfluss für die Visualisierung der Datenstrukturen vorgibt. Eine ausführliche Beschreibung über die Framework Features ist dem Abschnitt 3.12 zu entnehmen.

Das Komponentendiagramm in Abbildung 3.4 zeigt die verschiedenen Komponenten am Beispiel des ADV UI Containers. Die weiteren zwei Container sind analog dazu strukturiert.

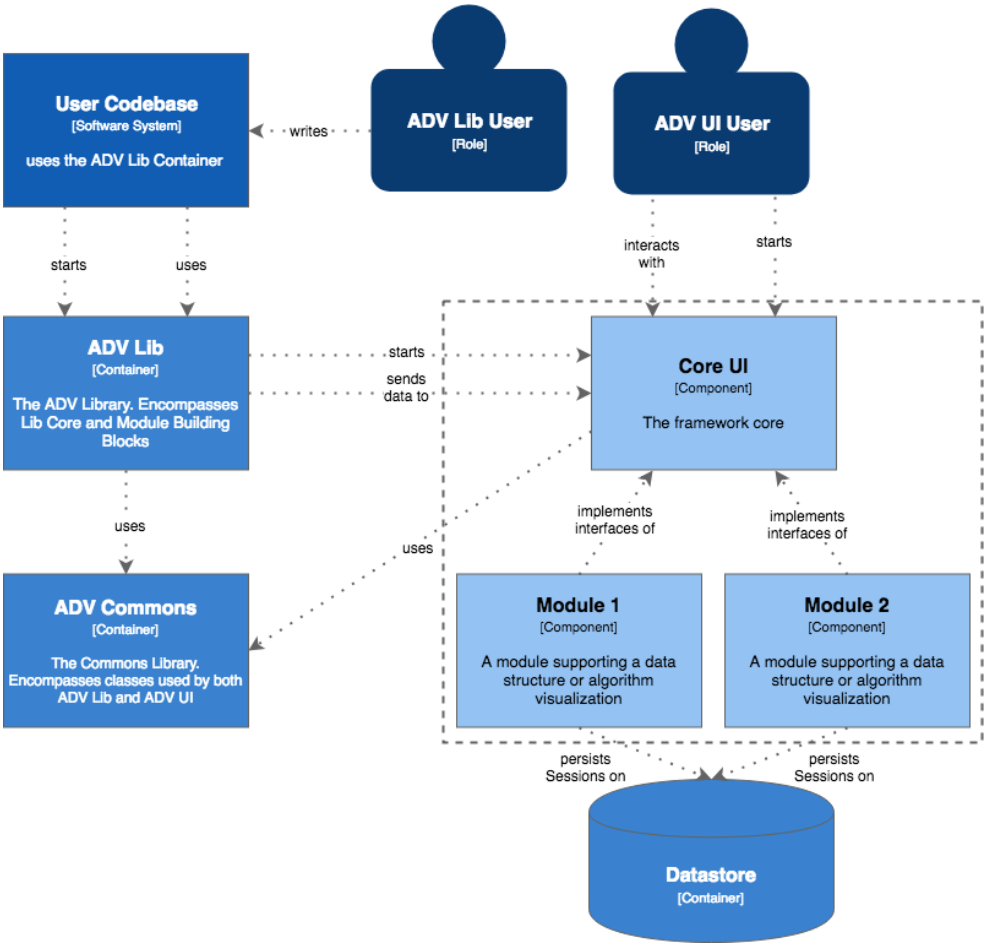


Abbildung 3.4: C4 Component Diagram: ADV UI

3.6 Layering

Das ADV Projekt gliedert sich in mehrere Layer, welche sich in der Package-Struktur widerspiegeln. Damit wird das Pattern *Layered Architecture* [46] umgesetzt. Zwischen den Schichten sind nur Zugriffe von höheren Schichten zu tiefer liegenden Schichten erlaubt. Die Abbildung 3.5 zeigt die Layer der entsprechenden Container. Analog zu den Containern sind auch die Components gemäss dieser Struktur organisiert.

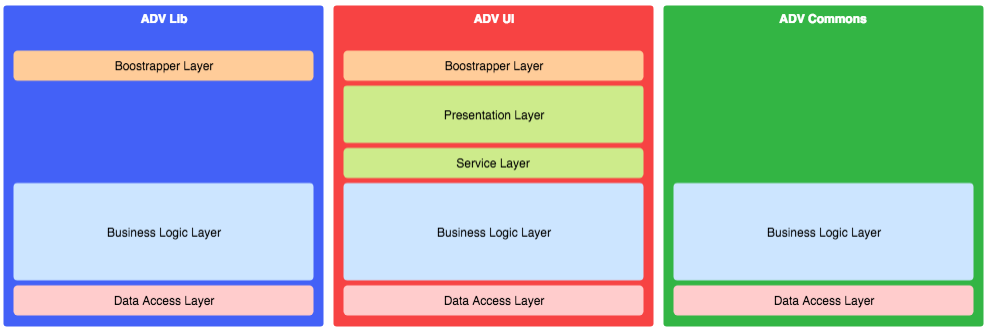


Abbildung 3.5: Layering der Container

3.6.1 Bootstrapper Layer

Der Bootstrapper Layer initialisiert den DI Container. Ferner liest er die annotierten Modul-Services per Reflection aus und registriert sie im Framework (siehe *Module* Bootstrapping 3.12.3).

3.6.2 Presentation Layer

Der Presentation Layer visualisiert die Objekte aus dem Business Layer. Er ist nach dem MVVM Prinzip organisiert (siehe Model-View-ViewModel 3.6.2). Der Presentation Layer beinhaltet die meisten Abhängigkeiten zu JavaFX. Es existieren aber auch einige Klassen im Business Layer, welche JavaFX Objekte erzeugen. Um die Architektur so einfach wie möglich zu halten, wird auf ein zusätzliche Indirektion zugunsten einer besseren Kapselung von JavaFX verzichtet. Die Ausnahme betrifft die Klassen *LayoutedSnapshot*, *LayoutedSnapshotStore*, *FlowControl* sowie *Core Layouter* (inkl. *Layouter Interface*).

Model-View-ViewModel

MVVM [8] dient der Trennung zwischen dem User Interface und der Anzeigelogik. Es ist eine Konkretisierung des verbreiteten MVC Pattern und setzt stark auf Databindings. Es wurde von Microsoft für das Windows Presentation Foundation (WPF) Framework entwickelt und wird auch in modernen JavaFX Applikation eingesetzt.

Model Das *Model* enthält die Business Logik. Werden die Werte im Model verändert, wird das *ViewModel* über einen Property Change Event benachrichtigt. Das Model liegt im ADV Projekt nicht im Presentation Layer sondern im Business Logic Layer.

View Die *View* stellt den UI Zustand des *ViewModels* dar. Sie setzt sich aus einer Controller Klasse sowie einem deklarativen FXML zusammen. Die View hält eine Instanz des ViewModels.

ViewModel Das *ViewModel* kennt die View nicht. Es hat somit keine Abhängigkeiten zu konkreten Anzeige-Elementen. Dies hat den grossen Vorteil, dass die gesamte UI Logik im *ViewModel* gekapselt und somit besonders gut testbar ist. Das ViewModel enthält JavaFX spezifische Properties [6], die für das bidirektionale Binding zwischen den UI-Komponenten und dem Code Behind nötig sind.

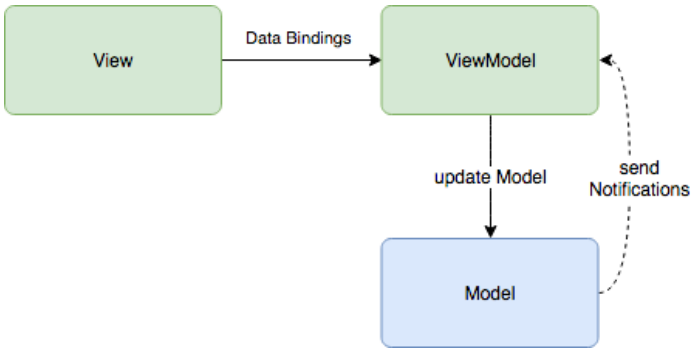


Abbildung 3.6: Übersicht MVVM

3.6.3 Service Layer

Der Service Layer definiert eine öffentliche Schnittstelle in das ADV UI. Über einen Socket Endpoint, welcher im Service Layer definiert ist, werden die Daten empfangen und an die unteren Layer zur Verarbeitung weitergereicht. Der Service Layer ist neben dem Presentation Layer ein alternativer Kanal in die Applikation (vgl. Service Layer [4]).

3.6.4 Business Logic Layer

Im Business Logic Layer liegt die Logik der Applikation. Er enthält die Kontrollflusssteuerung des Frameworks, sowie die Modul-Services welche eingehende Anfragen verarbeiten. Ausserdem enthält der Business Logic Layer die Domänenobjekte (meist POJOs).

3.6.5 Data Access Layer

Der Data Access Layer hat je nach Container unterschiedliche Aufgabenbereiche. Im ADV Lib ist er für die Socket Kommunikation mit dem ADV UI zuständig (vgl. Service Agent [5]). Im AVD UI regelt der Data Access Layer den Zugriff auf die Festplatte und ist für die Persistierung von Sessions zuständig (vgl. Data Access Components [5]).

3.7 Klassendiagramme

Die Klassendiagramme für das ADV UI, die ADV Lib und das ADV Commons zeigen eine Übersicht über die wichtigsten Klassen der drei Container. Sie sind dem Anhang C zu entnehmen.

3.8 IPC: Interprozesskommunikation

Die beiden Container ADV Lib und ADV UI werden in unterschiedlichen Prozessen modelliert. Daraus entsteht die Anforderung der IPC, die im folgenden Abschnitt genauer beschrieben ist.

3.8.1 Dimensionen

In der Interprozesskommunikation kommen verschiedene Dimensionen zum Tragen [38]. Die Tabelle 3.2 erklärt die vier Dimensionen und zeigt die Bedürfnisse des ADV.

Dimension	Erklärung	ADV
one-to-one	ein Sender sendet nur an einen Empfänger	one-to-one oder many-to-one
one-to-many	ein Sender sendet an mehrere Empfänger	
may-to-one	mehrere Sender senden an einen Empfänger (Webserver)	
simplex	Zwischen Sender und Empfänger findet eine Ein-Weg Kommunikation statt	duplex
duplex	Zwischen Sender und Empfänger findet eine Zwei-Weg Kommunikation statt	
local	Die beteiligten Prozesse laufen lokal auf dem gleichen Rechner	beides
distributed	Die beteiligten Prozesse sind verteilt	
synchronous	Die Prozesse kommunizieren synchron, d.h. beide Prozesse nehmen aktiv an der Kommunikation teil (z.B HTTP)	synchronous
asynchronous	Die Prozesse kommunizieren asynchron, d.h. der Sender wird nicht blockiert und die Prozesse müssen nicht gleichzeitig aktiv sein (z.B Asynchronous Message Queueing).	

Tabelle 3.2: IPC Dimensionen

3.8.2 Variantenvergleich und Architekturentscheid

Um eine prozess-übergreifende Kommunikation zu gewährleisten, werden mehrere mögliche Varianten evaluiert und miteinander verglichen. Die Ergebnisse sind in der Tabelle 3.2 beschrieben.

#	Variante	one-to-one	many-to-one	duplex	local	distributed	synchron
1	Anonymous Pipes	✓			✓		✓
2	Named Pipes	✓	✓	✓	✓	✓	✓
3	Shared Memory	✓	✓	✓	✓		✓
4	Sockets	✓	✓	✓	✓	✓	✓

Tabelle 3.3: IPC Variantenvergleich

(WH)Y Statement *IPC Kommunikation*:
Im Kontext des ADV für den AD-Unterricht an der HSR, konfrontiert mit dem Bedarf einer gemäss Tabelle 3.2 spezifizierten Interprozesskommunikation wird die oben aufgelistete *Variante 4* gewählt. Diese ermöglicht eine IPC die dem Bedarf entsprechend synchron und many-to-one implementiert werden kann, ohne dass dabei Software von Dritt-Anbietern benötigt wird. Zudem können Sockets für lokale Anwendungen sowie für verteilte Systeme verwendet werden. Ein weiterer Vorteil von Sockets ist, dass sie von vielen Programmiersprachen nativ unterstützt werden, was unterschiedliche Client Anbindungen potentiell ermöglicht.

Verworfen wird die Variante *Named Pipes*, da diese von Java nicht ohne Native Interface genutzt werden können. Die Varianten *Anonymous Pipes* und *Shared Memory* eignen sich nicht, da sie nicht in verteilten Applikationen verwendet werden können.

Mit diesem Entscheid nehmen wir in Kauf, dass ein Encoding definiert werden muss, damit die Daten von Client und Server interpretiert werden können.

Zur plattformunabhängigen Übertragung der Daten stehen drei verbreitete Datenformate zur Auswahl: JavaScript Object Notation (JSON), YAML Ain't Markup Language (YAML) und Extensible Markup Language (XML).

(WH)Y Statement *Auszeichnungssprache*:
Im Kontext des ADV für den AD-Unterricht an der HSR, konfrontiert mit dem Bedarf einer menschenlesbaren Auszeichnungssprache mit hoher Flexibilität (siehe 2.8), wird JSON als Dateiformat verwendet. Für JSON existieren in allen grossen Programmiersprachen entsprechende Produkte zum Parsen und Serialisieren des Formats. JSON hat im Gegensatz zu YAML ein kleineres Feature-Set und kann deshalb schneller verarbeitet werden.

Verworfen wird die Variante XML, da ein Schema den Modulentwickler zu stark einschränken würde. Obschon YAML weniger *verbose* ist wie JSON, wird auch auf YAML verzichtet, da dieses tendenziell eher für Konfigurationsfiles verwendet wird. Da YAML aber ein Superset von JSON ist, wäre eine Transformation nach YAML theoretisch möglich.

Mit diesem Entscheid nehmen wir in Kauf, dass keine Kommentare verwendet werden können. Dieses Features ist nur bei YAML und XML implementiert.

Das aus diesen Architekturentscheiden resultierende Kommunikations-Protokoll ist im Abschnitt 3.8.3 festgehalten.

3.8.3 Kommunikations-Protokoll

Die Abbildung 3.7 zeigt die Kommunikation zwischen der *ADV Lib* und dem *ADV UI* über die Socket. Jede JSON-Payload entspricht einem Snapshot. Mehrere Übertragungen entsprechen einer Session.

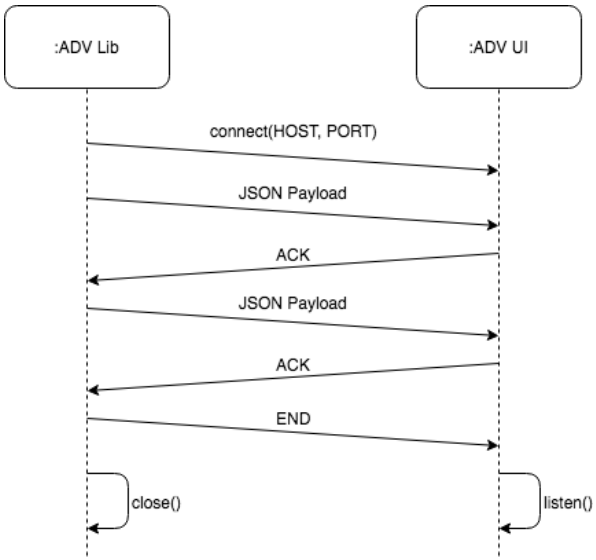


Abbildung 3.7: IPC: Sequenzdiagramm Request/Reply

Protokoll

Für den ADV wird ein minimales Protokoll mit spezifizierten Kommandos definiert. Das *TRANSMIT* Kommando beschreibt die Übertragung eines Snapshots. Der *END* Command signalisiert das Ende einer Übertragung (siehe Code-Listing 3.1).

Zur Bestätigung einer erfolgreichen Übertragung wird vom Socket-Server ein *ACK* Kommando gesendet. Solange die Socket Verbindung offen ist, werden Fehlermeldungen aus dem ADV UI mit dem *EXCEPTION* Kommando an die ADV Lib übermittelt und geloggt (siehe Code-Listing 3.2). Somit wird die Fehlersuche vereinfacht, da der Stacktrace des ADV UI auch in der User Codebase ersichtlich ist.

Listing 3.1: ADVRequest

```
{
    "command": "[TRANSMIT | END]",
    "json": {"JSON Payload"}        // optional
}
```

Listing 3.2: ADVResponse

```
{
    "command": "[ACK | EXCEPTION]",
    "exceptionMessage": "String"    // optional
}
```

JSON Schema

Im Gegensatz zu XML verwendet JSON kein fixes Schema, welches eingehalten werden muss. Modulentwickler können somit frei entscheiden, welcher Inhalt innerhalb einer ModuleGroup übertragen wird, da die *Stringifyer* und *Parser* in den Modulen implementiert werden. Damit die JSON Files der verschiedenen Modules jedoch einheitlich aufgebaut sind, wird die Verwendung des JSON-Schema aus Code-Listing 3.3 empfohlen.

Das Schema ist so ausgelegt, dass es sowohl für die Übertragung einzelner Snapshots sowie das Abspeichern ganzer Sessions mit mehreren Snapshots gebraucht werden kann. Zur Übertragung modul-spezifischer Inhalte wird der Bereich `elements.content` empfohlen. Theoretisch muss in einer *ModuleGroup* jedoch nur der `moduleName` gesetzt sein.

Genauere Definitionen der einzelnen Felder sind dem Benutzerhandbuch für Modul-Entwickler zu entnehmen (Anhang I).

Listing 3.3: Empfohlenes JSON Schema

```
{
    "sessionId": "Long",              // mandatory
    "sessionName": "String",
    "snapshots": [{
        "snapshotId": "Long",
        "snapshotDescription": "String",          // optional
        "moduleGroups": [{
            "moduleName": "String",               // mandatory
            "flags": [ "String" ],                // can be empty
            "elements": [{
                "elementId": "Long",
                "style": {
                    "fillColor": "String",
                    "strokeColor": "String",
                    "strokeStyle": "String",
                    "strokeThickness" : "String"
                },
                "fixedPosX": "Integer",       // optional
                "fixedPosY": "Integer",       // optional
                "content": {
                    ...                       // module-specific content
                }
            }],
            "relations": [{
                "sourceElementId": "Long",
                "targetElementId": "Long",
                "label": "String",            // optional
                "style": {                    // optional
                    "fillColor": "String",
                    "strokeColor": "String",
                    "strokeStyle": "String",
                    "strokeThickness" : "String"
                }
            }]
        }]
    }]
}
```

3.8.4 Technologieunabhängigkeit

Durch den Entscheid, Sockets für die IPC-Kommunikation zu verwenden (siehe 3.8.2), können in Zukunft zusätzliche Clients in einer alternativen Programmiersprache entwickelt werden, falls z.B die unterrichtete Programmiersprache der Fachhochschule wechselt. Infolgedessen muss nur der ADV Lib Container neu erstellt werden. Das ADV UI und das ADV Commons können übernommen werden.

3.9 Persistierung

Die übertragenen Sessions an das ADV UI sollen persistent abgespeichert werden können (siehe 2.9.5). Es ist zudem wichtig, dass der Datenaustausch zwischen mehreren Anwendern des ADV unkompliziert ermöglicht wird, damit beispielsweise Musterlösungen an die Studenten verteilt werden können. Zur persistenten Speicherung von Daten kommen gemeinhin zwei Varianten in Frage: Das Filesystem des Computers oder eine Datenbank (in den meisten Fällen NoSQL oder relationale Datenbank).

(WH)Y Statement *Datastore*:
Im Kontext des ADV für den AD-Unterricht an der HSR, konfrontiert mit dem Bedarf einer persistenten Speicherung der Session-Daten sowie einer einfachen Portabilität zum Austausch zwischen den Anwendern des ADV, wird das *Filesystem* als Datastore verwendet. Dieses erlaubt das effiziente Speichern und Laden von unstrukturierten Daten. Im Gegensatz zu einer Datenbank Technologie entsteht kein zusätzlicher Overhead, da das Filesystem ohne zusätzliche Technologien angesprochen werden kann. Ausserdem ist das Filesystem auf allen Geräten bereits verfügbar.

Verworfen wird die Variante *Datenbank*, da das Austauschen von ganzen Datenbanken signifikant aufwändiger ist, als bei einfachen Dateien.

Mit diesem Entscheid nehmen wir in Kauf, dass das Speichern einer Session nicht als atomare Transaktionen durchgeführt wird.

3.10 Prozesse und Threading

Damit der Zustand der Visualisierungen gemäss User Story 2 (siehe 2.9.1) auch nach dem Neustart der User Codebase nicht verloren geht, wird das ADV UI in einem eigenständigen Prozess modelliert. Damit läuft das ADV UI bis es manuell beendet wird.

Beim ADV wird das Client-Server Pattern [7] implementiert. Die User Codebase (Client) läuft typischerweise in einem Thread. Das ADV UI (Server) benötigt mehrere Threads, damit das GUI nicht einfriert. Für den Empfang von neuen Daten läuft der Socket Server in einem eigenständigen Thread, welcher bis zu seiner Terminierung auf eingehende Anfragen wartet (siehe Technische Anforderung *Parallelität* 2.9.5). Der Server Thread akzeptiert immer nur eine Anfrage. Weitere Anfragen werden von der Klasse *ServerSocket* gepuffert und seriell abgearbeitet. Unter JavaFX darf nur der UI-Thread Änderungen am GUI vornehmen [18]. Der UI-Thread von JavaFX wird automatisch beim Ausführen der Methode `Application.launch()` erstellt und läuft bis zur Terminierung der Applikation.

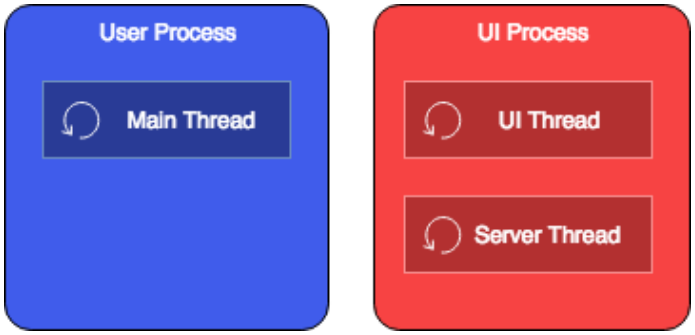


Abbildung 3.8: ADV Prozesse und Threads

3.11 Benutzeroberfläche

Bei der Entwicklung der Benutzeroberfläche wird auf das Erstellen eines interaktiven Wireframes verzichtet, da mit JavaFX relativ einfach ein UI erstellt werden kann, dass direkt für den Prototyp verwendet wird. Das Hauptaugenmerk liegt in der Einfachheit des GUI sowie bei einer guten Benutzerführung. Diese wird mit entsprechenden Usability-Tests überprüft (siehe 4.13.2).

3.11.1 Prototyp

In der Umfrage im Anhang E wird von den Benutzern des GVS ein intuitives GUI gefordert. Das ADV UI wird deshalb in zwei Sektionen unterteilt, mit dem Ziel vorhandene Funktionalitäten offensichtlicher anzuordnen.

Auf der linken Seite werden alle aktiven Sessions dargestellt, welche vom Benutzer betrachtet werden können. Durch einen Mausklick auf die Session wird diese geöffnet. Ebenfalls kann die Session aus der Liste entfernt, oder auf die Festplatte persistiert werden (vgl. 2.7.1).

Auf der rechten Seite sind die Sessions in eigenständigen Reitern zugreifbar. Sobald eine Session geöffnet wird, wird der Benutzer sogleich auf die Session-Toolbar aufmerksam, da diese am oberen Rand platziert ist. Die Toolbar enthält alle Interaktionsmöglichkeiten die mit der Session möglich sind. Über die Toolbar können die Snapshots einzeln und im persönlichen Lerntempo durchgesehen oder mit der Replay-Funktion in einer konfigurierbaren Geschwindigkeit abgespielt werden.

Am unteren Rand wird die Snapshot Beschreibung eingeblendet, da diese eine geringere Priorität als die Toolbar hat und nur optional befüllt ist.

In der Top-Toolbar sind die Buttons für globale Aktionen, wie das Laden einer bestehenden Session von der Festplatte, platziert. Ebenfalls gibt es einen Convenience-Button zum Löschen sämtlicher aktiven Sessions.

Durch die Verwendung eines verbreiteten Icon-Sets [24], kann der Benutzer die Aktion hinter einem Button gut erahnen. Ebenfalls helfen entsprechende Tooltips bei allfälligen Verständnisschwierigkeiten.

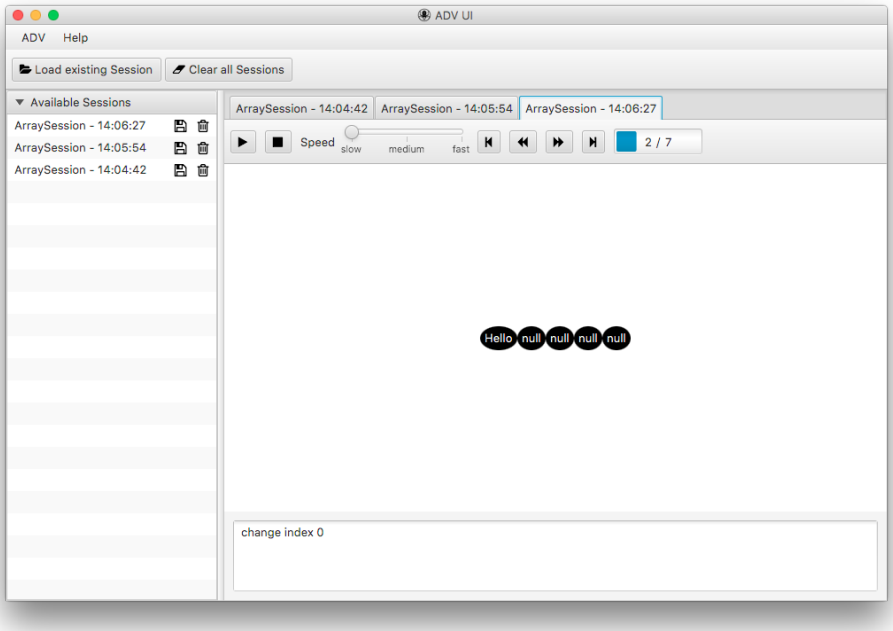


Abbildung 3.9: Prototyp ADV UI

3.11.2 Endgültige Version

Das GUI wird massgeblich durch die Resultate der Usability Tests beeinflusst. Die Protokolle der durchgeführten Tests befinden sich im Anhang F.2. Die Abbildung 3.10 zeigt das ADV UI in Version 1.0.

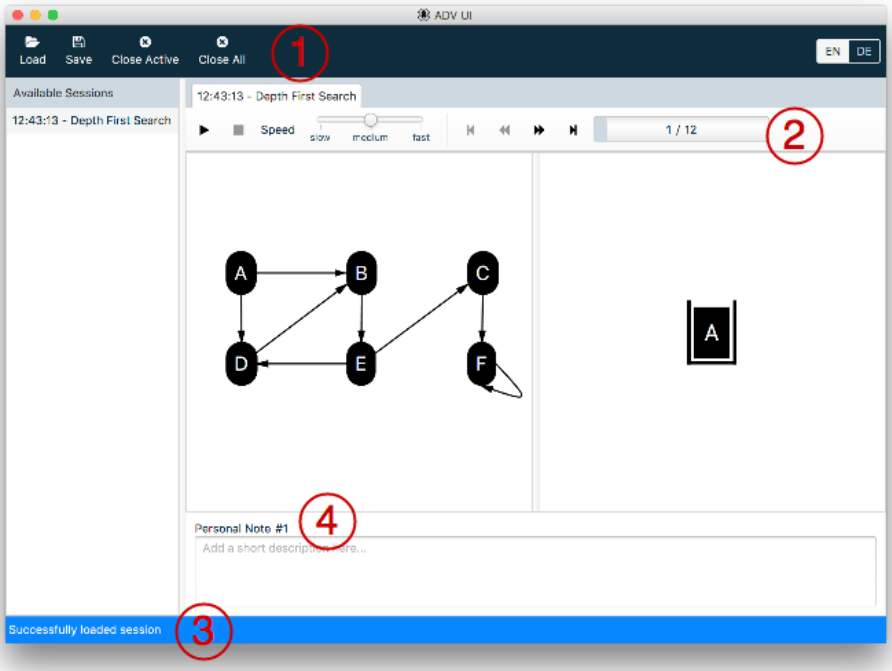


Abbildung 3.10: ADV UI Version 1.0

Folgende Änderungen wurden aufgrund des Usability-Feedbacks umgesetzt.

Button Bar
Damit die Funktionen auf einen Blick ersichtlich sind, befinden sich nun alle Buttons in einer der beiden Button Bars (1) und (2).

Statusleiste
Die Statusleiste (3) gibt Feedback über soeben durchgeführte Interaktionen wie zum Beispiel das erfolgreiche Speichern einer Session.

Snapshot Description
Das Label an der Snapshot Description Box (4) zeigt nun klar, dass eine Notiz immer zu einem Snapshot gehört.

Tooltips und Icons
Durch Tooltips sowie Labels an den Buttons in der Buttonbar (1) ist gut verständlich, welche Funktionalitäten hinter den Buttons stecken.

Shortcuts
Gängige Shortcuts erleichtern die Interaktion mit dem ADV. Eine Liste aller Shortcuts ist dem User Manual, Kapitel H.6.2 zu entnehmen.

Detachable Tabs

Das Vergleichen von mehreren Sessions ist eine der Anforderungen des ADV (vgl. User Story 6, Kapitel 2.9). Um diese Anforderung möglichst umfassend abzudecken, sind die Tabs im ADV *detachable*. Durch *Drag and Drop* können beliebig viele Tabs in eigene Fenster verschoben werden, ähnlich wie man das von Browsern kennt.

3.11.3 Logo

Das Logo wurde von den Eigenschaften des Kraken inspiriert [45]. Kraken sind bekannt dafür, dass sie Irrgarten-Probleme effizient lösen können. Dies ist eine Anspielung auf die Algorithmen, die vom ADV unterstützt werden. Ebenfalls wurden die Saugnäpfe des Kraken als Graph Nodes visualisiert und auf der Stirn ist ein Teil eines binären Trees zu erkennen.

Abbildung 3.11: ADV Logo

3.12 Framework

Im Abschnitt 2.2 sind die typischen Eigenschaften eines Frameworks beschrieben. Im ADV Projekt entspricht die Komponente *UI Core* dem Framework. Sie wird an diversen Stellen durch die Modules erweitert, behält aber stets die Kontrolle über den Programmfluss. Somit sind alle Teilschritte zur Visualisierung der Datenstrukturen für alle *Modules* gleich. Eine schematische Übersicht gibt die Abbildung 3.12. Sie zeigt alle beteiligten Services sowie die generierten Zwischenprodukte, die in der ADV Lib und dem ADV UI entstehen.

Abbildung 3.12: Kontrollfluss des Frameworks

Module In der User Codebase werden die modul-spezifischen Elemente instantiiert und mit der `snapshot` Methode an den *Builder* übertragen.

ModulGroup Der *Builder* verpackt das Module samt seinem Content in eine ModuleGroup und gibt diese dem *Stringifyer* weiter.

JSON Der *Stringifyer* serialisiert die erhaltenen Daten zu JSON. Das JSON wird anschliessend über eine Socket Verbindung an das ADV UI übermittelt.

ModulGroup Das JSON wird vom *SocketServer* empfangen und an einen modul-spezfischen *Parser* weitergeleitet. Dieser parsed das JSON und generiert entsprechende Domänenobjekte, welche vom *Layouter* weiter bearbeitet werden.

JavaFX Pane Der *Layouter* erstellt für alle Domänenobjekte entsprechende JavaFX Elemente. Diese werden auf einer Pane (JavaFX Container) optisch ansprechend positioniert.

Die Kontrollflusssteuerung der Teilschritte auf Seite des ADV UI übernimmt die Klasse *FlowControl*. Dabei werden modul-übergreifende Konzepte wie Sessions oder Snapshots von Klassen UI-Core verarbeitet. Modul-spezifische Konzepte werden über dynamische Polymorphie an die Module delegiert. Der genaue Ablauf ist im Abschnitt 3.12.1 erklärt.

3.12.1 FlowControl

Vom Empfangen der Daten über die Socket bis hin zur Darstellung der Daten im Presentation Layer sind einige Verarbeitungsschritte nötig. Die Klasse *FlowControl* ist zuständig, dass diese Schritte von Core-Klassen oder vom richtigen Module in einer vorgegebenen Reihenfolge durchgeführt werden.

Ausgelöst wird dieser Ablauf durch das Eintreffen von Daten über die Socket oder das Laden von Sessions aus dem *Datastore*. Mit Hilfe von Gson [33] werden die eintreffenden Daten geparsed. Das Sequenzdiagramm FlowControl

im Anhang D zeigt die nötigen Schritte sowie die beteiligten Klassen am Beispiel des *Array Modules*.

3.12.2 Strategy Pattern

Die von den beiden Core Komponenten (Core UI und Core Lib) vorgegebenen Interfaces bilden das Kernstück der Applikation. Hier wird das Strategy Pattern umgesetzt [63]. Jedes *Module* stellt pro Interface die entsprechende *Strategy* Implementierung bereit (siehe Abbildung 3.13). Dadurch arbeiten die Cores nur mit der Abstraktion und haben keine Kenntnisse der einzelnen Implementierungen, was zu einer tiefen Koppelung führt.

Für den *Stringifyer* gibt das Framework eine Standard-Implementierung vor, da dieser Service in den meisten Fällen über alle Module identisch implementiert wird. Die Module Array, Queue und Stack verwenden den *Default Stringifyer*. Das *Graph Module* besitzt eine eigene *Stringifyer* Strategie, um zukünftigen Modulentwicklern eine Beispiel-Implementation zu unterbreiten.

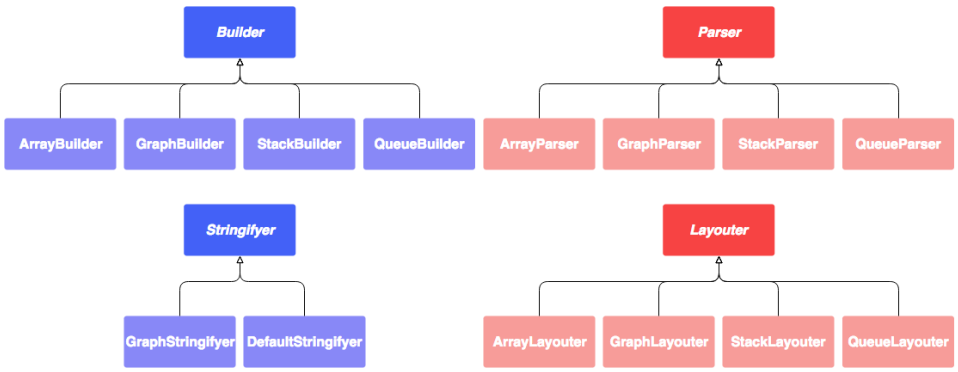


Abbildung 3.13: Strategy Pattern Implementierung durch Modul-Services

3.12.3 *Module* Bootstrapping

Um die Integration neuer Module zu vereinfachen, müssen die Modul-Services aus Abschnitt 3.12.2 mit dem Modulnamen annotiert werden. Annotierte Services werden automatisch per Reflection ausgelesen und im DI Container Guice [34] registriert.

Für die Registrierung wird das Guice Feature *MapBinder* [35] verwendet. MapBinder wurden speziell für die Verwendung in verschiedenen Modulen konzipiert. Die API erlaubt das sukzessive befüllen einer globalen Map aus mehreren Modulen, um diese dann später als Ganzes verwenden zu können.

Für jeden Modul-Service aus dem Abschnitt 3.12.2 existiert eine eigenständige Map, wessen Einträge durch die Modulnamen identifiziert werden. Die

Maps werden in der Klasse *Bootstrapper* befüllt und in der Klasse *ServiceProvider* injected.

Sobald der Dependency Injection Container vollständig initialisiert ist, können die Module-Services in der ganzen Applikation via *ServiceProvider* bezogen werden.

Der genaue Ablauf ist im Sequenzdiagramm Bootstrapping im Anhang D abgebildet.

3.12.4 *Child Modules*

Eine Session beinhaltet eine variable Anzahl Snapshots. Innerhalb eines Snapshots können mehrere *Modules* angesprochen werden (siehe Abbildung 2.2 im Kapitel *Ubiquitous Language*). Ein Module kann mehrere *ChildModules* besitzen, welche zur Visualisierung von Hilfsdatenstrukturen verwendet werden. Der *Builder* in der ADV Lib erstellt für jedes Child-Module eine eigenständige ModuleGroup.

Darstellung

Multimodules (Modul mit einem oder mehreren *Child Modules* 3.12.4) werden mit JavaFX SplitPanes voneinander getrennt. Jedes Modul wird somit in einem eigenständigen Container dargestellt, dessen Grösse vom Benutzer dynamisch verändert werden kann.

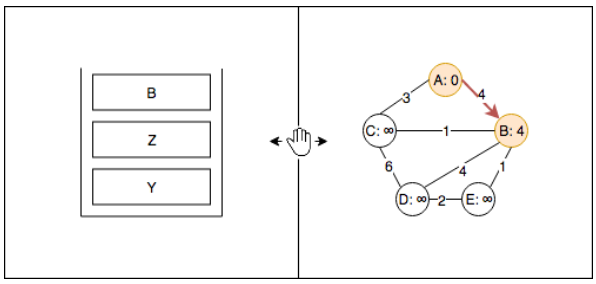


Abbildung 3.14: Tiefensuche: Graph-Module mit Stack-Module als Kind im Split Screen

3.12.5 Framework Events

Gemäss User Story 27: Hooks (siehe 2.9.1) soll ein *Module* in den Ablauf des Frameworks eingreifen können. Um dies zu gewährleisten stehen zwei Varianten zur Verfügung.

1. Hook Methoden
2. Events

(WH)Y Statement *Events*:
Im Kontext des ADV für den AD-Unterricht an der HSR, konfrontiert mit dem Bedarf eines erweiterbaren Frameworks gemäss User Story 27, wird die oben aufgelistete *Variante 2* gewählt. Diese Methode stellt dem Modulentwickler frei, ob er einzelne Teilschritte modul-spezifisch erweitern möchte oder nicht. Intern werden dazu Property Change Events von Java verwendet, die das Observer Pattern implementieren.

Verworfen wird die *Variante 1* , da sonst alle Modulentwickler gezwungen wären, die vordefinierten Hook-Methoden zu implementieren, auch wenn sie diese nicht benötigen.

Aus diesem Architekturentscheid ergibt sich folgender Ablauf:

- *Module* registriert sich für einen Event
- *Module* wir durch den Event über Änderungen notifiziert
- *Module* kann durch die Übergabe des alten und neuen Zustandes Einfluss nehmen

Genauere Informationen zu den angebotenen Events sowie deren Nutzung sind im *Module-Manual* I.8.4 festgehalten.

Im *SessionStore* und *LayoutedSnapshotStore* führt das Hinzufügen von Daten zum Auslösen eines Property Change Events, welcher den Presentation Layer über neue Daten informiert. Dieser Event fürht zum einem Neu-Zeichnen des GUIs. Der Vorgang ist im Sequenzdiagramm EventManager im Anhang D genauer dargestellt.

3.12.6 Widgets

Zur Unterstützung der Modulentwickler und zur Vermeidung von Duplicated Code, stellt das Framework Widget Klassen zur Verfügung. Die Widgets können von allen Modules verwendet werden. Beispiele für ein angebotenes Widget ist ein Knoten mit einem zentrierten Text oder eine automatisch zentrierende und zoomende *Pane*.

Eine Auflistung und Nutzungsbeispiele zu allen vorhandenen Widgets findet sich im Abschnitt I.8.1

3.13 Modules

Gemäss Aufgabenstellung (siehe Anhang A) sind zwei Modules im Umfang dieser Arbeit inbegriffen. Zur Auswahl stehen die Modulideen aus der Anforderungsspezifikation (siehe Abschnitt 2.8).

Die Auswahl wird so getroffen, dass der Funktionsumfang des Frameworks möglichst vielfältig überprüft werden kann. Beim ersten Modul geht es des-

halb darum, den grundlegenden Ablauf des Frameworks zu überprüfen, ohne viel Zeit in einen möglichst grossen Modulumfang zu investieren. Beim zweiten Modul soll dann die Komplexität erhöht werden, um die Flexibilität des Frameworks weiter zu überprüfen.

3.13.1 Array Module

Dieses Modules hat zum Ziel, eine grundlegende Datenstruktur zu visualisieren sowie erste Erfahrungen mit der Framework-Schnittstelle zu sammeln.

Das *Array Module* bietet sich an, da es graphisch wenig Komplexität aufweist und somit der Fokus auf der Ausarbeitung der Framework-Konzepte liegt.

Die Architektur des *Array Modules* ist mehrheitlich durch die Designentscheide aus der Core Architekturphase vorgegeben. Entsprechend wird je ein modul-spezifischer Builder und Stringifyer in der ADV Lib, und ein Parser, Stringifyer und Layouter im ADV UI erstellt. Zudem werden im *Array Modul* mehrere Layouter benötigt (siehe 3.13.1).

Layouter Design
Gemäss der User Story 31 (siehe Abschnitt 2.9) soll es möglich sein, die Array-Layouts unterschiedlich zu berechnen. Konkret unterscheidet sich die Darstellung von Arrays mit und ohne Objektreferenzen. Damit die beiden Ausprägungen unterschiedlich dargestellt werden, delegiert der *ArrayLayouter* den Aufruf an seine "Kind-Layouter".

Layouter
Alle Arrays, die von diesem *Module* dargestellt werden können, sind Object-Arrays. Primitive Arrays können nicht dargestellt werden, da *Primitives* in Java nicht mit *Generics* genutzt werden können. Das Darstellen von Objektreferenzen stört aber bei der Veranschaulichung von Algorithmen. Deshalb werden Arrays per Default ohne Objekt-Referenzen dargestellt.

Damit die verschiedenen Layouter angesprochen werden können, sind auf Stufe ModuleGroup modul-spezifische Flags einsetzbar (Siehe Abschnitt 3.8.3). Dieser Mechanismus wird genutzt, um Algorithmen zu layouten, die nicht der Standard Array Repräsentation entsprechen. Die Flags werden in der User Codebase je nach Algorithmus gesetzt und von der ADV Lib übertragen. Anschliessend kann die Layout-Aufforderung vom *ArrayLayouter* an den entsprechenden "Kind"-Layouter dispatched werden.

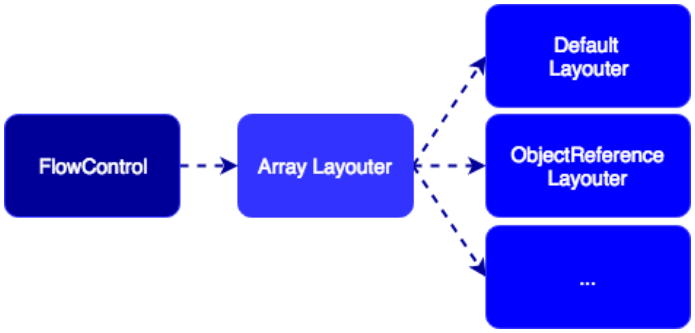


Abbildung 3.15: Layouter-Konzept im *Array Module*

Tabelle 3.4 zeigt, welche Konzepte aus den *Module*-Ideen (2.8.1) umgesetzt werden.

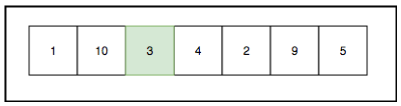
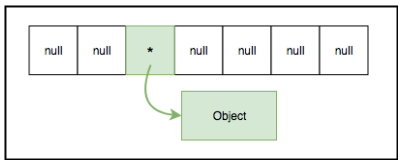
	DefaultLayouter	ObjectReferenceLayouter
Flag	*keines*	`SHOW_OBJECT_RELATIONS`
Resultat		

Tabelle 3.4: Layouter-Varianten im *Array Module* und die unterschiedlichen Darstellungen

3.13.2 DFS/BFS Module

DFS und BFS sind Graph-Traversierungs-Algorithmen, welche neben einem Graphen jeweils eine Hilfs-Datenstruktur zur Berechnung verwenden. Das *DFS/BFS Module* bietet sich also an, da es gleichzeitig zwei Modules anspricht. Zudem sind Graphen grafisch deutlich anspruchsvoller als Arrays.

Diese erhöhte Komplexität ermöglicht einen Härte-Test für das Framework, wodurch allfällige Schwachstellen aufgedeckt werden können.

Das *DFS/BFS Module* wird nicht als eigenständiges *Module* umgesetzt. Stattdessen werden die benötigten Datenstrukturen als eigenständige *Modules* entwickelt. Dank der *Child Module* Funktionalität können die beiden Algorithmen ohne zusätzlichen Code dargestellt werden (siehe 3.12.4).

Die Architektur der drei *Modules* - Graph, Stack und Queue - ist durch die Designentscheide aus der Core Architekturphase vorgegeben und ist analog zur Architektur des *Array Modules*.

Kapitel 4

Projektplanung

Datum	Version	Änderungen	Autor
22.02.18	0.01	Grundentwurf Projektplan erstellt	mtrentini
23.02.18	0.02	Risikomanagement beschrieben	mtrentini, mwieland
05.03.18	0.03	Milestones definiert	mtrentini
02.04.18	0.04	Entwicklungskonzept	mwieland
04.04.18	0.1	Review Dokument	mwieland, mtrenini
04.06.18	0.11	Dependencies ergänzt	mtrentini
11.06.2018	1.0	Review Dokument	mwieland, mtrenini

Tabelle 4.1: Versionshistory Projektplanung

4.1 Zeitplanung

Das Projekt wird im Rahmen der Bachelorarbeit durchgeführt. Insgesamt stehen 16 Wochen zur Verfügung in welchen jedes Teammitglied 360 Stunden leisten muss. Somit entstehen ca. 20 Stunden Arbeitsaufwand pro Woche und Teammitglied für die ersten 14 Wochen des Projekts. Für den Abschluss des Projekts verbleiben 40 Stunden pro Woche und Teammitglied während zwei Wochen.

Datum	Beschreibung
19.02.2018	Start der Bachelorarbeit
26.04.2018	Zwischenpräsentation
15.06.2018	Abgabetermin
29.06.2018	Schlusspräsentation

Tabelle 4.2: Wichtige Termine

4.2 Projektverwaltung

Als Projektmanagement Software wird Jira [41] eingesetzt. In Jira werden alle Requirements als Issues erfasst und in den Product Backlog eingepflegt. Pro Iteration sollen jeweils so viele Issues eingeplant werden, wie unter Berücksichtigung von administrativen Aufgaben abgearbeitet werden können. Dabei spielt der Teamspeed eine grosse Rolle, welcher sich über die Projektdauer einpendeln soll.

4.2.1 Iterationsplanung

Die Iterationsplanung orientiert sich grob am Rational Unified Process (RUP) und ist unterteilt in eine Inception-, Elaboration-, Construction- sowie eine Transition-Phase. Jede Phase besteht aus einer oder mehreren Iterationen, die jeweils 1 Wochen dauern und am Freitag enden. Die Sprints sind bewusst nur 1 Woche lang, um eventuell ändernden Anforderungen gerecht zu werden. Zudem bringen die Wochenmeetings (siehe 4.4) jeweils neuen Input, welcher dann direkt in den nächsten Sprint einfliessen kann.

Eine Iteration wird nach SCRUM organisiert. Am Anfang des Sprints definiert das Projektteam die zu erledigenden Issues und schätzt deren Aufwand (siehe 4.2.2). Jeweils am Ende eines Sprints wird die geleistete Arbeit sowie die Stimmung im Team reflektiert. Entsprechende Erkenntnisse und Verbesserungen werden fortlaufend in die Projektorganisation aufgenommen. Durch die Nutzung von Continuous Integration und das Arbeiten mit Feature-Branches, besteht auf dem Develop-Branch am Ende jedes Sprints eine lauffähige Applikation.

4.2.2 Schätzungen

Issues werden auf Basis von Story Points geschätzt. Dieses Vorgehen hat sich mit SCRUM etabliert. Die Nutzung von Story Points führt dazu, dass nicht die individuell unterschiedliche Bearbeitungszeit, sondern die Komplexität

einer User Story geschätzt wird. Dies vereinfacht und homogenisiert die Schätzungen und hilft den Teamspeed zu bestimmen.[62, 66]

4.2.3 Zeitauswertung

Für die Zeitauswertung wird das Jira Plugin Tempo [12] verwendet. Dieses bietet umfassende Auswertungsmöglichkeiten, sowie Exports nach MS Excel. Die Reports zur geleisteten Zeit befinden sich im Anhang J.

4.3 Meilensteine

Meilenstein 1: Anforderungsspezifikation

Der Meilenstein 1 hat zum Ziel, die Anforderungen an den ADV zu ermitteln. Die Anforderungsspezifikation umfasst bei erreichen des Meilenstein 1 folgende Punkte:

- Die Stakeholder und Aktoren sind erfasst.
- User Stories zeigen den funktionalen Umfang des Frameworks. Für Abläufe, die der Benutzer über das GUI einleitet, bestehen casual Use Case Beschreibungen.
- NFRs sind wo immer möglich SMART erfasst.
- Die Definition der Ubiquitous Language steht fest.
- Die Analyse von Konkurrenzprodukten ist durchgeführt.
- Die groben Konzepte der Modul-Ideen sind aufgestellt.
- Die Risiken wurden identifiziert und Eskalationspläne definiert

Funktionale und nicht-funktionale Anforderungen können in späteren Iterationen noch angepasst oder erweitert werden.

Meilenstein 2: Design- und Architekturspezifikation

Das Ziel des zweiten Meilensteins ist es, die Architektur und das Design des ADV zu gestalten. Es sind Modelle von verschiedenen Abstraktionsstufen des Systems zu erstellen. Diese Modelle können sich in nachfolgenden Iterationen noch verändern, bieten aber eine erste Grundlage für die Implementierung.

Konkret sind folgende Punkte zu erfüllen:

- Ein Context, ein Container und ein Component Diagramm des C4 Modells bestehen.
- Architekturentscheide sind mit Y-Statements begründet.

Meilenstein 3: Prototype

Im Meilenstein drei werden die Entwürfe aus Meilenstein 2 erstmals grob in einem oder mehreren Prototypen umgesetzt. Es gilt zu prüfen, ob die angedachten Konzepte wie geplant umsetzbar sind.

Ziele des dritten Meilensteins:

- Erstellen von Prototypen
- Erstellen des C4: Code Diagramms
- Bereitstellen der Grundlage für Unit-Tests
- Evaluation von Drittanbieter Libraries zur Erfüllung der technischen Anforderungen

Meilenstein 4: Core Framework und erstes *Module*

Mit der Umsetzung des Frameworks und der Implementierung eines ersten Modules ist der vierte Meilenstein erreicht. Es sind alle hoch priorisierten User Stories des Frameworks und des *Modules* umgesetzt (vgl. Abschnitt 2.9.1). Die umgesetzte Funktionalität ist mit Unit-Tests abgedeckt.

Meilenstein 5: Core Framework und zweites *Module*

Das Erreichend des fünften Meilensteins benötigt die Implementierung eines weiteren *Modules*. Mindestens die hoch priorisierten User Stories des *Modules* sind umgesetzt. Das Framework wird wo nötig angepasst und erweitert. Alle mittel und möglichst viele der tief priorisierten User Stories des Frameworks sind umgesetzt (vgl. Abschnitt 2.9.1). Die umgesetzte Funktionalität ist mit Unit-Tests abgedeckt und Usability Tests sind durchgeführt.

Meilenstein 6: End of Project

Das Ziel des letzten Meilensteins ist, die Dokumentation abzuschliessen, die endgültigen Software Artefakte auszuliefern und alle administrativen Aufgaben zu vollenden.

4.3.1 Artefakt Übersicht

Eine Übersicht aller in dieser Arbeit entstehenden Artefakte finden sich im Anhang B. Neben einem Zeitplan ist der grobe Umfang aller Artefakte und Versionen gelistet.

4.4 Meetings

Über die gesamte Projektdauer findet jeweils am Donnerstag um 15:15 ein wöchentliches Standortmeeting statt. Die Beschlüsse aus den Meetings werden protokolliert und bis spätestens 24h nach Abschluss des Meetings an alle Teilnehmer versendet. Allfälliges Feedback wird nachträglich eingepflegt und versioniert abgelegt.

4.5 Verantwortlichkeiten

Da in der Diskussion bessere Architekturen entstehen, wird die Analyse und Design Phase von beiden Projektmitglieder gleichermassen durchgeführt. Während der Implementierung sind die Verantwortlichkeiten nach dem agilen Konzept «self organizing teams» verteilt. Somit werden die persönlichen Stärken der beiden Teammitglieder besser genutzt. Die genaue Arbeitsteilung ist der git-History, den Java-Docs sowie den Änderungshistories der Projektdokumentation zu entnehmen.

4.6 Repositories

Die Artefakte werden in verschiedenen Repositories auf GitHub versioniert abgelegt. So gibt es für die Dokumentation und den Sourcecode jeweils ein eigenes Repository.

Repository	Art	URL
adv-thesis	Dokumentation	`https://github.com/michiwieland/adv-thesis`
ADV UI	Quellcode	`https://github.com/ADVisualizer/ADV-UI`
ADV Lib	Quellcode	`https://github.com/ADVisualizer/ADV-Lib`
ADV Commons	Quellcode	`https://github.com/ADVisualizer/ADV-Commons`
ADV User Codebase	Quellcode	`https://github.com/ADVisualizer/ADV-User_Codebase`
ADV Starter Gradle	Quellcode	`https://github.com/ADVisualizer/ADV-Starter-Gradle`

ADV Starter Maven	Quellcode	`https://github.com/ADVisualizer/ADV-Starter-Maven`

Tabelle 4.3: ADV Repositories

4.7 Entwicklungs Konzept

4.7.1 Branch Struktur

Die Branch Struktur orientiert sich an Gitflow [28]. Pro Issue wird ein Feature Branch erstellt, der nach erfolgreichem Review (siehe 4.7.5) in den Development Branch gemerged wird. Beim Erreichen eines Meilensteins (siehe 4.3) wird der Entwicklungszweig in den Master Branch gemerged.

4.7.2 Definition of Done

Source Code wird erst zum Review freigegeben, wenn folgende Kriterien erfüllt sind.

- Die Integrated Development Environment (IDE) rsp. das Build Tool zeigt keine Warnungen und Fehler
- Es gibt keinen auskommentierten Code
- Es existieren sinnvolle Unit- und Integrationstests für das Feature
- Alle Metriken geben grünes Licht
- Das Issue wurde in Jira [41] zum Review vermerkt

4.7.3 Buildprozess

Zur Erstellung der Software Artefakte wird Gradle [30] eingesetzt. Gradle verwaltet alle externen Abhängigkeiten und garantiert einen einheitlichen Buildprozess. Zusätzlich wird die Software nur dann gebuildet, wenn alle Metriktools grünes Licht geben (siehe 4.10). Es resultiert ein ausführbares JAR.

4.7.4 Continuous Integration

Die Software wird nach jedem Push ins Github Repository mit Travis CI [68] gebuildet. Dies garantiert, dass sämtliche Qualitätsmassnahmen eingehalten werden und ermöglicht eine frühzeitige Erkennung von Integrationsproblemen.

4.7.5 Review

Nach Abschluss eines Issues wird dieses an den Teampartner zum Review übergeben. Durch das Vier-Augen-Prinzip kann die Qualität des Produktes hoch gehalten werden. Zusätzlich haben beide Teampartner stets den selben Wissensstand.

4.8 Backups

Zur Minimierung von allfälligen Datenverlusten wird wie folgt vorgegangen:

1. Täglich automatisiertes Backup aller Projektdaten im JIRA [41] (Zeiterfassung, erstellte Issues, JIRA Konfiguration)
2. Die Projektdokumentation sowie der Programmcode wird in den vier Github Repositories der Github Organisation [27] versioniert abgelegt (siehe Tabelle 4.3). Für die Projektmitglieder gilt der Grundsatz "Commit early and often".
3. Weitere Artefakte werden auf Dropbox [22] abgelegt und dort automatisch gesichert und versioniert.

4.9 Risikomanagement

Nachfolgend sind die wichtigsten Risiken für diese Arbeit aufgelistet. Gewichtet werden die Risiken nach Eintrittswahrscheinlichkeit und Schadenshöhe. [56] Auf eine Schätzung bezüglich dem zeitlichen Mehraufwand wird bewusst verzichtet, da diese erfahrungsgemäss sehr ungenau ist und wenig praktischen Nutzen hat. Dennoch erzwingt die Liste ein aktives Auseinandersetzen mit den möglichen Risiken und zeigt Wege für die Eskalation rsp. Reaktion beim Eintritt eines Risikos.

#	Eintrittswahrscheinlichkeit (E)	Schadensschwere (S)
1	gering	leicht
2	mittel	mittelschwer
3	hoch	schwer

Tabelle 4.4: Legende Risiken

#	Beschreibung	E	S	Prävention	Massnahmen beim Eintritt
1	Verlust von Daten aufgrund von technischen Störungen oder Diebstahl von persönlichen Notebooks	2	1	Backups (siehe 4.8). Dadurch muss höchstens ein Datenverlust von 8h aufgearbeitet werden. «Commmit early and often»	Backup wiederherstellen
2	Der angedachte Projektumfang übersteigt die Zeit, welche für diese Bachelorarbeit eingeplant ist.	1	3	Gute Planung und frühzeitiges Abstecken des Scopes	Rücksprache mit dem Betreuer
3	Eingesetzte Frameworks, Libraries und Cloud Services harmonieren nicht wie angenommen oder unterstützen die eingesetzte Java Version nicht	2	2	Auf einen Software Stack setzen, der beliebt und weit verbreitet ist. Dies reduziert das Risiko, dass der gegenseitige Support fehlt.	Alternativen für inkompatible Services finden
4	Aufgrund von wachsenden Anforderungen seitens der Modulentwickler wächst das Core Framework zu einem unübersichtlichen Klumpen an Sourcecode (Code Bloat).	2	2	Klare Aufgabenteilung zwischen Framework und Modulen definieren und dokumentieren. Framework Umfang mit Referenzmodulen überprüfen	Verschieben von Funktionalität zwischen Core Framework und Modulen

5	Die Komplexität der Schnittstellen ist so hoch, dass die Verwendung des Frameworks trotz Dokumentation sehr schwer ist.	2	3	Implementieren der Referenzmodulen («Eat Your Own Dog Food»). Frühes erstellen von Prototypen. Ausführliche und verständliche Architekturdokumentation sowie Anleitung für die Entwicklung von Modulen.	Überdenken des Kontrollflusses mit Hauptaugenmerk auf Einfachheit
6	Das Core Framework ist starr in seinem Ablauf und zu wenig offen für Erweiterungen	2	3	Ausführliche Analyse der potentiellen Module. Anbieten von Hook-Methoden.	Erweitern des Core Frameworks
7	Die Lernkurve bei der Verwendung der ADV Library ist für den Benutzer zu steil.	1	2	Ausführliches und verständliches Benutzerhandbuch	Verbessern der Dokumentation sowie praktisches Vorführen durch den Übungsbetreuer
8	Ein spezifisches Modul lässt sich nicht mit dem entwickelten Framework umsetzen.	2	3	Frühzeitiges Überlegen des Umfangs von möglichen Modulen	Anpassung des Framework oder verändern des Modulumfangs

Tabelle 4.5: Risikoliste

4.10 Qualitätsattribute

Um die Softwarequalität hoch zu halten, werden eine Reihe von Qualitätsmassnahmen eingeführt, die den Entwicklungsprozess aktiv beeinflussen. Eine Übersicht über die Metriken zur Qualität des Produktes ist im Anhang G ersichtlich.

4.10.1 Travis CI

Travis CI [68] ist für das Continuous Integration verantwortlich. Travis automatisiert den Gradle Build sowie die Ausführung der Unit Tests. Somit können Integrationsprobleme rasch erkannt und behoben werden. Travis CI ist für OpenSource Projekte frei verfügbar.

4.10.2 Codacy

Als Metrikdashboard wird Codacy [19] verwendet, dass für OpenSource Projekte gratis verwendet werden darf. Codacy prüft sämtliche Pull-Requests auf Ihre Konformität mit den aufgesetzten Checkstyle Regeln.

4.10.3 Checkstyle

Für das ADV Projekt werden Coding Guidelines in Form von Checkstyle [15] Regeln erstellt. Ein Pull-Request wird erst akzeptiert, wenn keine Checkstyle Regeln verletzt werden. Die Guidelines orientieren sich an den offiziellen Java Coding Conventions von Sun [17], wurden jedoch an einigen Stellen leicht angepasst. Das angepasst Checkstyle XML ist im Root-Folder der beiden Projekte abgelegt (siehe 4.6).

4.10.4 Spotbugs

Spotbugs [61] dient der statischen Code-Analyse und erkennt gefährliche Code Konstrukte. Es wird in den Gradle Build integriert. Der Build läuft nur durch, wenn Spotbugs keine Fehler findet.

4.10.5 Structure101

Structure101 [64] wird zur Analyse der Package-Abhängigkeiten innerhalb des Projekts eingesetzt. Mit ihm lassen sich Verletzungen der angedachten Architektur gut erkennen und beheben. Structure101 ist für OpenSource Projekte frei verfügbar.

4.10.6 Jacoco

Jacoco [39] wird für die Erstellung von HTML und XML Reports der Testabdeckung verwendet. Zusätzlich wird ein Gradle Task geschrieben, der die

Coverage Reports aus allen Teilprojekten aggregiert und als einziges File im Root Projekt ablegt. Damit kann der aggregierte Report nach jedem CI Build auf das Metrikdashboard hochgeladen werden.

4.11 Exception Handling

Exceptions werden sofern möglich lokal behandelt und mit einer entsprechenden Fehlermeldung sowie dem gesamten Stacktrace geloggt (siehe 4.12). Exceptions die während der Übertragung zwischen der ADV Lib und dem ADV UI auftreten, werden an die ADV Lib übertragen und dort geloggt.

4.12 Logging

Gemäss der Anforderungsspezifikation 2.9.5 soll ein intensives Logging betrieben werden. Als Logging Facade wird SLF4J [57] mit der Logging Implementation Logback [48] eingesetzt. Logback ist der Nachfolger des verbreiteten Framework Log4J [47]. SLF4J wird so konfiguriert, dass täglich ein Log File erstellt wird, dass laufend ergänzt wird. Zusätzlich werden alle Log Einträge in der Standardausgabe (STDOUT) ausgegeben. Das Log-Level kann statisch in einem XML File oder dynamisch zur Laufzeit mit der JConsole [40] konfiguriert werden.

4.13 Testing

Testing ist ein unerlässlicher Baustein um den hohen Qualitätsanforderungen einer robusten Software zu genügen. Für das ADV Projekt werden drei verschiedene Tests durchgeführt. Protokolle über die durchgeführten Tests sind im Anhang F und G einsehbar.

4.13.1 Unit Tests

Unit Tests testen die Methoden einer isolierten Klasse. Sie sollen systematisch, wiederholbar und automatisiert sein sowie genau einen Ausführungspfad testen. Jede Testmethode ist nach dem Arrange (Given), Act (When), Assert (Then), Teardown Prinzip organisiert. Abhängigkeiten werden mit dem Mocking Framework Mockito [51] weg abstrahiert. Beim Schreiben der Tests wird der Schwerpunkt auf Qualität anstatt Quantität gesetzt. Insbesondere werden die Framework Komponenten ausgiebig getestet, da ihre fehlerfreie Ausführung für die Modulentwickler von höchster Wichtigkeit ist.

JUnit

Für das automatische Ausführen des Testcodes wird auf den Java Standard JUnit [44] gesetzt.

Jukito

Jukito ist ein TestRunner für JUnit, der Abhängigkeiten automatisch mit Mockito [51] mockt und Dependency Injection von Guice [34] unterstützt. Jukito vereint somit alle Anforderungen an das Unittest-Setup.

Hamcrest

Java Hamcrest [36] erweitert die JUnit und Jukito Library mit *Matchern* so können einfacher Test geschrieben werden, welche beispielsweise überprüfen, ob eine Methode mit einem gewissen Input aufgerufen wurde.

TestFX und Monocle

Damit Klassen mit JavaFX-Abhängigkeiten getestet werden können, muss das JavaFX-Toolkit initialisiert werden. Hierzu wird die Library TestFX [65] verwendet. Damit diese Tests auch auf Travis laufen, wird ein Headless-Build mit Hilfe von Monocle [52] durchgeführt

4.13.2 Usability Tests

Usability Tests werden mit nicht vorbelasteten Testpersonen etwa in der Hälfte, sowie am Ende des Projektes durchgeführt. Den Probanden werden mehrere Szenarien vorgelegt, welche sie intuitiv lösen sollen. Dabei werden die Hilfestellungen minimal gehalten. Das Verhalten der Benutzer wird protokolliert und mögliche Erkenntnisse niedergeschrieben.

4.13.3 System Tests

Systemtests testen das Gesamtsystem nach der Erfüllung der gestellten Anforderungen. Systemtests werden nach Erreichung der beiden Meilensteinen in der Construction Phase durchgeführt, damit Modules nicht auf den Fehlern der Vorgänger aufbauen.

4.14 Dependencies

Der ADV nutzt folgende Libraries von Dritt-Anbietern:

Guice

Guice [34] ist ein Dependency Injection Framework von Google. Sämtliche Abhängigkeiten von ADV Klassen auf Services werden im ADV über den Konstruktor übergeben, was die Kopplung verringert und die Testbarkeit der Applikation verbessert.

Reflections

Um Dependency Injection von *Module* Klassen zur Laufzeit zu unterstützen, nutzt der Bootstrapper Reflection (siehe 3.12.3). Dies wird durch die Reflections Library [55] unterstützt.

Gson

Gson [33] ist eine Java Library zum Serialisieren und Deserialisieren von Java Objekten nach JSON und zurück. JSON wird zur Übertragung von Daten zwischen der ADV Lib und dem ADV UI verwendet.

ControlsFX

Die Library ControlsFX [21] erweitert JavaFX um zusätzliche UI-Elemente. Für das ADV UI werden die beiden Elemente *Statusbar* und *SegmentedButton* genutzt.

Teil III

Anhang

Anhang A

Aufgabenstellung

Framework zur Visualisierung von Algorithmen und Datenstrukturen

Studenten

- Murièle Trentini
- Michael Wieland

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 Diplomarbeit und einer Studienarbeit der Graphs Visualization Service (GVS) entwickelt. Dieser unterstützt die Studierenden beim Erlernen der Datenstrukturen Graph und Tree sowie einiger Algorithmen aus der Graphentheorie durch Visualisierung.
Der GVS ist allerdings in der Darstellung auf Graphen und Trees limitiert und die angedachte Architektur sieht keine grösseren Erweiterungen am Funktionsumfang vor. Zudem würde ein Refactoring der bestehenden Architektur zu einem modularen Framework einer Neuentwicklung gleichkommen.

Aufgabenstellung

Das Ziel dieser Bachelorarbeit ist die Entwicklung eines modularen Frameworks, welches die verschiedenen Informatikkonzepte aus "Algorithmen und Datenstrukturen 1 und 2" in eigenständigen Modulen abbilden kann.

Folgende Fragestellungen sind dabei zu untersuchen:

- Welche Gemeinsamkeiten können im Core Framework vereinigt werden. Welche Funktionalitäten müssen in den Modulen implementiert werden?

- Wie können die verschiedenen Informatikkonzepte visualisiert werden, damit der Lernerfolg der Studierenden verbessert werden kann?

- Wie weit sollen die Studenten beim ausprogrammieren von Informatikkonzepten unterstützt werden?
  Welche Klassen resp. API sollen vom Framework bereitgestellt werden?

Technologisch sollen dafür Java und JavaFX verwendet werden. Zusätzlich sollen geeignete Frameworks, Technologien und Tools evaluiert werden, welche die Entwicklung weiterer Module vereinfachen.

Die Arbeit hat folgende spezifische Ziele:

- Das Hauptziel der Arbeit ist die Erstellung einer erweiterbaren Softwarearchitektur, sowie zweier Module, welche als Referenzimplementierung herangezogen werden können.

- Potentielle Module sind:
  - Visualisierung der grundlegenden Datenstrukturen Array, LinkedList, Queue oder Stack.
  - Visualisierung eines Sortieralgorithmus wie z.B Bubblesort oder Quicksort.
  - Visualisierung von rekursiven Methodenaufrufen.
  - Visualisierung der beiden Datenstrukturen Graph und Tree sowie einiger Algorithmen wie z.B Dijkstra Shortest Path oder Euler Traversierung.

- Auslieferung eines Softwareartefakts, welches einfach zu installieren und bedienen ist.

- Erstellen einer umfassenden Architekturdokumentation für Folgearbeiten.

- Erstellen einer Anleitung zur Entwicklung zusätzlicher Module.

Hinweise

- Bei dieser Bachelorarbeit handelt es sich um eine Folgearbeit der Studienarbeit "Graphs-Visualization-Service GVS 2.0".
- Das resultierende Produkt soll für den Unterricht in den HSR Modulen "Algorithmen und Datenstrukturen 1 und 2" verwendet werden.
- Es sind Folgearbeiten zu erwarten, welche das Framework um zusätzliche Module erweitern.

Generelles

- Die Vorgaben der Abteilung Informatik betreffend der Dokumentation [1] sind einzuhalten.
- Die "Generelle Richtlinien für Studien- und Bachelorarbeiten" [2] 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, 19.02.18 Beginn der Bachelorarbeit
- Freitag, 15.06.18 12:00 Uhr Abgabe der Bachelorarbeit

Betreuung

- Betreuer
  Thomas Letsch
  tletsch@hsr.ch
  055 - 22 24 567 (HSR Büro 5.204); 055 - 214 43 50 (Geschäft)
- Besprechungen
  Wöchentliche Besprechung jeweils Donnerstag 15:15 Uhr

Referenzen

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

Rapperswil, 19.Februar 2018

Thomas Letsch

Anhang B

Artefakt Übersicht

B.1 Overview

Meetingsprotokolle	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x
Persönliche Berichte																						v1.0
Eigenständigkeitserklärung																						v1.0
Vereinbarung über Urheber- und Nutzungsrechte																						v1.0
Zeitauswertung																						v1.0
Metriken																						v1.0
Plakat																						v1.0
Abstract																					v0.1	v1.0
Management Summary																						v1.0
Technischer Bericht																						v1.0
Anforderungsspezifikation				v0.1			v0.2															v1.0
Architektur- und Designspezifikation							v0.1															v1.0
Projektplan							v0.1															v1.0
Benutzerhandbuch																						v1.0
Manual Modulentwicklung									v0.1													v1.0
Glossar und Abkürzungsverzeichnis							v0.1															v1.0
Moduldokumentation Modul 1 *										v0.1												v1.0
Moduldokumentation Modul 2 *																v0.1						v1.0
ADV Release							v0.1							v0.2						v0.3	v1.0	
Systemtests Protokolle														v0.1						v0.2	v1.0	
Usabilitytests Protokolle																v0.1						v1.0
Artefakt Übersicht				v0.1	v0.2																	v1.0
Phase	**Incep tion**	**Elaboration**								**Construction**												**Transi tion**
Datum	19.02	26.02	5.03	12.03	19.03	26.03	2.04	9.04	16.04	23.04	30.04	7.05	14.05	21.05	28.05	4.06	11.06					
Woche/Iteration	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17					
Milestone		1				2	3					4				5		6				

** Enthält Anforderungen und Architektur*

Milestones

1 Anforderungsspezifikation
2 Architektur- und Designspezifikation
3 Prototype
4 Framework und 1. Modul
5 Framework und 2 Module
6 End of project

B.2 Inhalte

Meetingsprotokolle	Persönliche Berichte	Eigenständigkeitserklärung	Vereinbarung über Urheber- und Nutzungsrechte	Zeitauswertung
Protokolle zu allen wöchentlichen Sitzungen vom Projektteam und Betreuer. Struktur: - Rückblick - Aktuelles - Beschlüsse - Ausblick	Persönliche Berichte einschliesslich (selbst-)kritische Reflexion der Projektmitglieder zu ihren Erfahrungen bei der Arbeit.	Unterschriebene Erklärung zur Eigenständigkeit der Arbeit	Unterschriebene Vereinbarung bezüglich Urheber- und Nutzungsrechten	Auswertung der investierten Zeit aller Projektmitglieder. Enthaltene Diagramme: - Zeitauswertung nach Woche und Teammitglied - Auswertung Soll-/Ist-Zeit - Auswertung nach Kategorien - Auswertung nach Milestone
Versionen: *1.0 Endgültige Version*	*Versionen:* *1.0 Endgültige Version*	*Versionen:* *1.0 Endgültige Version*	*Versionen:* *1.0 Endgültige Version*	*Versionen:* *1.0 Endgültige Version*
Metriken	**Plakat**	**Abstract**	**Management Summary**	**Technischer Bericht**
Grafiken und Zahlen zur Projektqualität - LOC - Codacy Grade - Issues - Complex Classes - Duplicated Code - Test Coverage	Zusammenfassung der Arbeit auf einem Poster - Ausgangslage - Vorgehen / Technologien - Ergebnisse	Der Abstract richtet sich an den Spezialisten auf dem entsprechenden Gebiet und beschreibt daher in erster Linie die (neuen, eigenen) Ergebnisse und Resultate der Arbeit.	Das Management Summary richtet sich an die Vorgesetzten des Auftraggebers. - Ausgangslage - Vorgehen / Technologien - Ergebnisse	Zusammenfassung der Arbeit auf ca 10 Seiten - Ausgangslage und Problembeschreibung - Lösungskonzept - Umsetzung - Ergebnisse - Schlussfolgerungen
Versionen: *1.0 Endgültige Version*	*Versionen:* *1.0 Endgültige Version*	*Versionen:* *0.1 Vollständige Version zum Review* *1.0 Endgültige Version mit Korrekturen*	*Versionen:* *1.0 Endgültige Version*	*Versionen:* *1.0 Endgültige Version*
Anforderungsspezifikation	**Architektur- und Designspezifikation**	**Projektplan**	**Benutzerhandbuch**	**Manual Modulentwicklung**
Vollständige, einheitlich dokumentierte und konsistente Zusammenstellung aller Anforderungen an den ADV	Detaillierte Dokumention der geplanten Architektur inkl Architekturentscheide und das effektive Design	Projektmanagement und -organisation mit Risikoliste	Installations- und Benutzeranleitung für Enduser	Installations- und Benutzeranleitung für Modulentwickler
Versionen: *0.1* *- Terminologie - Stakeholder* *- User Stories, Use Cases* *- Referenzarchitekturen* *- Technische Anforderungen, NFR* *- Modul-Konzepte - User Analyse* *0.2* *- Allg. Korrekturen und Ergänzungen* *1.0* *- Endgültige Version*	*Versionen:* *0.1* *- C4 Modelle* *- IPC* *- Qualitätsattribute* *- Threading* *- Events* *1.0* *- Benutzeroberfläche* *- Usability Test Resultate* *- Systemtest Resultate*	*Versionen:* *0.1* *- Projektverwaltung* *- Risikomanagement* *- Verantwortlichkeiten* *- Backup System* *- Milestones* *- Entwicklungskonzept* *1.0* *Endgültige Version*	*Versionen:* *1.0* *- Installationsanleitung* *- Benutzeranleitun* *- Allgemeines zum Core* *- Module Array* *- Module Stack* *- Module Queue* *- Module Graph*	*Versionen:* *0.1* *- Projektstruktur* *- Entwicklungsumgebung/Installation* *- Modulentwicklung* *- API (Protokoll, Widgets, Styles, ...)*

Glossar und Abkürzungsverzeichnis	Moduldokumentation Modul 1	Moduldokumentation Modul 2	ADV Release	Systemtests Protokolle
Auflistung und Erklärung aller verwendeter Abkürzungen und Begrifflichkeiten	Anforderungs-, Architektur- und Designspezifikation für Modul 1	Anforderungs-, Architektur- und Designspezifikation für Modul 2	Lauffähige bzw. verwendbare JAR Files für ADV Lib und ADV UI	Testszenarien mit Score, Kommentare
Versionen: *0.1* *- Ubiquitous Language* *- Begriffe aus Anforderungs- und Designspezifikation* *1.0* *- Begriffe aus Modules* *- Begriffe aus Manuals*	*Versionen:* *0.1* *- User Stories* *- Architektur* *1.0* *- Ergänzungen und Korrekturen*	*Versionen:* *0.1* *- User Stories* *- Architektur* *1.0* *- Ergänzungen und Korrekturen*	*Versionen:* *0.1* *- Lib und UI Core* *- Verbindungsaufbau und Kommunikation* *- Array Module* *- Prototyp* *0.2* *- abgeschlossenes Array Module* *0.3* *- abgeschlossenes BFS/DFS Module* *1.0* *- endgültige Version*	*Versionen:* *0.1* *- Testszenarien und Scores* *Durchführung 1* *0.2* *- Testszenarien und Scores* *Durchführung 2* *1.0* *- endgültige Version*
Usabilitytests Protokolle	**Artefakt Übersicht**			
Testszenarien mit Score, Kommentare Testperson und Tester	Zeitplanung nach Wochen und erstellten Artefakten			
Versionen: *0.1* *- Testszenarien und Scores* *Durchführung 1 und 2* *1.0* *- endgültige Version*	*Versionen:* *0.1* *- Auflistung aller Artefakte* *0.2* *- Zeitliche Planung der Artefakte* *1.0* *- endgültige Version*			

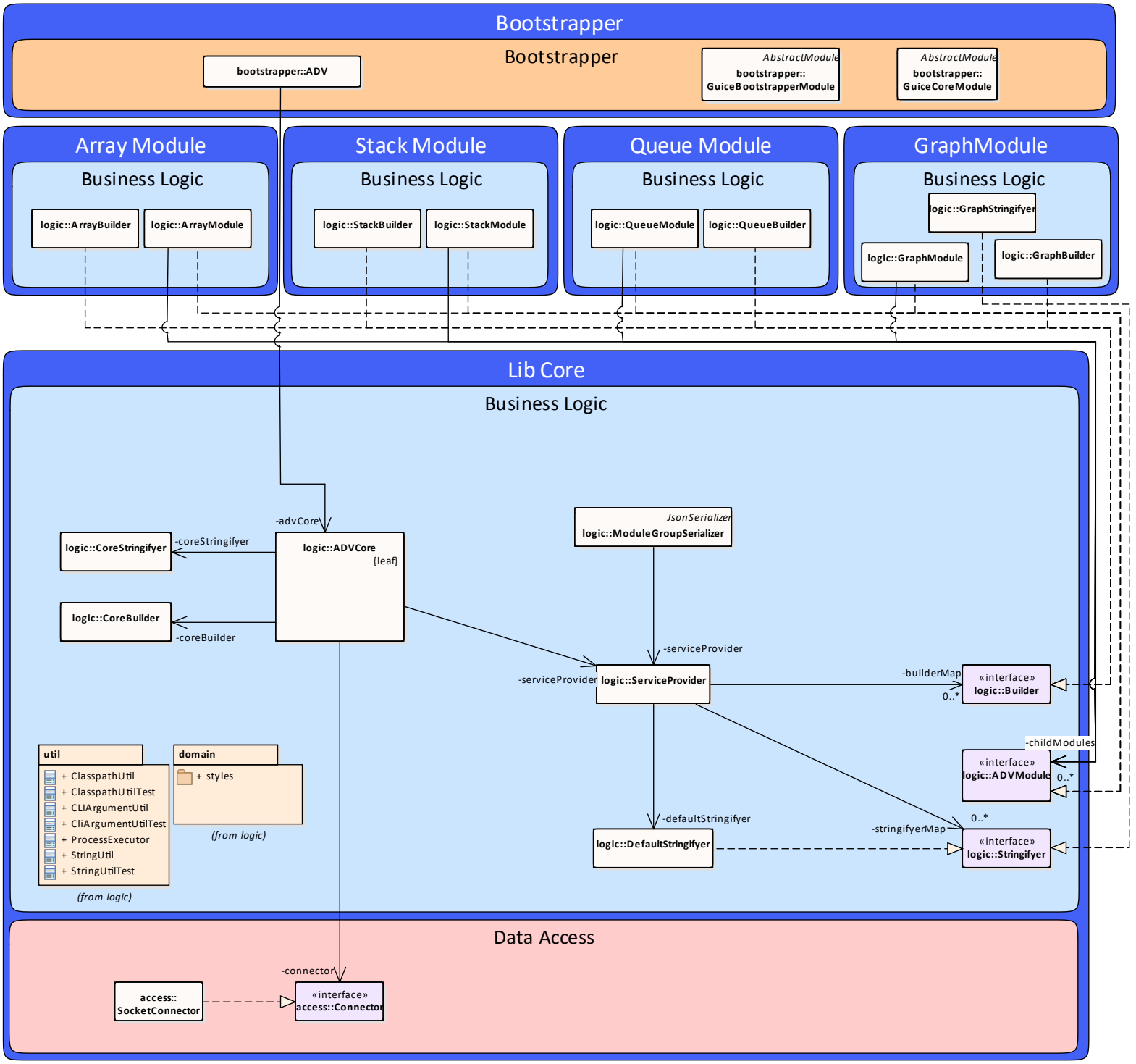
Anhang C

Klassendiagramme

Klassendiagramme mit den wichtigsten Klassen sowie einer Einteilung in Layers

1. Klassendiagramm ADV Lib
2. Klassendiagramm ADV UI
3. Klassendiagramm ADV Commons

Klassendiagramm ADV Lib



Klassendiagramm ADV UI

Bootstrapper

Bootstrapper

bootstrapper::Bootstrapper

AbstractModule
bootstrapper::
GuiceBootstrapperModule

AbstractModule
bootstrapper::
GuiceCoreModule

Array Module

Presentation

-arrayDefaultLayouter
presentation::ArrayDefaultLayouter

-arrayObjectReferenceLayouter
presentation::
ArrayObjectReferenceLayouter

presentation::
ArrayLayouter

Business Logic

logic::
ArrayParser

Stack Module

Presentation

presentation::
StackLayouter

Business Logic

logic::
StackParser

Queue Module

Presentation

presentation::
QueueLayouter

Business Logic

logic::
QueueParser

Graph Module

Presentation

presentation::
GraphLayouter

Business Logic

logic::
GraphParser

logic::
GraphStringifyer

UI Core

Presentation

Application
presentation::ADVApplication

widgets

(from presentation)

presentation::SessionView

presentation::RootView

-replayViewModel
presentation::ReplayViewModel

-stateViewModel
-stateViewModel
-steppingViewModel
-steppingViewModel
-stateViewModel
-rootViewModel

presentation::SteppingViewModel

presentation::StateViewModel

presentation::RootViewModel

Service

-socketServer
Thread
service::SocketServer

service::ADVConnection

Business Logic

-layoutedSnapshotStore
-layoutedSnapshotStore
-layoutedSnapshotStore
-layoutedSnapshotStore

stores::LayoutedSnapshotStore

-eventManager
-eventManager
-eventManager
-eventManager

events::EventManager

logic::CoreParser
-coreParser

logic::CoreLayouter
-coreLayouter

logic::CoreStringifyer
-coreStringifyer

-flowControl

logic::FlowControl

-sessionStore
-sessionStore

stores::
SessionStore

-flowControl

-stringifyerMap
0..*

«interface»
logic::Stringifyer

-serviceProvider

logic::ServiceProvider

-serviceProvider
-serviceProvider

-layouterMap
0..*

«interface»
logic::Layouter

JsonDeserializer
logic::ModuleGroupDeserializer

JsonSerializer
logic::ModuleGroupSerializer

-parserMap
0..*

«interface»
logic::Parser

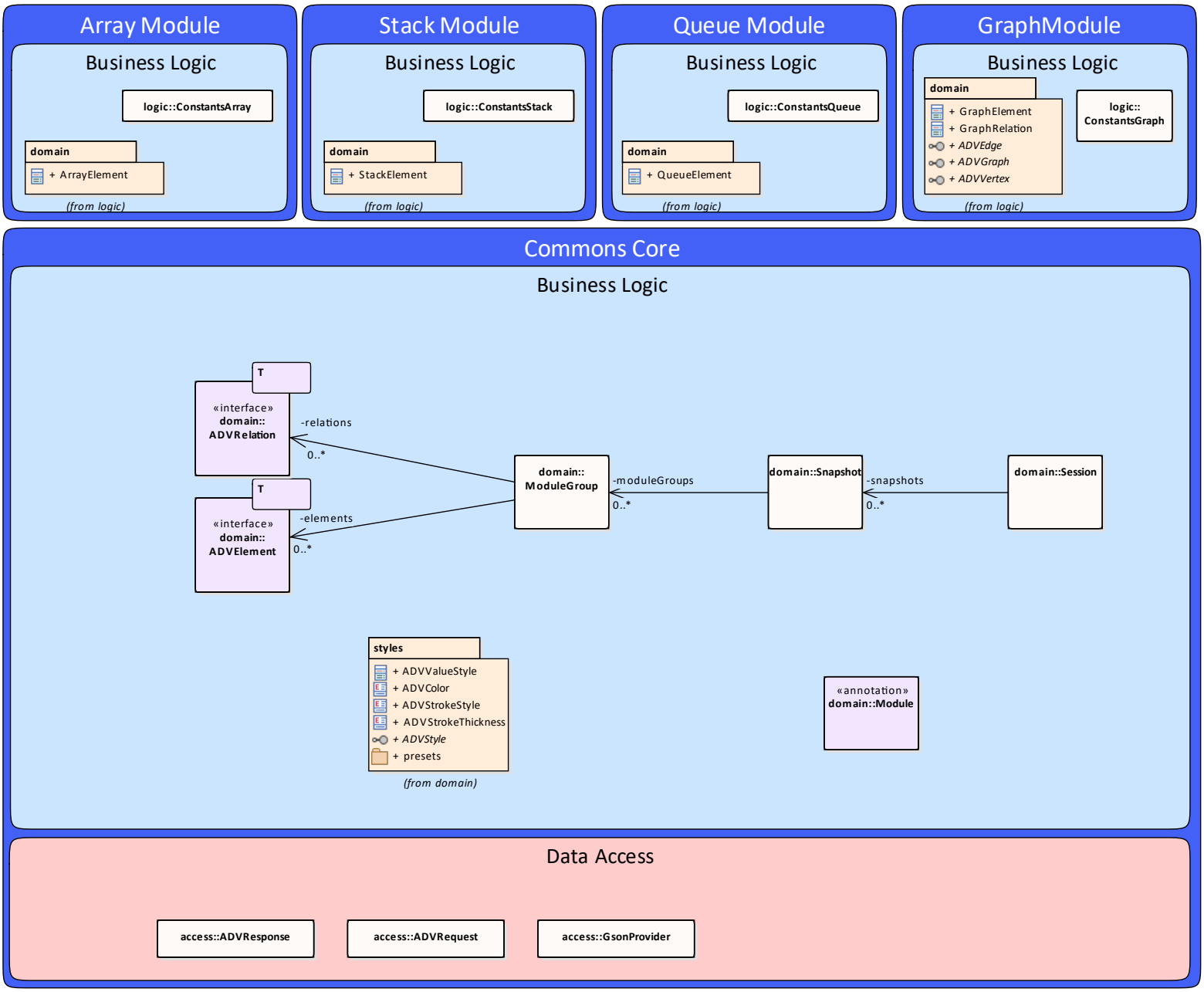
Data Access

-fileAccess
-fileAccess

«interface»
access::DatastoreAccess

access::FileDatastoreAccess

Klassendiagramm ADV Commons



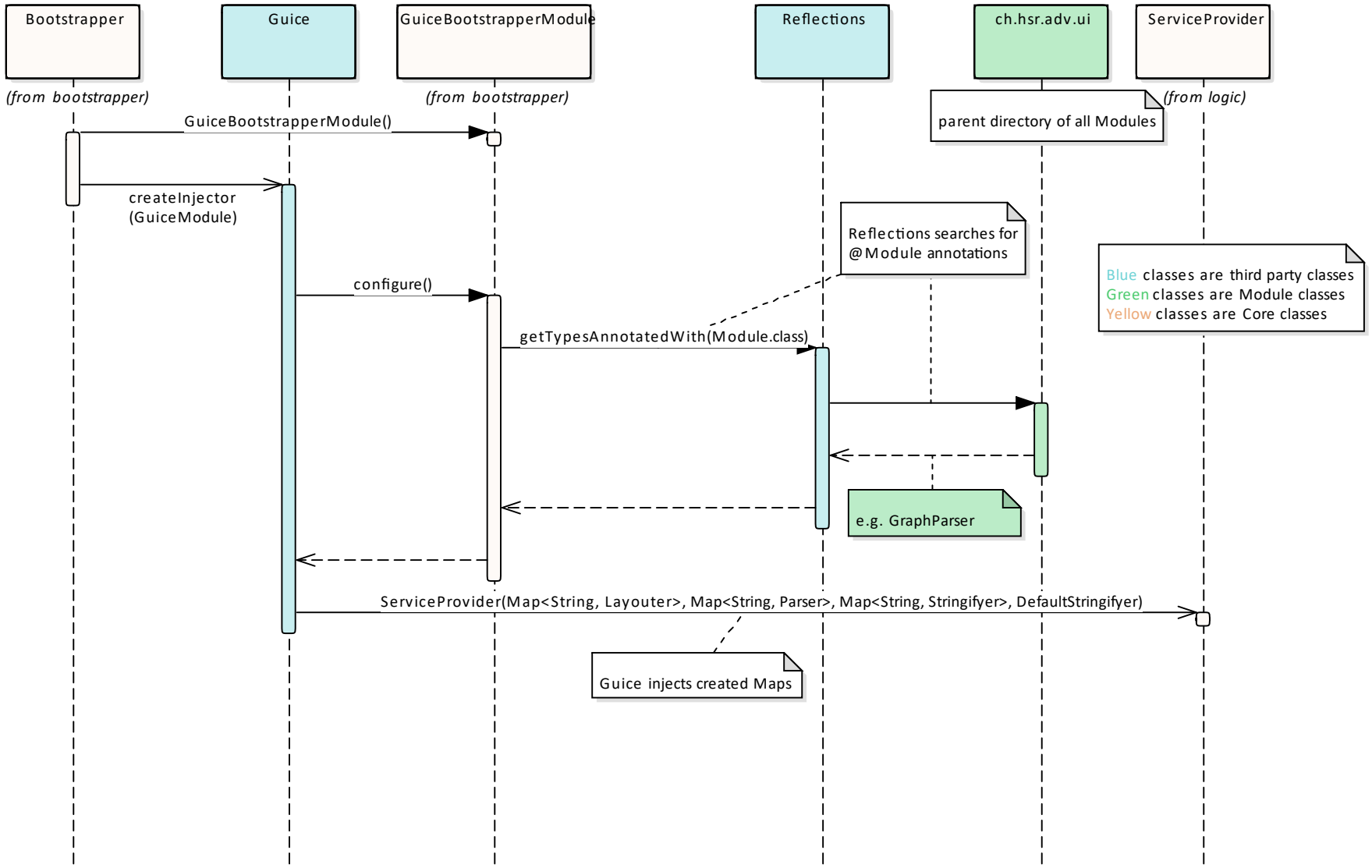
Anhang D

Sequenzdiagramme

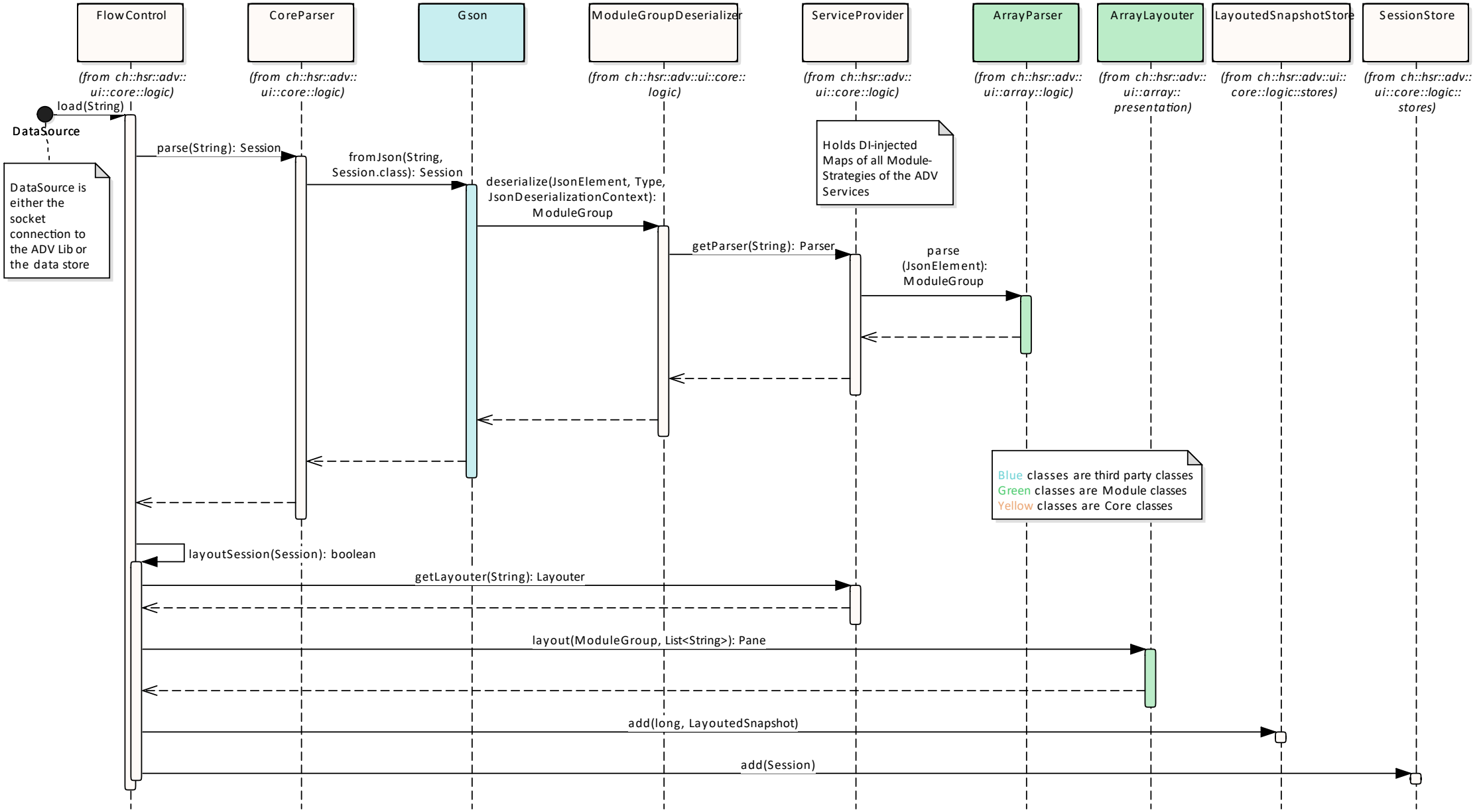
Sequenzdiagramme der wichtigsten Abläufe im ADV

1. Sequenzdiagramm Bootstrapping
2. Sequenzdiagramm FlowControl
3. Sequenzdiagramm EventManager

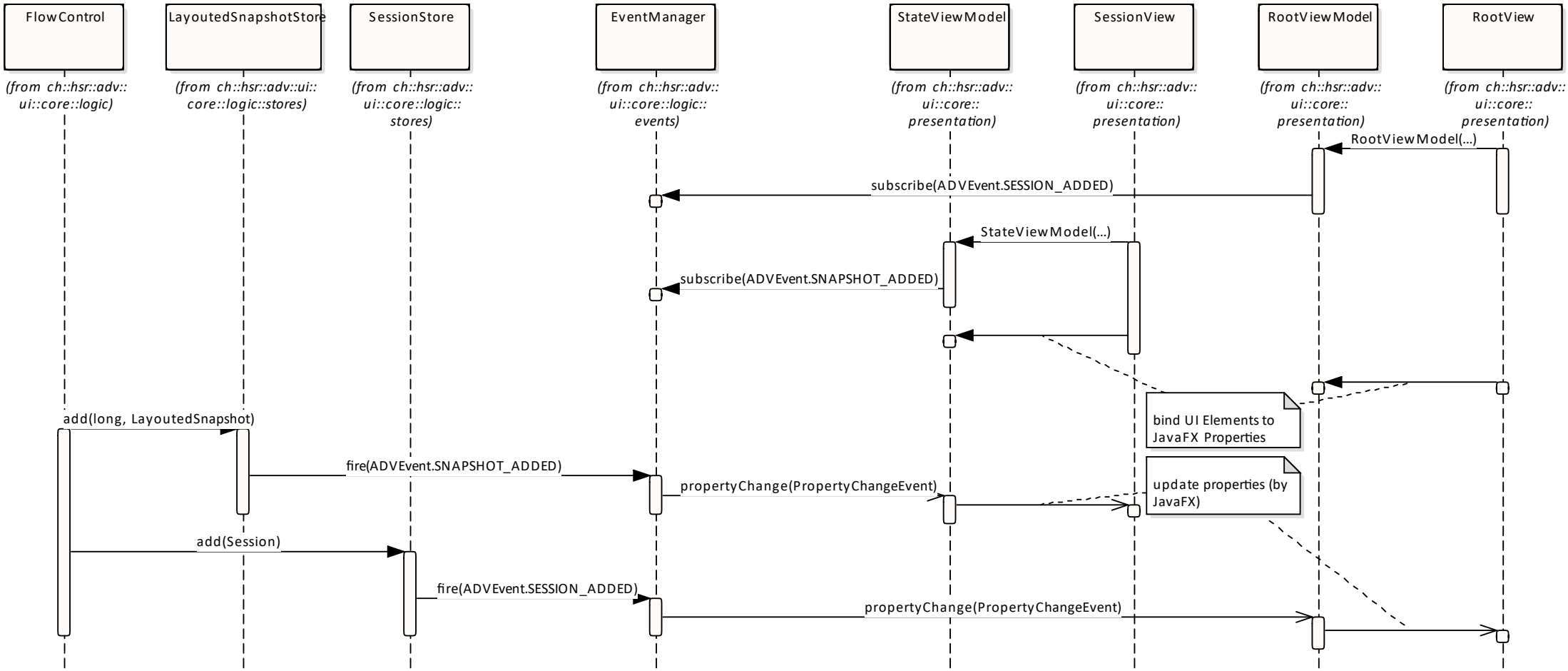
Sequenzdiagramm Bootstrapping



Sequenzdiagramm FlowControl



Sequenzdiagramm EventManager



Anhang E

User Analyse

Mit Nutzern des GVS wurde eine Nutzeranalyse durchgeführt.

E.1 Ergebnisse

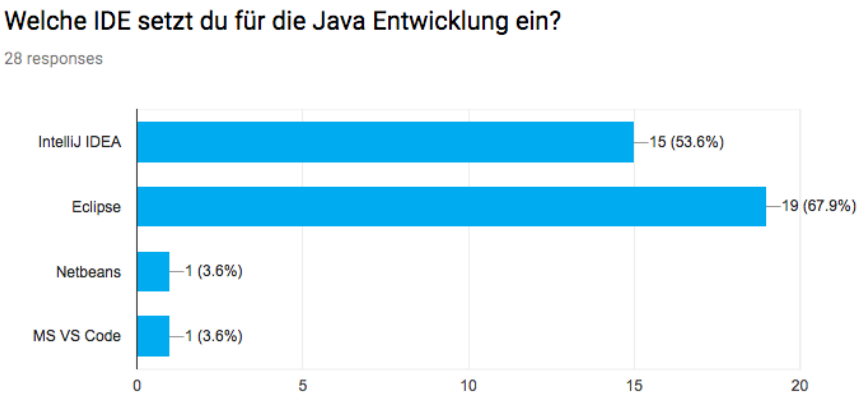


Abbildung E.1: User Analyse: Frage 1

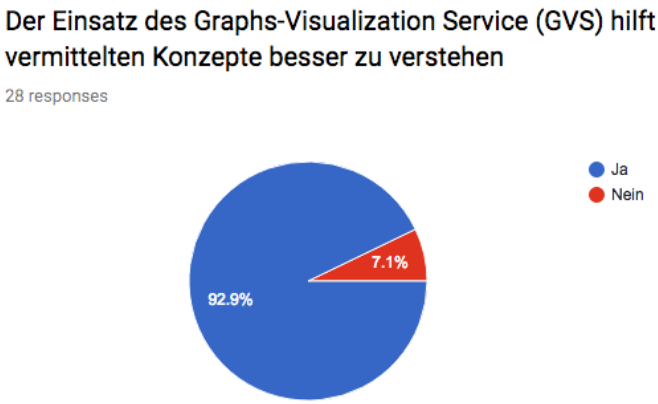


Abbildung E.2: User Analyse: Frage 2

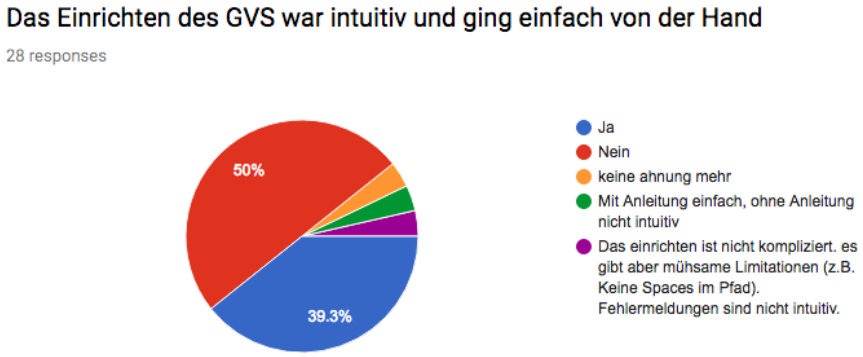


Abbildung E.3: User Analyse: Frage 3

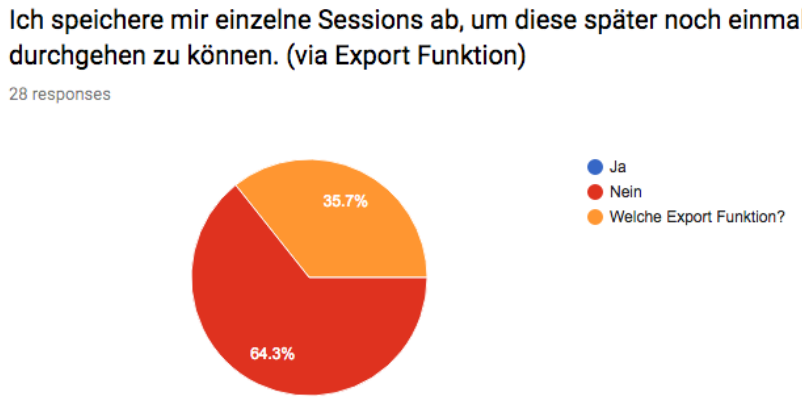


Abbildung E.4: User Analyse: Frage 4

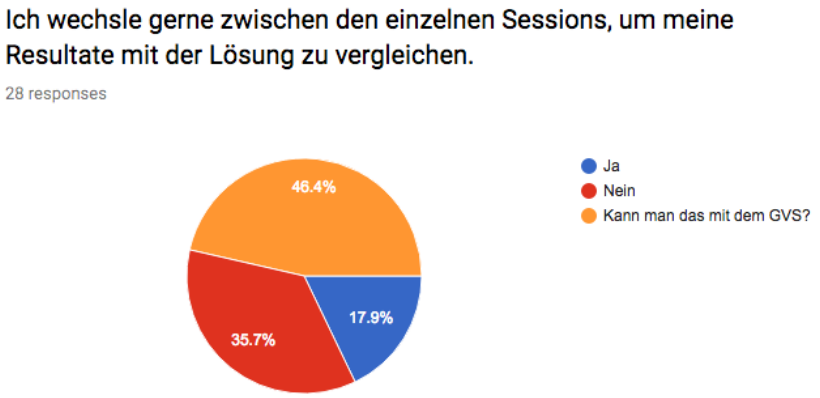


Abbildung E.5: User Analyse: Frage 5

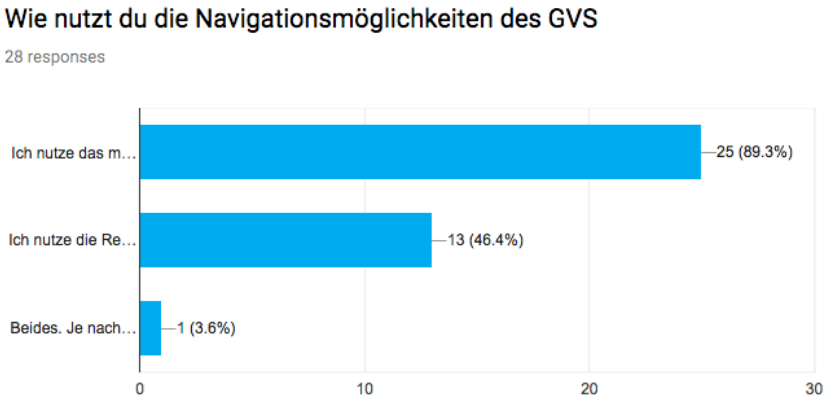


Abbildung E.6: User Analyse: Frage 6

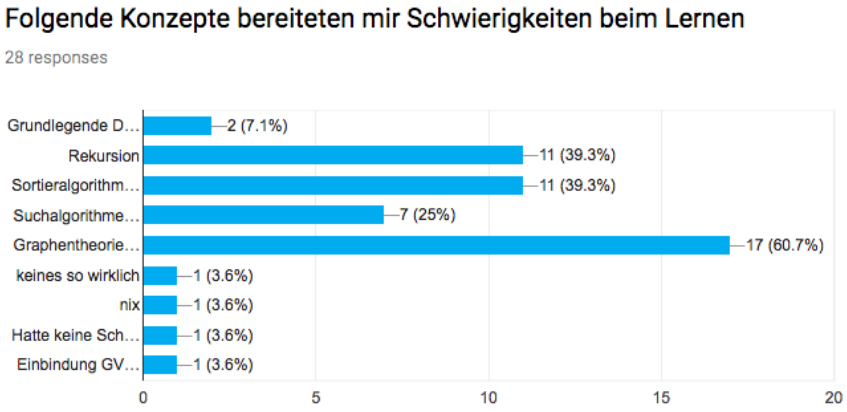


Abbildung E.7: User Analyse: Frage 7

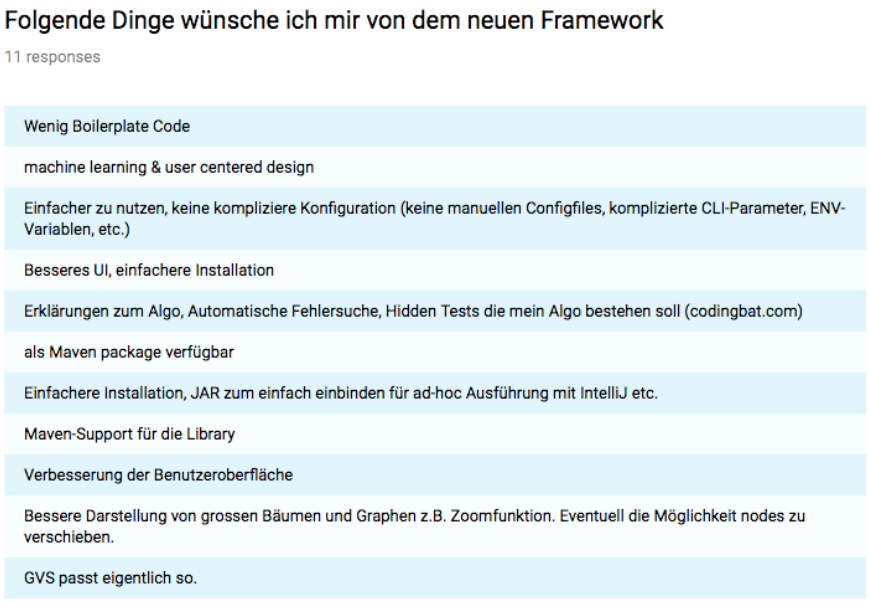


Abbildung E.8: User Analyse: Frage 8

Anhang F

Testprotokolle

F.1 Systemtests

Zur Überprüfung ganzer Abläufe, welche die Library und das UI umfassen, wurden Systemtests durchgeführt. Die definierten Szenarien wurden zwei Mal während der Construction Phase getestet.

ADV Systemtests

System under Test	https://github.com/ADVisualizer	+	Bestanden
Datum	23.04.18	-	Nicht Bestanden

Testergebnis ✔92%

ADV UI

#	Test File	Test Beschreibung	Erwartetes Resultat	Ergebnis	Bemerkungen
1	adv-ui.jar	Starten des UI's	Das UI kann mit dem JAR File gestartet werden.	+	
2	adv-ui.jar	Konfigurieren des Socket Port	Der Socket-Server wird auf dem Konfigurierten Port gestartet (Host = Default)	+	
3	adv-ui.jar	Konfigurieren des Socket Hosts	Der Socket-Server wird auf dem Konfigurierten Host gestartet (Port = Default)	+	
4	Bubblesort.adv	Laden einer Session über das UI	Im Lade-Pop-up sind nur *.adv Dateien auswählbar.	+	
			Die geladene Session wird in der Session Liste markiert und erscheint zuunterst in der Liste	+	
			Ein Tab mit dem Session Inhalt öffnet sich	+	
			Statusbar zeigt Erfolg an	+	
5	Bubblesort.adv	Speichern einer Session über das UI	Die Session wird im json Format gespeichert.	+	
			Als Dateiname wird der Name der Session vorgeschlagen	+	
			Die Dateiendung *.adv wird automatisch an den Pfad angehängt	+	
			Statusbar zeigt Erfolg an	+	
6	Bubblesort.adv	Steppen durch eine Session	Die Step-Funktionalitäten funktionieren.	+	
			Die korrekten Buttons sind aktiviert/deaktiviert.	+	
			Die Progressbar zeigt den jeweiligen Stand.	+	
7	Bubblesort.adv	Replay einer Session	Das Replay beginnt beim aktiven Snapshot. Der ausgewählte Speed wird verwendet.	+	
			Der Default Speed ist 'medium'	+	
			Während dem Replay sind alle Buttons deaktiviert mit ausnahme von 'Cancel' und 'Pause'	+	
			'Pause' hält beim momentanen Snapshot an und aktiviert alle nötigen Buttons. Bei erneutem 'Replay' wird allenfalls mit neuem Speed weitergefahren.	+	
			'Cancel' setzt die Session auf den ersten Snapshot zurück und aktiviert alle nötigen Buttons.	+	
8	Bubblesort.adv, ObjectArray.adv	Wechseln zwischen zwei aktiven Sessions	Zwischen aktiven Sessions kann jederzeit (auch während einem Replay) geswitcht werden.	+	
9	Bubblesort.adv	Löschen einer Session	Die Session wird aus der Session-List entfernt.	+	
			Das entsprechende Tab wird geschlossen.	+	
			Falls weitere Sessions offen sind, wird eine andere Session als 'ausgewählt' markiert.	+	
			Statusbar zeigt Erfolg an	+	
10	Bubblesort.adv, ObjectArray.adv	Löschen alles Sessions	Alle aktiven Sessions werden aus der Session-List entfernt.	+	
			Alle Tabs werden geschlossen.	+	
			Das ADV Logo wird stattdessen angezeigt.	+	
			Statusbar zeigt Erfolg an	+	
11	Bubblesort.adv	Wechseln der Sprache	Beim Starten des UI's ist die Sprache 'EN' ausgewählt.	+	
			Alle Labels und Tooltips sind auf englisch.	+	
			Beim Wechsel auf 'DE' ändern sich alle Labels und Tooltips.	-	Die Labels des Speed Slider ändern erst, für die neuen

12	Bubblesort.adv, Bubblesort.adv	Laden einer bereits geladenen Session (Duplicated Session)	Die Session wird nicht erneut geladen.	+	
			Die Statusbar zeigt einen entsprechenden Hinweis an	+	
13	Wrong.adv	Session mit falschem oder veraltetem Format wird geladen	Statusbar zeigt Hinweis, dass Laden nicht funktioniert hat.	-	Keine Meldung ist sichtbar
			Es wird keine zusätzliche Session in der Liste oder als Tab angezeigt.	+	
14	ObjectArray.adv	AutoScalePane Widget	Alle Kindelemente der Autoscale Pane werden automatisch nach dem verfügbaren Platz skalliert und zentriert	+	
15	ObjectArray.adv	LabeledNode Widget	Das Label wird zentriert dargestellt	+	
16	ObjectArray.adv	LabeledEdge Widget	Das Label wird in der Mitte der Beziehung angezeigt		Kann zum aktuellen Zeitpunkt nicht beurteilt werden
			Pfeile werden optional unibidirektional oder bidirektional dargestellt		Kann zum aktuellen Zeitpunkt nicht beurteilt werden
			Pfeile werden entweder an den vier Fixpunkten (Oben, Unten, Links, Rechts) oder am berechneten direktesten Punkt gebunden		Kann zum aktuellen Zeitpunkt nicht beurteilt werden

ADV Lib

#	Test File	Test Beschreibung	Erwartetes Resultat	Ergebnis	Bemerkungen
17	adv-lib.jar	Die eingebundene Lib wird genutzt. Das UI läuft noch nicht.	Die Lib startet das UI und schickt anschliessend die Snapshots an das UI.	+	
18	adv-lib.jar	Die eingebundene Lib wird genutzt. Das UI läuft bereits.	Das UI wird nicht erneut gestartet.	+	
			Die Lib verbindet an das UI und sendet die Snapshots.	+	
19	adv-lib.jar	Konfigurieren des Socket Port	Es wird eine Verbindung auf dem Konfigurierten Port aufgebaut (falls das ADV UI noch nicht läuft wird der Socket-Server auf dem konfigurierten Port gestartet)	+	
20	adv-lib.jar	Konfigurieren des Socket Hosts	Es wird eine Verbindung zu dem Konfigurierten Host aufgebaut (falls das ADV UI noch nicht läuft, wird der Socket-Server auf dem konfigurierten Host gestartet)	-	Wenn das UI von der Lib gestartet wird, wird der Host nicht gesetzt (rsp. Das UI wird mit host Argument gar nie
21	StylesShowcase.java	Die vorgefertigten Styles werden verwendet.	ADVDefaultStyle, ADVErrorStyle, ADVInfoStyle,ADVSuccessStyle,ADVWarningStyle werden wie gewünscht dargestellt	+	
22	StylesShowcase.java	ADVEnumStyle wird verwendet	Die EnumStyle Klasse kann mit Enumwerten für Color, StrokeStyle und StrokeThickness gefüllt werden und wird entsprechend dargestellt.	-	FillColor wird nicht übernommen
23	StylesShowcase.java	ADVValueStyle wird verwendet	Die ValueStyle Klasse kann mit beliebigen Hex-Values für Farben und beliebigen int Werten für die Randbreite gefüllt werden und wird entsprechend dargestellt.	+	
24	CoordsShowcase.java	Koordinaten werden definiert	Die gesetzten Koordinaten werden verwendet. Bei gemischten Elementen (solche mit und ohne Koordinaten) werden die Elemente ohne Koordinaten vom Modul-Layouter platziert.	+	
25	NullPointer.java	Zwischen zwei Snapshots kommt es in der User Codebase zu	Das UI zeigt die bis zum Fehler übertragenen Snapshots in einer Session an.	+	

25	NullPointer.java	der User Codebase zu einer NullPointerException	Die Socket ist für die nächste Verbindung bereit	+	
26	Bubblesort.java, ObjectArray.java	Zwei Sessions senden gleichzeitig Daten an das ADV UI	Die Sessions werden serialisiert empfangen und dargestellt.		

Array Module

#	Test File	Test Beschreibung	Erwartetes Resultat	Ergebnis	Bemerkungen
27	Bubblesort.java	Ausführen eines Javaprogramms zur Darstellung eines Array-Algorithmus' mit Hilfe von Styles	Styles zeigen aktive Abläufe	+	
28	ObjectArray.java	Ausführen eines Java Programms zur Darstellung von Objektreferenzen im Array Module	Objektreferenzen sind sichtbar	+	

ADV Systemtests

System under Test	https://github.com/ADVisualizer	+	Bestanden
Datum	29.05.18	-	Nicht Bestanden

Testergebnis ✔ 96%

ADV UI

#	Test File	Test Beschreibung	Erwartetes Resultat	Ergebnis	Bemerkungen
1	adv-ui.jar	Starten des UI's	Das UI kann mit dem JAR File gestartet werden.	+	
2	adv-ui.jar	Konfigurieren des Socket Port	Der Socket-Server wird auf dem Konfigurierten Port gestartet (Host = Default)	+	
3	adv-ui.jar	Konfigurieren des Socket Hosts	Der Socket-Server wird auf dem Konfigurierten Host gestartet (Port = Default)	+	
4	Bubblesort.adv	Laden einer Session über das UI	Im Lade-Pop-up sind nur *.adv Dateien auswählbar.	+	
			Die geladene Session wird in der Session Liste markiert und erscheint zuunterst in der Liste	+	
			Ein Tab mit dem Session Inhalt öffnet sich	+	
			Statusbar zeigt Erfolg an	+	
5	Bubblesort.adv	Speichern einer Session über das UI	Die Session wird im json Format gespeichert.	+	
			Als Dateiname wird der Name der Session vorgeschlagen. Leerzeichen werden mit Bindestrichen eretzt	+	
			Die Dateiendung *.adv wird automatisch an den Pfad angehängt	+	
			Statusbar zeigt Erfolg an	+	
6	Bubblesort.adv	Steppen durch eine Session	Die Step-Funktionalitäten funktionieren.	+	
			Die korrekten Buttons sind aktiviert/deaktiviert.	+	
			Die Progressbar zeigt den jeweiligen Stand.	+	
7	Bubblesort.adv	Replay einer Session	Das Replay beginnt beim aktiven Snapshot. Der ausgewählte Speed wird verwendet.	+	
			Der Default Speed ist 'medium'	+	
			Während dem Replay sind alle Buttons deaktiviert mit ausnahme von 'Cancel' und 'Pause'	+	Es sind nur die Buttons in der SessionView deaktiviert. Die Buttons in der Root View sind immernoch klickbar. Dies hat aber keinen negativen Effekt
			'Pause' hält beim momentanen Snapshot an und aktiviert alle nötigen Buttons. Bei erneutem 'Replay' wird allenfalls mit neuem Speed weitergefahren.	+	
			'Cancel' setzt die Session auf den ersten Snapshot zurück und aktiviert alle nötigen Buttons.	+	
8	Bubblesort.adv, ObjectArray.adv	Wechseln zwischen zwei aktiven Sessions	Zwischen aktiven Sessions kann jederzeit (auch während einem Replay) geswitcht werden.	+	
9	Bubblesort.adv	Löschen einer Session	Die Session wird aus der Session-List entfernt.	+	
			Das entsprechende Tab wird geschlossen.	+	
			Falls weitere Sessions offen sind, wird eine andere Session als 'ausgewählt' markiert.	+	
			Statusbar zeigt Erfolg an	+	
10	Bubblesort.adv, ObjectArray.adv	Löschen alles Sessions	Alle aktiven Sessions werden aus der Session-List entfernt.	+	
			Alle Tabs werden geschlossen.	+	

	ObjectArray.adv		Das ADV Logo wird stattdessen angezeigt.	+	
			Statusbar zeigt Erfolg an	+	
11	Bubblesort.adv	Wechseln der Sprache	Beim Starten des UI's ist die Sprache 'EN' ausgewählt.	+	
			Alle Labels und Tooltips sind auf englisch.	+	
			Beim Wechsel auf 'DE' ändern sich alle Labels und Tooltips.	-	Die Label der Speed Slider ändern sich nicht. Ebenfalls wird das Deutsche Label "mittel" nicht angezeigt
12	Bubblesort.adv, Bubblesort.adv	Laden einer bereits geladenen Session (Duplicated Session)	Die Session wird nicht erneut geladen.	+	
			Die Statusbar zeigt einen entsprechenden Hinweis an	+	
13	Wrong.adv	Session mit falschem oder veraltetem Format wird geladen	Statusbar zeigt Hinweis, dass Laden nicht funktioniert hat.	+	
			Es wird keine zusätzliche Session in der Liste oder als Tab angezeigt.	+	
14	ObjectArray.adv	AutoScalePane Widget	Alle Kindelemente der Autoscale Pane werden automatisch nach dem verfügbaren Platz skalliert	+	
			Die skalierten Elemente werden zentriert dargestellt	+	
15	ObjectArray.adv	LabeledNode Widget	Das Label wird zentriert dargestellt	+	
16	ObjectArray.adv	LabeledEdge Widget	Das Label wird in der Mitte der Edge angezeigt	+	
			Pfeile werden optional unibidirektional oder bidirektional dargestellt	+	
			Pfeile werden entweder an den vier Fixpunkten (Oben, Unten, Links, Rechts) oder am berechneten direktesten Punkt gebunden	+	
17	Graph.adv	CruvedLabeledEdge Widget	Pfeile haben eine Krümmung	+	
			Das Label wird in der Mitte der Edge angezeigt	+	

ADV Lib					
#	Test File	Test Beschreibung	Erwartetes Resultat	Ergebnis	Bemerkungen
18	adv-lib.jar	Die eingebundene Lib wird genutzt. Das UI läuft noch nicht.	Die Lib startet das UI und schickt anschliessend die Snapshots an das UI.	-	UI wird gestartet, es kann aber keine Verbindung zum Socket hergestellt werden (beim ersten Mal)
19	adv-lib.jar	Die eingebundene Lib wird genutzt. Das UI läuft bereits.	Das UI wird nicht erneut gestartet.	+	
			Die Lib verbindet an das UI und sendet die Snapshots.	+	
20	adv-lib.jar	Konfigurieren des Socket Port	Es wird eine Verbindung auf dem Konfigurierten Port aufgebaut (falls das ADV UI noch nicht läuft wird der Socket-Server auf dem konfigurierten Port gestartet)	+	
21	adv-lib.jar	Konfigurieren des Socket Hosts	Es wird eine Verbindung zu dem Konfigurierten Host aufgebaut (falls das ADV UI noch nicht läuft, wird der Socket-Server auf dem konfigurierten Host gestartet)	+	
22	StylesShowcase.java	Die vorgefertigten Styles werden verwendet.	ADVDefaultStyle, ADVErrorStyle, ADVInfoStyle,ADVSuccessStyle,ADVWarningStyle werden wie gewünscht dargestellt	+	
23	StylesShowcase.java	ADVEnumStyle wird verwendet	Die EnumStyle Klasse kann mit Enumwerten für Color, StrokeStyle und StrokeThickness gefüllt werden und wird entsprechend dargestellt.	+	

24	StylesShowcase.java	ADVValueStyle wird verwendet	Die ValueStyle Klasse kann mit beliebigen Hex-Values für Farben und beliebigen int Werten für die Randbreite gefüllt werden und wird entsprechend dargestellt.	+	
25	Graph.java	Koordinaten werden definiert	Die gesetzten Koordinaten werden verwendet.	+	
26	NullPointer.java	Zwischen zwei Snapshots kommt es in der User Codebase zu einer NullPointerException	Das UI zeigt die bis zum Fehler übertragenen Snapshots in einer Session an.	+	
			Die Socket ist für die nächste Verbindung bereit	+	
27	Bubblesort.java, ObjectArray.java	Zwei Sessions senden gleichzeitig Daten an das ADV UI	Die Sessions werden serialisiert empfangen und dargestellt.	+	getestet übers Netzwerk mit zwei Clients

Array Module					
#	Test File	Test Beschreibung	Erwartetes Resultat	Ergebnis	Bemerkungen
28	Bubblesort.java	Ausführen eines Javaprogramms zur Darstellung eines Array-Algorithmus' mit Hilfe von Styles	Styles zeigen aktive Abläufe	+	
29	ObjectArray.java	Ausführen eines Java Programms zur Darstellung von Objektreferenzen im Array Module	Objektreferenzen sind sichtbar	+	

DFS/BFS Module					
#	Test File	Test Beschreibung	Erwartetes Resultat	Ergebnis	Bemerkungen
30	DFS.java	Ausführen eines Javaprogramms zur Darstellung eines DFS mit Hilfe von Styles	*User Codebase:* Ans Graph Module kann das Stack Module als Child angehängt werden.	+	
			UI: ein Graph und ein Stack sind in einer Splitpane ersichtlich. Der Slider ist verstellbar. Die Position des Sliders wird beim Steppen persistiert	+	
31	BFS.java	Ausführen eines Javaprogramms zur Darstellung eines BFS mit Hilfe von Styles	*User Codebase* : Ans Graph Module kann das Queue Module als Child angehängt werden.	+	
			UI: ein Graph und eine Queue sind in einer Splitpane ersichtlich. Der Slider ist verstellbar. Die Position des Sliders wird beim Steppen persistiert	+	
32	Graph.java	Ausführen eines Javaprogramms zur Darstellung eines Graphen	Selbstreferenzen werden korrekt dargestellt.	+	
			Bei doppelten Verbindungen zwischen zwei Nodes wird eine der Verbindungen gerade und eine gekrümmt dargestellt.	+	
33	Stack.java	Ausführen eines Javaprogramms zur Darstellung eines Stacks	Leerer Stack wird korrekt dargestellt	+	
			Voller Stack wird korrekt dargestellt	+	
34	Queue.java	Ausführen eines Javaprogramms zur Darstellung einer Queue	Leere Queue wird korrekt dargestellt	+	
			Anfang (Head) und Ende (Tail) der Queue sind markiert	+	Das Tail ist zwar nicht markiert, es ist aber implizit durch die Head markierung klar
			Volle Queue wird korrekt dargestellt	+	

F.2 Usability Tests

Zwei Mal während der Construction Phase wurden Usability Tests durchgeführt. Nachfolgend finden sich die Protokolle der Durchführungen.

Usability Test: Szenarien

Infos für Testpersonen:
Handelt sich um Software zur Darstellung von Algorithmen & Datenstrukturen. Jede Durchführung eines Programms führt zu einer Session mit 1 bis mehreren Snapshots, die einzeln angesehen werden können.
Wir testen die Bedienbarkeit der Software und nicht die Fähigkeiten der Testperson!

Skala Auswertung	
Was soll ich tun??	1
War schwierig, aber hat geklappt	2
Leichtes Verbesserungspotential	3
Alles klar!	4

Anmerkungen Testperson:
Ideal wäre eine ein Feedback wenn die Session geladen, gespeichert und gelöscht wurde. (erfolgreich/nicht erfolgreich) -> Allenfalls über Message Bar

#	Szenario	Feedback Testperson	Bemerkungen Tester	Auswertung
1	Vor dir hast du ein Beispielprogramm. Klicke "Run" in der IDE und versuche im UI die eben übertragene Session zu finden und anzuschauen			4
2	Im Filesystem hast du eine abgespeicherte Session. Lade die Session über das UI.	Load Button sollte vielleicht nebst dem Tooltip eine Label haben. (Nur das Icon sieht jedoch schöner aus)	Die Testperson versucht die Datei per Doppelklick zu öffnen. In einem zweiten Versuch, ladet er die Session direkt über den Laden Button	3
3	Füge dem dritten Snapshot eine persönliche Notiz hinzu und speicherer diese für die spätere Verwendung ab.	Die InputBox sollte eine Label "Persönliche Notiz" haben	Command+S Shortcut fürs Speichern implementieren	4
4	Du möchtest nun die aktuelle Session automatisch abspielen um die Veränderungen über die Zeit betrachten zu können. Wähle dazu eine möglichst schnelle Geschwindigkeit		Das Kübel Icon sollte durch ein Kreuz ersetzt werden	4
5	Gehe zurück in die IDE und klicke noch zwei Mal auf "Run". Lösche im UI nun die zweit älteste aller Sessions	Kann man den Sessionnamen umbenennen? Wird das File nun auch auf dem Desktop gelöscht?	Die aktuelle Session in der Available Session-List soll farblich hervorgehoben werden	4
6	Zum Abschluss möchtest du noch aufräumen. Lösche dazu alle vorhandenen Sessions und schliesse das Programm	Das "Delete All"-Icon sollte gleich wie das Delete-Icon sein.		4

Usability Test: Szenarien

Infos für Testpersonen:
Handelt sich um Software zur Darstellung von Algorithmen & Datenstrukturen. Jede Durchführung eines Programms führt zu einer Session mit 1 bis mehreren Snapshots, die einzeln angesehen werden können.
Wir testen die Bedienbarkeit der Software und nicht die Fähigkeiten der Testperson!

Skala Auswertung	
Was soll ich tun??	1
War schwierig, aber hat geklappt	2
Leichtes Verbesserungspotential	3
Alles klar!	4

Anmerkungen Testperson:
Für die Prüfung (Unterlagen) wäre es cool, wenn man einen ganzen Ablauf mit Bild und Descr. Abspeichern könnte. Cool wäre, wenn man direkt zu einem gewissen Snapshot navigieren könnte.

#	Szenario	Feedback Testperson	Bemerkungen Tester	Auswertung
1	Vor dir hast du ein Beispielprogramm. Klicke "Run" in der IDE und versuche im UI die eben übertragene Session zu finden und anzuschauen	Liste -> sind das icons? "ungespeicherter Inhalt" oder Buttons?	Lange SessionNamen führen dazu, dass man die Icons nicht sieht (trash) -> sollten rechts kleben und name abkürzen	3
2	Im Filesystem hast du eine abgespeicherte Session. Lade die Session über das UI.			4
3	Füge dem dritten Snapshot eine persönliche Notiz hinzu und speicherer diese für die spätere Verwendung ab.	Zusammengehörigkeit des Inputfelds zu einzelnen steps unklar. Was ist überhaupt Funktion des Feldes?	Speicherbutton in Haupt Button bar gesucht. Aktive Session hatte keinen Hintergrund...!	3
4	Du möchtest nun die aktuelle Session automatisch abspielen um die Veränderungen über die Zeit betrachten zu können. Wähle dazu eine möglichst schnelle Geschwindigkeit	"schnell" ist immernoch recht langsam.		4
5	Gehe zurück in die IDE und klicke noch zwei Mal auf "Run". Lösche im UI nun die zweit älteste aller Sessions	Intuitiv erwartet, dass Liste alphabetisch sortiert ist		4
6	Zum Abschluss möchtest du noch aufräumen. Lösche dazu alle vorhandenen Sessions und schliesse das Programm	"Gummi" als Icon nicht intuitiv		3

Usability Test: Szenarien

Infos für Testpersonen:

Handelt sich um Software zur Darstellung von Algorithmen & Datenstrukturen. Jede Durchführung eines Programms führt zu einer Session mit 1 bis mehreren Snapshots, die einzeln angesehen werden können.

Wir testen die Bedienbarkeit der Software und nicht die Fähigkeiten der Testperson!

Skala Auswertung	
Was soll ich tun??	1
War schwierig, aber hat geklappt	2
Leichtes Verbesserungspotential	3
Alles klar!	4

#	Szenario	Feedback Testperson	Bemerkungen Tester	Auswertung
1	Vor dir hast du ein Beispielprogramm. Klicke "Run" in der IDE und versuche im UI die eben übertragene Session zu finden und anzuschauen		Wurde sofort ohne Aufforderung auf Step und Replay Funktionen aufmerksam.	4
2	Im Filesystem hast du eine abgespeicherte Session. Lade die Session über das UI.		Erste Reaktion war "auf dem Desktop Datei suchen". Erst mit Input "über das UI kannst du Files laden, wars klar	4
3	Füge dem letzen Snapshot eine persönliche Notiz hinzu und speicherer diese für die spätere Verwendung ab.			4
4	Du möchtest nun die aktuelle Session automatisch abspielen um die Veränderungen über die Zeit betrachten zu können. Wähle dazu eine möglichst schnelle Geschwindigkeit	Default Geschwindigkeit sollte besser schneller sein.		3
5	Gehe zurück in die IDE und klicke noch zwei Mal auf "Run". Lösche im UI nun die zweit älteste aller Sessions			4
6	Zum Abschluss möchtest du noch aufräumen. Lösche dazu alle vorhandenen Sessions und schliesse das Programm	Icon des "Gummies" ist nicht so intuitiv verständlich		3

Usability Test: Szenarien

Infos für Testpersonen:
Handelt sich um Software zur Darstellung von Algorithmen & Datenstrukturen. Jede Durchführung eines Programms führt zu einer Session mit 1 bis mehreren Snapshots, die einzeln angesehen werden können.
Wir testen die Bedienbarkeit der Software und nicht die Fähigkeiten der Testperson!

Skala Auswertung	
Was soll ich tun??	1
War schwierig, aber hat geklappt	2
Leichtes Verbesserungspotential	3
Alles klar!	4

Anmerkungen Testperson:
Ideal wäre eine ein Feedback wenn die Session geladen, gespeichert und gelöscht wurde. (erfolgreich/nicht erfolgreich) -> Allenfalls über Message Bar

#	Szenario	Feedback Testperson	Bemerkungen Tester	Auswertung
1	Vor dir hast du ein Beispielprogramm. Klicke "Run" in der IDE und versuche im UI die eben übertragene Session zu finden und anzuschauen			4
2	Im Filesystem hast du eine abgespeicherte Session. Lade die Session über das UI.	Ist auch Drag n Drop des Files möglich?		4
3	Füge dem dritten Snapshot eine persönliche Notiz hinzu und speicherer diese für die spätere Verwendung ab.	Ctrl + S als Shortcut um die aktuelle Session zu speichern		3
4	Du möchtest nun die aktuelle Session automatisch abspielen um die Veränderungen über die Zeit betrachten zu können. Wähle dazu eine möglichst schnelle Geschwindigkeit	Warum ist der Cancel Button der Replay Funktion nicht immer klickbar		4
5	Gehe zurück in die IDE und klicke noch zwei Mal auf "Run". Lösche im UI nun die zweit älteste aller Sessions	Die neusten Session sollten zuunterst in der Session erscheinen	Session List: Switch Time Label mit Sessionname Label	4
6	Zum Abschluss möchtest du noch aufräumen. Lösche dazu alle vorhandenen Sessions und schliesse das Programm	Der Delete All und Delete Button sollte das gleiche Label haben		3

Usability Test: Szenarien

Infos für Testpersonen:
Handelt sich um Software zur Darstellung von Algorithmen & Datenstrukturen. Jede Durchführung eines Programms führt zu einer Session mit 1 bis mehreren Snapshots, die einzeln angesehen werden können.
Wir testen die Bedienbarkeit der Software und nicht die Fähigkeiten der Testperson!

Skala Auswertung	
Was soll ich tun??	1
War schwierig, aber hat geklappt	2
Leichtes Verbesserungspotential	3
Alles klar!	4

Anmerkungen Testperson:
Rechtsklick auf Session: Open in new Window

#	Szenario	Feedback Testperson	Bemerkungen Tester	Auswertung
1	Vor dir hast du ein Beispielprogramm. Klicke "Run" in der IDE und versuche im UI die eben übertragene Session zu finden und anzuschauen			4
2	Im Filesystem hast du eine abgespeicherte Session. Lade die Session über das UI.			4
3	Füge dem dritten Snapshot eine persönliche Notiz hinzu und speicherer diese für die spätere Verwendung ab.		Werden Änderungen bei den Bemerkungen automatisch gespeichert? (Sofern die Session bereits gespeichert wurde)	4
4	Du möchtest nun die aktuelle Session automatisch abspielen um die Veränderungen über die Zeit betrachten zu können. Wähle dazu eine möglichst schnelle Geschwindigkeit			4
5	Gehe zurück in die IDE und klicke noch zwei Mal auf "Run". Lösche im UI nun die zweit älteste aller Sessions			4
6	Um Sessions zu vergleichen möchtest du ein SessionTab "detatchen". Schliesse danach das neu entstandene Fenster.		Nicht gewusst wie die Session geschlossen werden kann	2
7	Zum Abschluss möchtest du noch aufräumen. Lösche dazu alle vorhandenen Sessions und schliesse das Programm			4

Usability Test: Szenarien

Infos für Testpersonen:
Handelt sich um Software zur Darstellung von Algorithmen & Datenstrukturen. Jede Durchführung eines Programms führt zu einer Session mit 1 bis mehreren Snapshots, die einzeln angesehen werden können.
Wir testen die Bedienbarkeit der Software und nicht die Fähigkeiten der Testperson!

Skala Auswertung	
Was soll ich tun??	1
War schwierig, aber hat geklappt	2
Leichtes Verbesserungspotential	3
Alles klar!	4

Anmerkungen Testperson:
User Codebase -> launch() und disconnect() nicht so intuitive
Name: wie wärs mit startRecording() und endRecording()

#	Szenario	Feedback Testperson	Bemerkungen Tester	Auswertung
1	Vor dir hast du ein Beispielprogramm. Klicke "Run" in der IDE und versuche im UI die eben übertragene Session zu finden und anzuschauen			4
2	Im Filesystem hast du eine abgespeicherte Session. Lade die Session über das UI.			4
3	Füge dem dritten Snapshot eine persönliche Notiz hinzu und speicherer diese für die spätere Verwendung ab.	Speichern Tippfehler! Gehört Notiz zu einem snapshot oder zu allen?	Nummer des snapshots zu label hinzufügen	3
4	Du möchtest nun die aktuelle Session automatisch abspielen um die Veränderungen über die Zeit betrachten zu können. Wähle dazu eine möglichst schnelle Geschwindigkeit	Help Button mit Infos über Shortcuts wäre cool		4
5	Gehe zurück in die IDE und klicke noch zwei Mal auf "Run". Lösche im UI nun die zweit älteste aller Sessions	rechtsklick auf session in session list -> menü zum schliessen und speichern		4
6	Um Sessions zu vergleichen möchtest du ein SessionTab "detatchen". Schliesse danach das neu entstandene Fenster.		Versuchte Tab mit Draggen wieder einzufügen.	1
7	Zum Abschluss möchtest du noch aufräumen. Lösche dazu alle vorhandenen Sessions und schliesse das Programm			4

Usability Test: Szenarien

Infos für Testpersonen:

Handelt sich um Software zur Darstellung von Algorithmen & Datenstrukturen. Jede Durchführung eines Programms führt zu einer Session mit 1 bis mehreren Snapshots, die einzeln angesehen werden können.

Wir testen die Bedienbarkeit der Software und nicht die Fähigkeiten der Testperson!

Skala Auswertung	
Was soll ich tun??	1
War schwierig, aber hat geklappt	2
Leichtes Verbesserungspotential	3
Alles klar!	4

#	Szenario	Feedback Testperson	Bemerkungen Tester	Auswertung
1	Vor dir hast du ein Beispielprogramm. Klicke "Run" in der IDE und versuche im UI die eben übertragene Session zu finden und anzuschauen			4
2	Im Filesystem hast du eine abgespeicherte Session. Lade die Session über das UI.			4
3	Füge dem letzen Snapshot eine persönliche Notiz hinzu und speicherer diese für die spätere Verwendung ab.			4
4	Du möchtest nun die aktuelle Session automatisch abspielen um die Veränderungen über die Zeit betrachten zu können. Wähle dazu eine möglichst schnelle Geschwindigkeit	Algorithmus (BFS,DFS) effektiv verständlich mit Styles! =)	Pause Knopf nicht gefunden, da erwartet, dass es neben play noch einen extra knopf gibt	3
5	Gehe zurück in die IDE und klicke noch zwei Mal auf "Run". Lösche im UI nun die zweit älteste aller Sessions			4
6	Um Sessions zu vergleichen möchtest du ein SessionTab "detatchen". Schliesse danach das neu entstandene Fenster.		Versuchte Tab mit Draggen wieder einzufügen.	1
7	Zum Abschluss möchtest du noch aufräumen. Lösche dazu alle vorhandenen Sessions und schliesse das Programm			4

Usability Test: Szenarien

Infos für Testpersonen:
Handelt sich um Software zur Darstellung von Algorithmen & Datenstrukturen. Jede Durchführung eines Programms führt zu einer Session mit 1 bis mehreren Snapshots, die einzeln angesehen werden können.
Wir testen die Bedienbarkeit der Software und nicht die Fähigkeiten der Testperson!

Skala Auswertung	
Was soll ich tun??	1
War schwierig, aber hat geklappt	2
Leichtes Verbesserungspotential	3
Alles klar!	4

Anmerkungen Testperson:
Die aktive Datenstrutkur mit einem Namen anschreiben

#	Szenario	Feedback Testperson	Bemerkungen Tester	Auswertung
1	Vor dir hast du ein Beispielprogramm. Klicke "Run" in der IDE und versuche im UI die eben übertragene Session zu finden und anzuschauen			4
2	Im Filesystem hast du eine abgespeicherte Session. Lade die Session über das UI.			4
3	Füge dem dritten Snapshot eine persönliche Notiz hinzu und speicherer diese für die spätere Verwendung ab.			4
4	Du möchtest nun die aktuelle Session automatisch abspielen um die Veränderungen über die Zeit betrachten zu können. Wähle dazu eine möglichst schnelle Geschwindigkeit			4
5	Gehe zurück in die IDE und klicke noch zwei Mal auf "Run". Lösche im UI nun die zweit älteste aller Sessions			4
6	Um Sessions zu vergleichen möchtest du ein SessionTab "detatchen". Schliesse danach das neu entstandene Fenster.	Wollte das Detached Tab per Drag'n'Drop wieder in die Liste schieben		2
7	Zum Abschluss möchtest du noch aufräumen. Lösche dazu alle vorhandenen Sessions und schliesse das Programm			4

Anhang G

Metriken

Die hier aufgeführten Metriken geben eine Übersicht über die Qualität des Projekts. Einen vollständigen Einblick in alle vorhandenen Metriken gewinnt sich online (siehe G.2).

G.1 Übersicht

G.1.1 Codacy

Die Code-Qualität der ADV Projekte ist auf einem hohen Niveau. Es bestehen innerhalb der Projekte keine Code Duplikationen und es gibt keine offenen Spotbugs Issues. Codacy vergibt allen Projekten den Grade A.

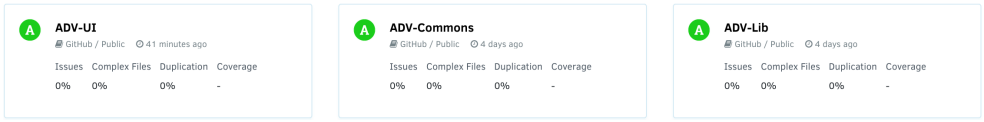


Abbildung G.1: Codacy Grades

G.1.2 Code Coverage

Die Abdeckung durch Unit Tests ist auf einem akzeptablem Level. Alle wichtigen Services und Klassen wurden getestet. Klassen, die für die Übertragung der Daten oder ausschliesslich für die Darstellung im UI zuständig sind, wurden nicht getestet. Dies führt zu einer Test-Abdeckung von ca. 50% in den Projekten ADV UI und ADV Lib. Die Klassen des ADV Commons wurden nicht getestet, da es sich um reine Datenklassen ohne Logik handelt.

adv-lib

Element	Missed Instructions	Cov.	Missed Branches	Cov.
ch.hsr.adv.lib.core.access		33%		35%
ch.hsr.adv.lib.bootstrapper		0%		0%
ch.hsr.adv.lib.core.logic		76%		36%
ch.hsr.adv.lib.core.logic.util		61%		60%
ch.hsr.adv.lib.graph.logic.domain.styles.presets		0%		n/a
ch.hsr.adv.lib.core.logic.domain.styles.presets		0%		n/a
ch.hsr.adv.lib.queue.logic		72%		100%
ch.hsr.adv.lib.stack.logic		72%		100%
ch.hsr.adv.lib.core.logic.domain.styles		49%		n/a
ch.hsr.adv.lib.graph.logic.domain		33%		n/a
ch.hsr.adv.lib.array.logic		86%		62%
ch.hsr.adv.lib.stack.logic.exceptions		0%		n/a
ch.hsr.adv.lib.queue.logic.exception		0%		n/a
ch.hsr.adv.lib.core.logic.exceptions		0%		n/a
ch.hsr.adv.lib.graph.logic		98%		75%
Total	796 of 1.774	55%	52 of 96	45%

adv-ui

Element	Missed Instructions	Cov.	Missed Branches	Cov.
ch.hsr.adv.ui.core.presentation		38%		25%
ch.hsr.adv.ui.core.presentation.widgets		41%		13%
ch.hsr.adv.ui.core.presentation.util		49%		40%
ch.hsr.adv.ui.bootstrapper		0%		0%
ch.hsr.adv.ui.core.service		70%		30%
ch.hsr.adv.ui.core.access		48%		75%
ch.hsr.adv.ui.core.logic		94%		87%
ch.hsr.adv.ui.core.logic.domain		75%		70%
ch.hsr.adv.ui.core.logic.stores		94%		85%
ch.hsr.adv.ui.graph.presentation		93%		80%
ch.hsr.adv.ui.array.presentation		100%		100%
ch.hsr.adv.ui.core.logic.events		100%		100%
ch.hsr.adv.ui.queue.presentation		100%		100%
ch.hsr.adv.ui.graph.logic		100%		n/a
ch.hsr.adv.ui.stack.presentation		100%		100%
ch.hsr.adv.ui.stack.logic		100%		n/a
ch.hsr.adv.ui.array.logic		100%		n/a
ch.hsr.adv.ui.queue.logic		100%		n/a
ch.hsr.adv.ui.core.logic.exceptions		100%		n/a
Total	4.030 of 8.827	54%	177 of 301	41%

Abbildung G.2: Coverage ADV Lib und UI

G.2 Online Metriken

Codacy	`https://app.codacy.com/organization/ADV`
CodeCov	`https://codecov.io/gh/ADVisualizer`
Travis	`https://travis-ci.org/ADVisualizer/`

Tabelle G.1: Übersicht aller Online Metrik Dashboards

Anhang H

Benutzerhandbuch

H.1 Installationsanleitung

H.1.1 Anforderungen und Hinweise

Entwicklungsumgebung

Für die Entwicklung von ADV wurde IntelliJ IDEA Ultimate 2017.3 [37] verwendet. Das Einbinden der Dependencies mit Hilfe von Gradle oder Maven sollte aber grundsätzlich auch in anderen IDE's umsetzbar sein.

Java Version

Das ADV UI setzt Java in der Version 9 oder neuer voraus. Falls neben Java 9 (oder neuer) noch eine ältere Java Version installiert ist, kann das UI spezifisch mit Java 9 (oder neuer) gestartet werden:

Windows	`'C:\Program Files\Java\jdk1.9.0_80\bin\java.exe'-jar adv-ui-1.0.jar`
Mac	`/usr/libexec/java_home -v 1.9 --exec java -jar adv-ui-1.0.jar`

Tabelle H.1: Starten mit spezifischer Java Version

Für Linux (Fedora Distribution) kann die Java Version wie folgt ausgewählt werden:

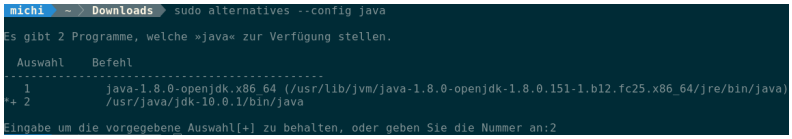


Abbildung H.1: Verändern der Java Version unter Fedora

H.2 Maven Dependencies

Der ADV ist als Maven Dependency auf jCenter oder im Maven Central Repository verfügbar. Um den ADV nutzen zu können muss im Minimum die ADV Lib im Projekt eingebunden werden. Falls zusätzlich das ADV UI eingebunden wird, wird das GUI automatisch gestartet.

Das ADV UI kann auch manuell mit einem Doppelklick auf das JAR oder mittels `java -jar adv-ui-1.0.jar` gestartet werden.

H.2.1 Gradle Setup

Bei Gradle müssen die Dependencies im `build.gradle` definiert werden.

Listing H.1: Gradle Setup

```
dependencies {
    compile 'ch.hsr.adv:adv-lib:1.0'
    compile 'ch.hsr.adv:adv-ui:1.0'
}
```

H.2.2 Maven Setup

Bei Maven müssen die Dependencies im `pom.xml` definiert werden.

Listing H.2: Maven Setup

```
<dependency>
    <groupId>ch.hsr.adv</groupId>
    <artifactId>adv-lib</artifactId>
    <version>1.0</version>
</dependency>
<dependency>
    <groupId>ch.hsr.adv</groupId>
    <artifactId>adv-ui</artifactId>
    <version>1.0</version>
</dependency>
```

H.3 Starten des ADV-UI

Das UI kann manuell über die Konsole gestartet werden:

```
java -jar adv-ui-<version>.jar
```

Alternativ startet sich das UI automatisch, wenn das UI beim Ausführen der `ADV.launch` Methode nicht bereits läuft.

H.4 Konfiguration Host und Port

Per Default läuft das ADV UI auf `localhost:8765`. Und die ADV Lib verbindet bei Benutzung automatisch auf diesen Port.

Bei Portkonflikten oder wenn das ADV UI remote betrieben wird, können Host und Port über die Verwendung von *Command Line Arguments* konfiguriert werden. Die ausgewählte Konfiguration muss immer für ADV Lib und ADV UI identisch verwendet werden.

Argument	Funktion	Beispiel
`--port`	Ändert den verwendeten Port	`--port=9876`
`--host`	Ändert den verwendeten Host	`--host=152.96.233.95`

Tabelle H.2: CLI: Command Line Interface Argumente

H.5 Konfigurationsfiles verändern

Um Änderungen an Konfigurationsfiles durchzuführen, die im JAR File enthalten sind, müssen folgende Schritte befolgt werden.

1. JAR File mit einem beliebigen File Archiver entpacken
2. In den entstandenen Ordner wechseln
   a) `cd adv-ui`
3. gewünschte Files verändern
4. Files wieder packetieren
   a) `jar cfm ../adv-ui.jar META-INF/MANIFEST.MF ch/hsr/adv/ui/bootstrapper/Bootstrapper *`

Folgende Einstellungen können vorgenommen werden:

File	Einstellung
`adv-ui‣logback.xml`	Festlegen des Log-Levels (siehe H.5.1)
`adv-ui‣css‣global.css`	Farben

Tabelle H.3: ADV Konfigurationsfiles

H.5.1 Ändern des Log-Level

Im XML File

Die XML Konfigurationsdatei befindet sich unter `adv-ui‣logback.xml`.
Das applikations-übergreifende Log Level kann über das Root Element bestimmt werden:

Listing H.3: Root Log Level verändern

```
<root level="DEBUG"> <!-- Log Level -->
<appender-ref ref="FILE" /> <!-- Log to File -->
<appender-ref ref="STDOUT" /> <!-- Log to Standard Output-->
</root>
```

Es ist auch möglich, dass Log Level für einzelnen Packages zu setzen. Dazu muss ein neues Logger Element erstellt werden.

Listing H.4: Package Log Level verändern

```
<logger name="ch.hsr.adv.ui-core.access" level="DEBUG">
<appender-ref ref="STDOUT" /> <!-- Log to Standard Output-->
</logger>
```

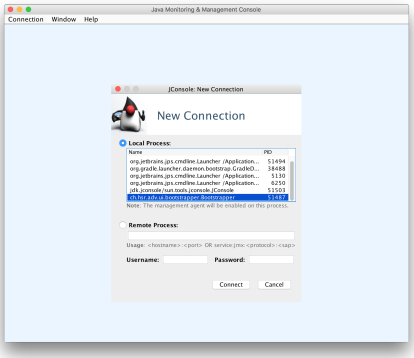
Mögliche Level sind:

- OFF
- TRACE
- DEBUG
- INFO
- WARN
- ERROR

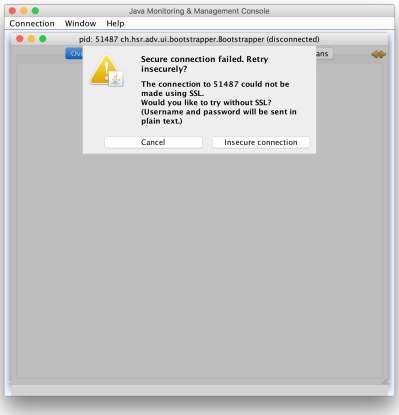
Zur Laufzeit

Über die JConsole kann das Log Level auch zur Laufzeit verändert werden.
Die JConsole kann einfach über die Konsole geöffnet werden:

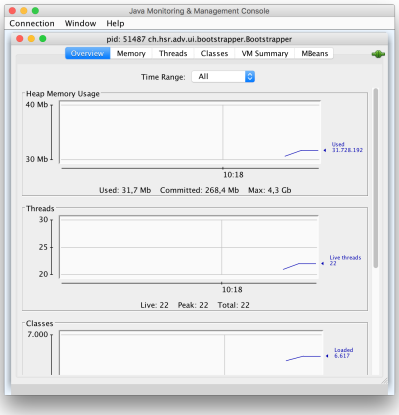
```
$bash> jconsole
```



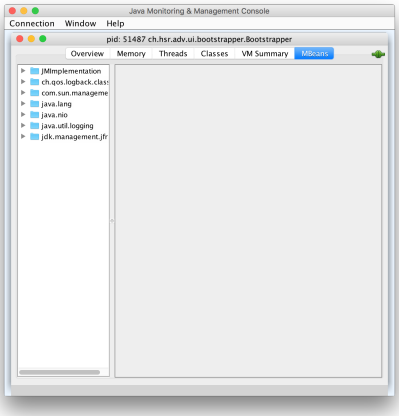
adv Bootstrapper aus Liste auswählen und mit Connect bestätigen



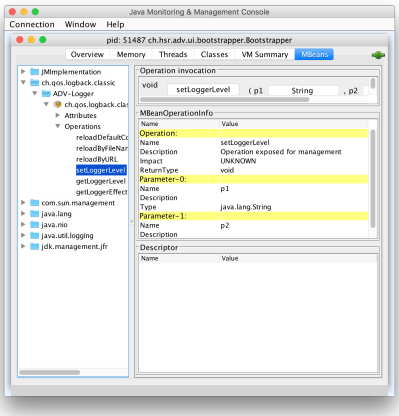
Warnung über unsichere Verbindung bestätigen



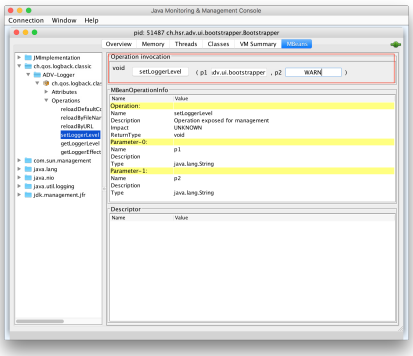
Ansicht *Overview* erscheint



Zu Ansicht MBeans wechseln



Links im Verzeichnis-Baum zu `ch.qos.logback.classic` ▸ `ADV-Logger` ▸ `Operations` ▸ `setLoggerLevel` wechseln



Im oberen Fenster-Bereich unter Operation invocation gewünschte Parameter angeben

1. Parameter: package (e.g. `ch.hsr.adv.ui.bootstrapper`)
2. Parameter: Log Level (e.g. `WARN`)

Mit setLoggerLevel bestätigen

H.6 Benutzeroberfläche und Funktionen

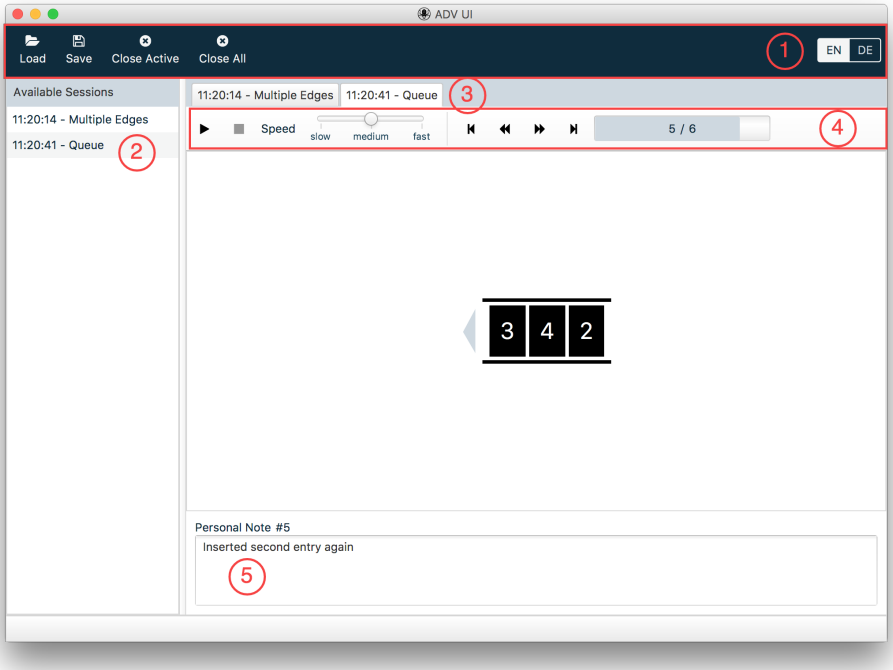


Abbildung H.2: Benutzeroberfläche des ADV UI

(1) Toolbar

Load
Laden eines *.adv Files

Save
Abspeichern einer Session als *.adv File

Close Active
Schliessen der aktiven Session

Close All
Schliessen aller Sessions

EN/DE
Wechseln der Sprache. Verfügbare Sprachen sind Englisch (EN) und Deutsch (DE)

(2) Sessionlist
Listet alle aktiven Sessions sortiert nach Aktualität mit der neusten Session zuunterst. Durch Klicken auf eine Session kann zwischen den einzelnen Sessions gewechselt werden.

(3) Session Tabs
Jede aktive Session wird in einem Tab angezeigt. Durch Klicken auf ein Tab kann zwischen den einzelnen Sessions gewechselt werden. Durch Drag-and-Drop kann ein Tab in ein eigenes Fenster verschoben werden (siehe H.6.1).

(4) Session Toolbar

Replay and Pause
Spielt alle Snapshots einer Session ab oder Pausiert das Abspielen

Cancel
Bricht das Abspielen ab und Navigiert zum ersten Snapshot der Session

Speed
Stellt die Geschwindigkeit des Replays ein

Step First
Navigiert zum ersten Snapshot der Session

Step Backward
Navigiert zum vorherigen Snapshot der Session

Step Forward
Navigiert zum nächsten Snapshot der Session

Step Last
Navigiert zum letzten Snapshot der Session

Progress Bar
Zeigt welcher Snapshot aktuell angezeigt wird

(5) Personal Note
Zeigt eine Notiz für den aktuellen Snapshot. Kann editiert werden. Wird beim Speichern der Session persistiert.

H.6.1 Detachable Tabs

Um Sessions gut miteinander vergleichen zu können, sind die Tabs des ADV *detachable*. Durch *Drag and Drop* können Tabs in ein eigenes Fenster verschoben werden. Beim Schliessen des entstandenen Fensters, wird das Tab wieder im Hauptfenster eingereiht. Eine Session kann wie gewohnt durch Klicken auf Close Active geschlossen werden, auch wenn das Tab *detached* ist.

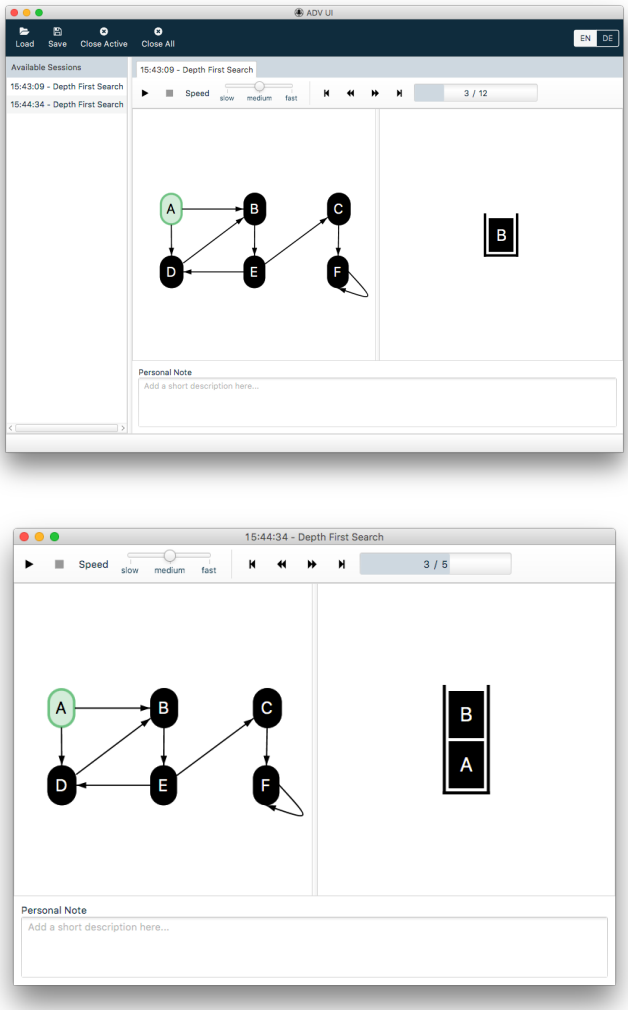


Abbildung H.3: Vergleich von Session im Hauptfenster (oben) mit Session im *detachten* Tab (unten)

H.6.2 Shortcuts

Um die Nutzung des GUI zu beschleunigen, sind verschiedene Funktionen mit Shortcuts belegt.

Hinweis: Für macOS anstelle von ctrl jeweils cmd verwenden

Shortcut	Funktion
ctrl + O	Laden einer Session
ctrl + S	Speichern der aktiven Session
ctrl + W	Schliessen der aktiven Session
ctrl + ⇧ + W	Schliessen aller Sessions
ctrl + R	Replay starten bzw. pausieren
esc	Replay abbrechen
ctrl + ⇧ + ←	Zum ersten Snapshot springen
ctrl + ←	Zum vorherigen Snapshot springen
ctrl + →	Zum nächsten Snapshot springen
ctrl + ⇧ + →	Zum letzten Snapshot springen

Tabelle H.4: Shortcuts

H.7 ADV Library

H.7.1 Methoden

`ADV.launch(String[]);`	Stellt eine Verbindung zum ADV UI her. Falls das ADV UI noch nicht läuft, wird es automatisch gestartet. Übergeben werden der Methode die Argumente der Main Methode. Für die lokale Verwendung des ADV UI ist die Methode optional. Beim Aufruf des ersten Snapshots wird der Launch bei Bedarf ebenfalls ausgeführt.
`ADV.snapshot(ADVModule, String);`	sendet den momentanen Zustand des Modules als Snapshot an das ADV UI. Optional kann eine Snapshot-Beschreibung beigefügt werden.
`ADV.disconnect();`	Schliesst die Verbindung zum ADV UI. Diese Methode muss nur aufgerufen werden, falls die User-Applikation nicht terminiert. Ansonsten wird die Verbindung automatisch bei Programmschluss getrennt.

Tabelle H.5: Verfügbare ADV Library Methoden

H.7.2 Styles

Styles werden genutzt um Elemente und *Relations* zu stylen. Es können vordefinierte Styles verwendet oder eigene Styles erstellt werden.

Presets

Zur einfacheren Verwendung sind bereits einige fertige Styles verfügbar.

Stylename	Preview
ADVDefaultStyle	
ADVSuccessStyle	
ADVInfoStyle	
ADVWarningStyle	
ADVErrorStyle	

Tabelle H.6: Preset Styles

Eigene Styles

Eine Implementation des *ADVStyle*-Interfaces besteht aus vier Bereichen:

- FillColor: Die Hintergrundfarbe
- StrokeColor: Die Farbe der Umrandung
- StrokeThickness: Die Dicke der Umrandung
- StrokeStyle: der Stil der Umrandung

Farben können als hex-Values, Thickness als Pixel-Wert und Style als String angegeben werden.

Zur angenehmeren Verwendung sind bereits einige vordefinierte Values verfügbar (siehe Tabellen H.7 und H.8). Im Code Ausschnitt H.5 ist zu sehen, wie Styles verwendet werden können.

Listing H.5: Style Example

```
import ch.adv.lib.core.domain.styles.*;
import ch.adv.lib.core.domain.styles.presets.*;

ADVStyle style = new ADVSuccessStyle();
style = new ADVEnumStyle(ADVColor.ORANGE, ADVStrokeStyle.SOLID,
    ADVStrokeThickness.THIN);
```

STANDARD	BLACK	WHITE
RED_LIGHT	RED	RED_DARK
PURPLE_LIGHT	PURPLE	PURPLE_DARK
BLUE_LIGHT	BLUE	BLUE_DARK
GREEN_LIGHT	GREEN	GREEN_DARK
YELLOW_LIGHT	YELLOW	YELLOW_DARK
ORANGE_LIGHT	ORANGE	ORANGE_DARK
BROWN_LIGHT	BROWN	BROWN_DARK
GRAY_LIGHT	GRAY	GRAY_DARK

Tabelle H.7: Verfügbare ADV Colors

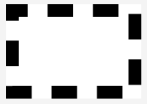
	STAN-DARD	THIN	MEDIUM	THICK
SOLID				
DASHED				
DOTTED				

Tabelle H.8: Thickness und Style

H.8 Modules

H.8.1 *Array Module*

Quick Guide

Library Klassen	
ADV	Einstiegspunkt in die Applikation
ArrayModule	Kapselt darzustellende Daten. Dem Konstruktor muss ein Array übergeben werden.
Interfaces	
keine	
Methoden	
arrayModule.getStyleMap()	Map zum Mappen von Styles auf Array Indexe
arrayModule.setShowObjectRelations(true);	Aktiviert das Anzeigen von Objekt-Referenzen.

Tabelle H.9: Quick Guide Array

Hinweis H.8.1: Ausführlichere Beschreibungen

Ausführlichere Beschreibungen der Klassen und Methoden finden sich in den Java Docs im ADV Lib Projekt.

Ausführliche Anleitung

Mit dem *Array Module* können Arrays angezeigt werden.

Da das *Array Module* einen generischen Array als Argument erwartet, können keine primitiven Arrays verwendet werden. Die verwendeten Objekt-Arrays können jedoch wahlweise mit oder ohne Objekt-Referenzen angezeigt werden.

Code Ausschnitt H.6 zeigt, wie zwischen den Varianten gewechselt werden kann.

Listing H.6: Array

```
public class MyArray {
    // instantiate data structure container
    private static final Integer[] array = new Integer[10];
    // library class to be used
    private static final ArrayModule arrayModule = new ArrayModule("Primitive
        Array", array);

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

        array[0] = 0;
        array[1] = 1;
        array[3] = 3;
        array[5] = 5;
        // send current state of array to the UI to be displayed
        ADV.snapshot(arrayModule, "Empty indices are default initialized with
            null");

        // show actual object references
        arrayModule.setShowObjectRelations(true);
        ADV.snapshot(arrayModule, "Empty indices are default initialized with
            null");
    }
}
```

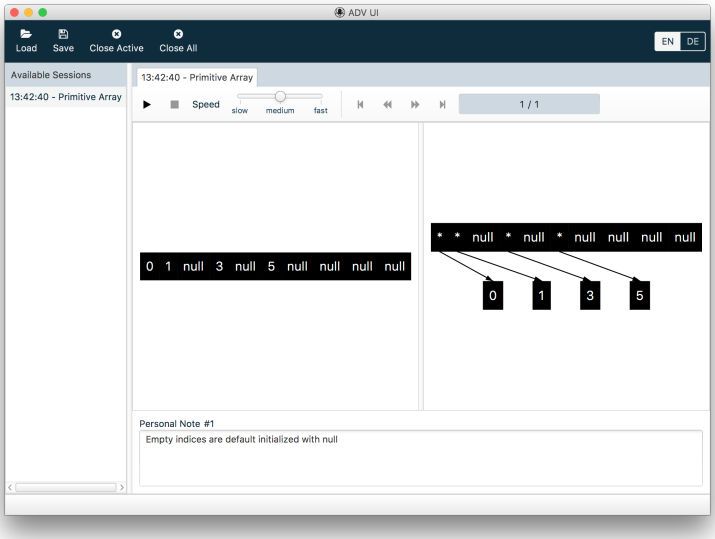


Abbildung H.4: Array Module: links ohne und rechts mit Objekt-Referenzen

Mir der Hilfe von Styles können Algorithmen auf einem Array, wie zum Beispiel ein Bubble Sort, visualisiert werden. Um Elemente eines Arrays zu Stylen müssen die Styles zusammen mit dem Index des Elements in der StyleMap gespeichert werden. Was für vorgefertigte Styles vorhanden sind oder wie eigene Styles erstellt werden können, ist im Abschnitt H.7.2 erklärt.

Listing H.7: Array mit Styles

```
public class StylesShowcase {
    // instantiate data structure container
    private static final int LENGTH = 1;
    private static final Integer[] array = new Integer[LENGTH];
    private static final ArrayModule arrayModule = new ArrayModule("Styles
        Showcase", array);

    public static void main(String[] args) throws ADVException {
        ADV.launch(args);
        // set style for array element on index 0
        setStyle(0, new ADVDefaultElementStyle());

        ADV.snapshot(arrayModule, "Using available preset styles.");
    }

    private static void setStyle(int i, ADVStyle style) {
        arrayModule.getStyleMap().put(i, style);
    }
}
```

Hinweis H.8.2: Ausführlichere Beispiele

Ausführlichere Beispiele finden sich in der *User Codebase* auf Github: `https://github.com/ADVisualizer/ADV-User_Codebase`

H.8.2 *Queue Module*

Quick Guide

Library Klassen	
ADV	Einstiegspunkt in die Applikation
QueueModule	Kapselt darzustellende Daten. Dem Konstruktor muss eine ADVQueue übergeben werden.
Interfaces	
ADVQueue<T>	Eigene Queue-Implementierung muss dem *QueueModule* als Konstruktor-Argument übergeben werden.
Methoden	
queueModule.getStyleMap()	Map zum Mappen von Styles auf Queue Positionen (0 = Head)

Tabelle H.10: Quick Guide Queue

Hinweis H.8.3: Ausführlichere Beschreibungen

Ausführlichere Beschreibungen der Klassen und Methoden finden sich in den Java Docs im ADV Lib Projekt.

Ausführliche Anleitung

Mit dem *Queue Module* können verschiedenste Implementationen von Queues angeschaut werden (FIFOQueue, PriorityQueue, ...). Das vordere Ende der Queue (”Head”) ist durch einen Pfeil markiert.

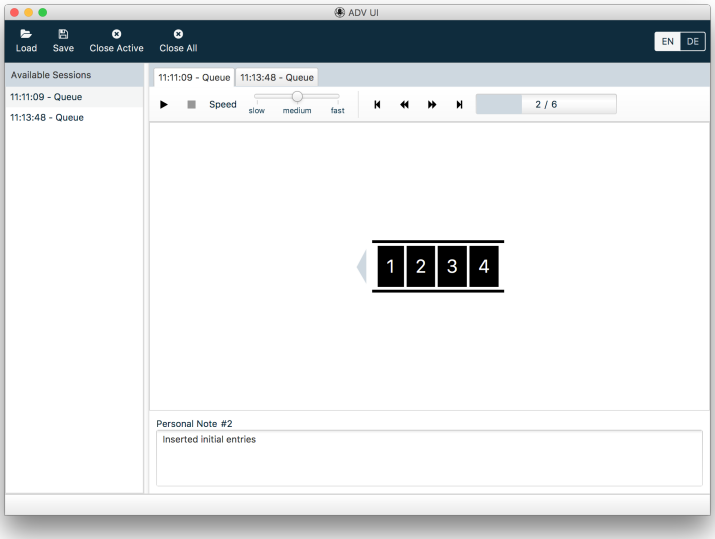


Abbildung H.5: Queue Module

Listing H.8: Queue

```
public class QueueShowcase {
    // user implementation of Queue Interface
    private static final ADVQueue<Integer> queue = new FIFOQueue<>();
    // library class to be used
    private static final QueueModule module = new QueueModule("Queue", queue);

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

        queue.insert(1);
        queue.insert(2);
        queue.insert(3);
        queue.insert(4);
        // send current state of queue to the UI to be displayed
        ADV.snapshot(module, "Inserted entries");
    }
}
```

Hinweis H.8.4: Ausführlichere Beispiele

Ausführlichere Beispiele finden sich in der *User Codebase* auf Github: `https://github.com/ADVisualizer/ADV-User_Codebase`

Falls Styles verwendet werden möchten, können diese über eine StyleMap gesetzt werden. Styles sind an Positionen in der Queue und nicht an einzelne Elemente gebunden. Index 0 in der StyleMap entspricht dem ersten Element (”Head”) in der Queue. Was für vorgefertigte Styles vorhanden sind oder wie eigene Styles erstellt werden können, ist im Abschnitt H.7.2 erklärt.

Listing H.9: Queue Styles

```
public class QueueShowcase {
    private static final ADVQueue<Integer> queue = new FIFOQueue<>();
    private static final QueueModule module = new QueueModule("Queue", queue);

    public static void main(String[] args) throws ADVException {
        ADV.launch(args);

        queue.insert(1);
        queue.insert(2);
        queue.insert(3);
        queue.insert(4);
        // set style for head element
        setStyle(0,new ADVWarningStyle());
        // set style for second element in queue
        setStyle(1,new ADVSuccessStyle());
        ADV.snapshot(module, "Inserted entries");
    }

    private static void setStyle(int i, ADVStyle style) {
        module.getStyleMap().put(i, style);
    }
}
```

H.8.3 *Stack Module*

Quick Guide

Library Klassen	
ADV	Einstiegspunkt in die Applikation
StackModule	Kapselt darzustellende Daten. Dem Konstruktor muss eine ADVStack übergeben werden.
Interfaces	
ADVStack<T>	Eigene Stack-Implementierung muss dem *StackModule* als Konstruktor-Argument übergeben werden.
Methoden	
stackModule.getStyleMap()	Map zum Mappen von Styles auf Stack Positionen (0 = Top of Stack)

Tabelle H.11: Quick Guide Stack

Hinweis H.8.5: Ausführlichere Beschreibungen

Ausführlichere Beschreibungen der Klassen und Methoden finden sich in den Java Docs im ADV Lib Projekt.

Ausführliche Anleitung

Mit dem *Stack Module* können eigene Stack Implementierungen angeschaut werden. Stack Elemente können mit Styles eingefärbt werden. Dies geschieht mit Hilfe einer StyleMap. Das heisst Styles werden auf Positionen im Stack und nicht auf Elemente im Stack gebunden. So kann zum Beispiel der Top-of-Stack farblich hervorgehoben werden. Dabei entspricht Index 0 dem Top of Stack und Index `Stack.size()-1` dem untersten Element im Stack. Was für vorgefertigte Styles vorhanden sind oder wie eigene Styles erstellt werden können, ist im Abschnitt H.7.2 erklärt.

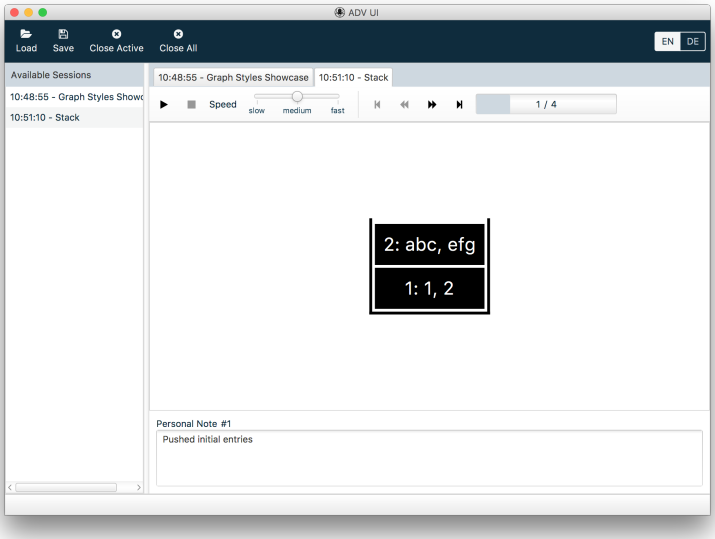


Abbildung H.6: Stack Module

Listing H.10: Stack

```
public class StackShowcase {
    // user implementation of Stack Interface
    private static final ADVStack<Integer> stack = new Stack<>();
    // library class to be used
    private static final StackModule stackModule = new StackModule("Stack",
        stack);

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

        stack.push(1);
        stack.push(2);

        // set style for top of stack
        setStyle(0, new ADVSuccessStyle());

        // send current state of array to the UI to be displayed
        ADV.snapshot(stackModule, "Pushed styled entries");
    }

    private static void setStyle(int i, ADVStyle style) {
        stackModule.getStyleMap().put(i, style);
    }
}
```

Hinweis H.8.6: Ausführlichere Beispiele

Ausführlichere Beispiele finden sich in der *User Codebase* auf Github:
`https://github.com/ADVisualizer/ADV-User_Codebase`

H.8.4 *Graph Module*

Quick Guide

Library Klassen	
ADV	Einstiegspunkt in die Applikation
GraphModule	Kapselt darzustellende Daten. Dem Konstruktor muss eine ADVGraph übergeben werden.
Interfaces	
ADVGraph<Vertex, Edge>	Eigene Graph-Implementierung muss dem *GraphModule* als Konstruktor-Argument übergeben werden.
ADVVertex<T>	Interface für eigene Vertex-Implementierung.
ADVEdge<E>	Interface für eigene Edge-Implementierung.
Methoden	
graph.addVertices(Vertex... vertices)	Hinzufügen von neuen Vertices zum Graphen.
graph.addEdges(Edge... edges)	Hinzufügen von neuen Edges zum Graphen.
vertex.setFixedPosX(int x)	Vertices müssen manuell platziert werden.
vertex.setFixedPosY(int Y)	Vertices müssen manuell platziert werden.
vertex.setStyle(ADVStyle style)	Setzen von Styles für einen Vertex
edge.setStyle(ADVStyle style)	setzen von Styles für eine Kante

Tabelle H.12: Quick Guide Graph

Hinweis H.8.7: Ausführlichere Beschreibungen

Ausführlichere Beschreibungen der Klassen und Methoden finden sich in den Java Docs im ADV Lib Projekt.

Ausführliche Anleitung

Mit dem *Graph Module* können verschiedene Graph-Implementierungen und Algorithmen auf Graphen dargestellt werden. Das *Graph Module* unterstützt gerichtete, ungerichtete, azyklische und zyklische Graphen. Selbstreferenzen und bis zu zwei Verbindungen zwischen den gleichen Nodes können dargestellt werden. Styles helfen bei der Visualisierung von Algorithmen wie Dijkstra oder DFS. Was für vorgefertigte Styles vorhanden sind oder wie eigene Styles erstellt werden können, ist im Abschnitt H.7.2 erklärt.

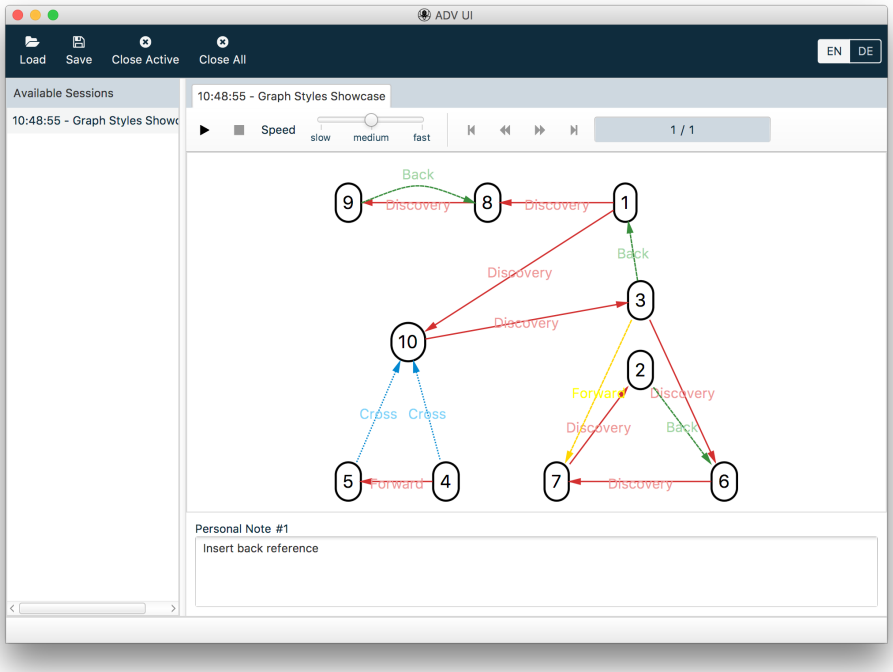


Abbildung H.7: Graph Module

Listing H.11: Graph

```
public class GraphShowcase {
    // user implementation of Graph Interface
    private static final Graph graph = new Graph();
    // library class to be used
    private static final GraphModule graphModule = new GraphModule("Breadth
        First Search", graph);

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

        // instantiate user implementation of Vertex Interface
         Vertex<String> a = new Vertex<>("A");
        a.setFixedPosX(20);
        a.setFixedPosY(20);

        Vertex<String> b = new Vertex<>("B");
        b.setFixedPosX(100);
        b.setFixedPosY(20);

        graph.addVertices(a, b);

        Edge<Integer> ab = new Edge<>(a, b, true);
        graph.addEdges(ab);

        // send current state of graph to the UI to be displayed
        ADV.snapshot(graphModule, "Initial state");

        ab.setStyle(new ADVDiscoveryEdgeStyle());
        ADV.snapshot(graphModule, "show style");
}
```

Hinweis H.8.8: Ausführlichere Beispiele

Ausführlichere Beispiele finden sich in der *User Codebase* auf Github: `https://github.com/ADVisualizer/ADV-User_Codebase`

H.8.5 *Child Modules*

Modules können auch kombiniert werden. Dazu können an ein *Module* beliebig viele *Child Modules* angehängt werden. Dies erlaubt es neben einer Hauptstruktur auch eine oder mehrere Hilfsstrukturen anzuzeigen.

Der `snapshop`-Methode muss jeweils das Haupt-*Module* übergeben werden. Im ADV UI werden die *Modules* dann in einem Grid angezeigt. Die entstandenen Dividers können an die gewünschten Positionen gezogen werden.

Code Ausschnitt H.12 zeigt ein Beispiel eines Haupt-*Modules* mit drei Child-*Modules*. Abbildung H.8 zeigt, wie der Snapshot im UI aussieht.

Listing H.12: Child Modules

```
public class MultiModule {
    // Datastructures
    private static final Stack<Integer> stack = new Stack<>();
    private static final FIFOQueue<Integer> queue = new FIFOQueue<>();
    private static final Graph graph = new Graph();
    private static final Integer[] array = new Integer[5];

    // ADVModules
    private static final String SESSION_NAME = "Multiple Modules";
    private static final QueueModule queueModule = new
        QueueModule(SESSION_NAME, queue);
    private static final GraphModule graphModule = new
        GraphModule(SESSION_NAME, graph);
    private static final StackModule stackModule = new
        StackModule(SESSION_NAME, stack);
    private static final ArrayModule arrayModule = new
        ArrayModule(SESSION_NAME, array);

    public static void main(String[] args) throws ADVException {

        ADV.launch(args);

        // instantiate data structure container
        queueModule.addChildModule(arrayModule);
        queueModule.addChildModule(stackModule);
        queueModule.addChildModule(graphModule);

        fillHelpers();
        ADV.snapshot(queueModule, "Showing stack, queue, array and graph
            module");

        fillGraph();
        ADV.snapshot(queueModule, "Filling graph from helpers");
    }
}
```

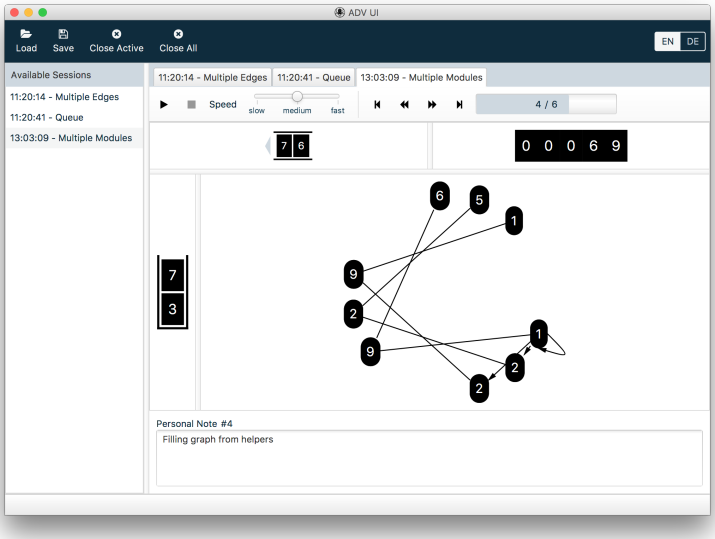


Abbildung H.8: Child Modules

Hinweis H.8.9: Ausführlichere Beispiele

Ausführlichere Beispiele finden sich in der *User Codebase* auf Github: `https://github.com/ADVisualizer/ADV-User_Codebase`

Anhang I

Handbuch zur Modulentwicklung

I.1 Einleitung

Dieses Handbuch beschreibt die wichtigsten Schritte, um möglichst schnell eine funktionsfähige Entwicklungsumgebung aufsetzen zu können. Zudem werden wiederverwendbare Komponenten vorgestellt, die vom Framework zu Verfügung gestellt werden. Es ist wichtig, dass die folgenden Anweisungen befolgt werden, damit der ADV einheitlich weiterentwickelt wird. Dies erleichtert den Einstieg für zukünftige Mitwirkende am Projekt.

I.2 Accounts

Um den ADV weiterentwickeln zu können, werden einige Benutzerkonten vorausgesetzt. Insbesondere wird ein *Github* Account benötigt. Für das Projekt wurde eine Github-Organisation gegründet, bei welcher jede(r) Mitwirkende Mitglied sein muss.

Account	URL	Provider
Github	`https://github.com/join`	-
Bintray	`https://bintray.com/signup/oss`	Github
Travis CI	`https://travis-ci.org/`	Github
Codacy	`https://www.codacy.com/signup`	Github
Codecov	`https://www.codacy.com/signup`	Github

Tabelle I.1: Benutzerkonten für die Entwicklung

I.3 Git Repositories

Der Quellcode befindet sich auf Github: `https://github.com/ADVisualizer`. Eine genaue Auflistung der Projekte ist der Tabelle I.2 zu entnehmen.

Projekt	Beschreibung
ADV-Lib	API Klassen damit der ADV genutzt werden kann
ADV-UI	JavaFX Applikation welche für die Visualisierung der Lib Klassen zuständig ist
ADV-Commons	Enthält gemeinsam verwendete POJO Klassen der Container UI und Lib
ADV-User_Codebase Github	Enthält Beispiel Anwendungen für alle verfügbare ADV Modules
ADV-Starter-Gradle	Gradle Boilerplate Projekt
ADV-Starter-Maven	Maven Boilerplate Projekt

Tabelle I.2: Github Code Repositories

I.4 Projektstruktur

Der ADV setzt sich aus drei Projekten zusammen: ADV UI, ADV Lib und ADV Commons. Das Commons Projekt bündelt die Gemeinsamkeiten der beiden Container ADV UI und ADV Lib.

Alle drei Projekte sind Gradle Multi-Projects [31]. Die Projektstruktur ist in allen drei Projekten gleichermassen organisiert. Eine Übersicht gibt die Tabelle I.3.

Jedes Projekt enthält einen Core, welcher modulübergreifende Funktionalitäten kapselt. Die beiden Projekte ADV UI und ADV Lib enthalten zudem eine Bootstrapper-Komponente, welche den Dependency Injection Container initialisiert. Modul-spezifische Klassen werden im entsprechenden Module Projekt abgelegt. Die Modul Projekte entsprechen folgendem Namensschema: *module-[name]*. Um Code-Duplikationen zu minimieren, sollen gemeinsame Klassen des ADV UI und AVD Lib im ADV Commons abgelegt werden.

ADV Lib Projekt	ADV UI Projekt
ADV-Lib ├── bootstrapper ├── lib-core ├── module-array ├── module-graph ├── module-queue ├── module-stack │ └── src │ ├── main │ │ ├── java │ │ │ └── *package-structure* │ │ └── resources │ └── test └── *module-xyz*	ADV-UI ├── bootstrapper ├── ui-core ├── module-array ├── module-graph ├── module-queue ├── module-stack │ └── src │ ├── main │ │ ├── java │ │ │ └── *package-structure* │ │ └── resources │ └── test └── *module-xyz*

Tabelle I.3: Projekt Struktur ADV UI und ADV Lib

I.4.1 Referenz-*Modules*

Als Anschauungsbeispiel dienen die bestehenden Modules ***Array***, ***Stack***, ***Queue*** und ***Graph***. Diese sind bereits in den Projekten ADV UI, ADV Lib und ADV Commons integriert. Genauere Informationen zu diesen *Modules* finden sich in der ADV-Dokumentation. [10]

I.5 Installationsanleitung

I.5.1 Entwicklungsumgebung

Als IDE wird IntelliJ IDEA [37] empfohlen, da diese Gradle Multiprojects [31] besonders gut unterstützt. In IntelliJ kann das Projekt ohne zusätzliche Konfigurationen weiterentwickelt werden. Wie das Projekt in Eclipse importiert werden kann, ist im Abschnitt I.5.3 beschrieben.

I.5.2 Projekt importieren

In der IntelliJ IDEA unter File > Open das entsprechende ADV-Projekt öffnen. Die Teilprojekte werden als *Gradle Submodules* importiert.

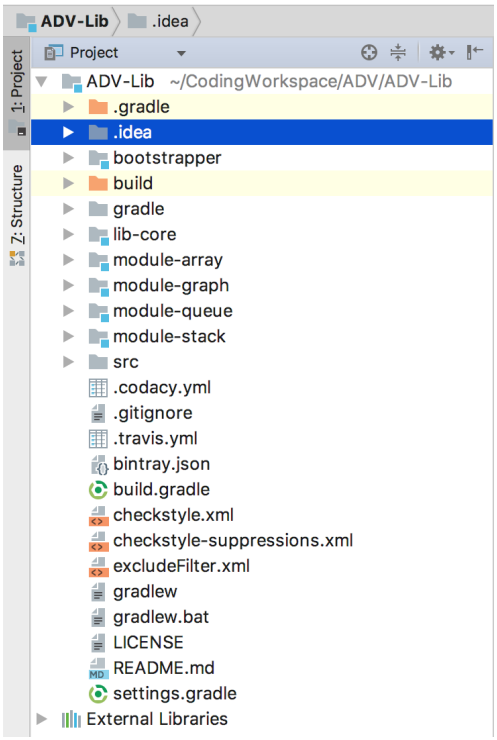


Abbildung I.1: Ansicht nach dem Import in IntelliJ IDEA

I.5.3 Eclipse Setup

Unter Eclipse muss vorerst das Plugin Buildship [23] installiert werden. Anschliessend können die ADV Projekte via File > Import > Existing Gradle Project importiert werden. Es wird empfohlen die Projekte in ein eigenständiges *Working Set* zu importieren.

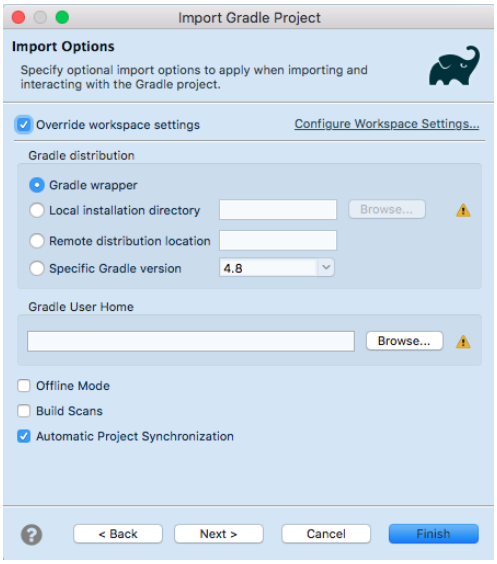


Abbildung I.2: Projektimport in Eclipse mit dem Buildship Plugin

Damit die Subprojekte ansprechender dargestellt werden, kann im *Project Explorer* die Ansicht *Hierarchical* ausgewählt werden (siehe I.3).

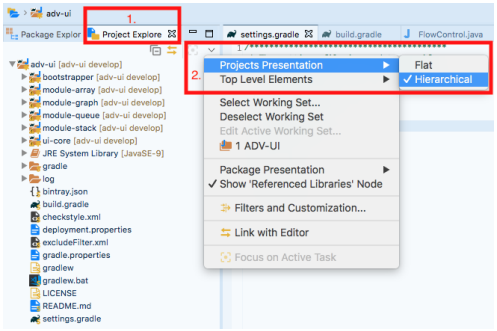


Abbildung I.3: Hierarchische Ansicht der Gradle Submodules in Eclipse

Für den Fall, dass die lokale Gradle Version Probleme mit Java 9 bekundet, muss im Home Verzeichnis der Gradle Installation ein Property File (`gradle.properties`) erstellt werden, welches auf die aktuelle Java Version verweist.

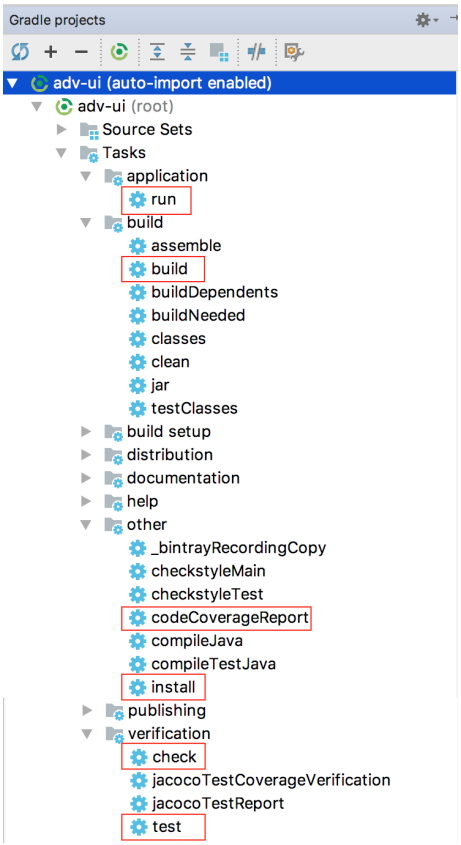
Listing I.1: Property File im Gradle Home Verzeichnis

```
org.gradle.java.home=/Library/Java/JavaVirtualMachines/jdk-9.0.4.jdk/Contents/Home
```

I.5.4 Werkzeuge

Gradle

Als Build Tool wird Gradle verwendet. Gradle Tasks können direkt aus der Entwicklungsumgebung aufgerufen werden.



Die wichtigsten Gradle Tasks:

1. *run*: startet die Applikation
2. *build*: erzeugt **ein** JAR File aus allen *Gradle Submodules* im Ordner `build‣libs` und führt die Tests aus.
3. *install*: Publiziert das Projekt ins lokale Maven Repo (`~/.m2`).
4. *codeCoverageReport*: Aggregiert die Testabdeckung aller *Gradle Submodules* und erstellt einen Report unter `build‣reports‣jacoco`.
5. *check*: Überprüft die Codequalität (Spotbugs, Checkstyle) und führt die Tests aus.
6. *test*: Führt alle Unit-Tests aus

Spotbugs

Spotbugs hilft die Code-Qualität des Projekts zu überprüfen. Entdeckte Fehler führen zu einem *Build Fail*. Die Reports werden für jedes Subprojekt separat im Ordner `build‣reports‣spotbugs` abgelegt. [61]

Checkstyle

Checkstyle [15] überprüft die Formatierung sowie weitere Qualitätsattribute des Codes. Es wird eine adaptierte Version der Sun Code Conventions [17] verwendet. Checkstyle Warnings führen nicht zu einem *Build Fail*. Die Reports können lokal für jedes Subprojekt im Ordner `build‣reports‣checkstyle` oder etwas übersichtlicher auf Codacy (siehe I.6.2) eingesehen werden.

Es wird empfohlen, Checkstyle direkt in der IDE zu integrieren, damit der IDE-Formatter die Checkstyle Rules berücksichtigt. In IntelliJ muss dazu das Checkstyle Plugin [16] installiert werden. In der Konfiguration des Plugins, muss das ADV-Checkstyle XML angegeben werden. Dieses liegt im Root Verzeichnis des Projekts.

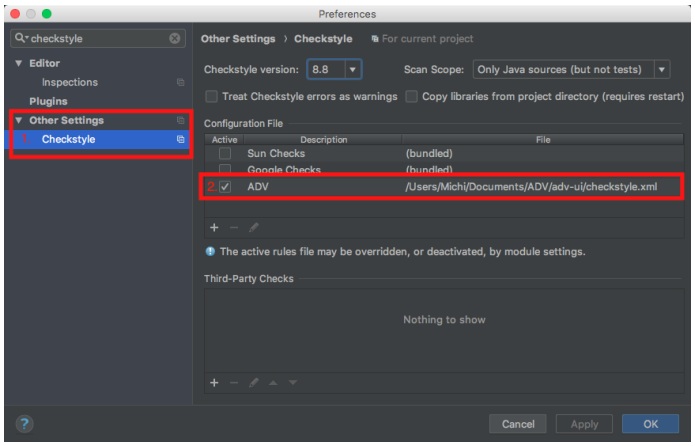


Abbildung I.4: Integration des Checkstyle Plugin in IntelliJ IDEA

Structure 101

Structure101 [64] gibt einen guten Überblick über die Abhängigkeiten innerhalb des Projekts. Beim Entwickeln neuer Modules soll darauf geachtet werden, dass die angedachte Architektur nicht verletzt wird.

Abbildung I.5 zeigt die Package-Struktur am Beispiel der ADV UI. Architekturfehler im Sinne von Tangles, werden von Structure101 als gestrichelter Pfeil angezeigt.

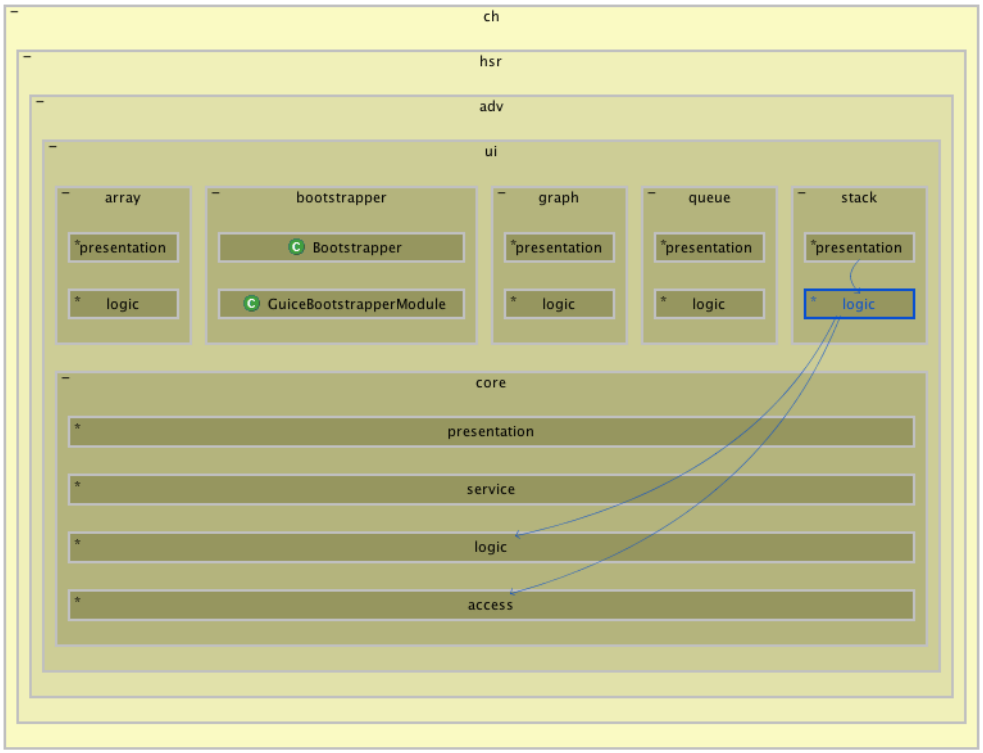


Abbildung I.5: Structure101 Architekturdiagramm

Jacoco

Jacoco [39] dient dem Erstellen von Test-Coverage Reports. Die Reports können mit den Gradle Tasks `jacocoTestReport` und `codeCoverageReport` generiert werden. Letzterer Task aggregiert alle Sub-Report im Ordner `build‣reports‣jacoco` des jeweiligen Root-Projekts. Der Root-Report wird auch für den automatischen Upload nach Codecov verwendet (siehe I.6.1).

Jukito

Für die Unit-Tests wird das Framework Jukito [42] verwendet. Es ermöglicht die Nutzung von Mockito und Guice mit JUnit. Beispiele von Unit-Tests finden sich in den *Referenz-Modules* (I.4.1).

I.6 Build and Deploy

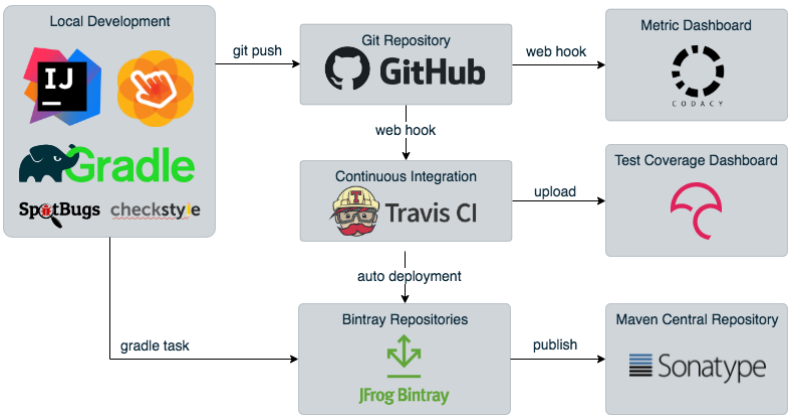


Abbildung I.6: Deployment Übersicht

I.6.1 Travis CI

Beim *pushen* in die Repositories, wird der Quellcode auf Travis CI [68] *gebuildet* und die Unit-Tests sowie Qualitäts-Tests werden ausgeführt. Läuft der Build ohne Fehler durch, wird in einer zweiten Phase die Test-Coverage nach Codecov geladen und die erzeugten JAR-Files nach Bintray deployed (siehe I.6.4) .

I.6.2 Codacy

Der Develop-Branch und bei Bedarf auch weitere Branches werden von Codacy [19] auf die Checkstyle Rules und weitere Metriken untersucht. Codacy gibt einen guten Überblick, wie sich die Code-Qualität im Laufe des Projekts verändert. Codacy kann auch die Test-Coverage anzeigen. Zum Zeitpunkt dieser Arbeit ist es jedoch nicht möglich die Jacoco Test-Coverage auf Codacy hochzuladen, da der Codacy-Uploader nicht mit Java 9 kompatibel ist. Als Alternative wurde deshalb Codecov verwendet (siehe I.6.3).

I.6.3 Codecov

Codecov [20] visualisiert die generierten Jacoco Coverage Files in einem Cloud Dashboard.

I.6.4 Bintray

Im Java Umfeld sind zwei grosse Repositories verbreitet: jCenter von Bintray (JFrog) und MavenCentral von Sonatype. Die Stärken von Bintray ist die höhere Geschwindigkeit des Repository, sowie ein angenehmeres UI. Zudem lassen sich Artefakte bei Bintray relativ gut nach Maven Central deployen.

Sämtliche ADV Projekte werden deshalb mit Travis automatisch nach Bintray deployed. Von dort können die Artefakte weiter nach jCenter und Maven Central publiziert werden, sofern die Anforderungen für das jeweilige Repository eingehalten werden. Bei Maven Central sind die Anforderungen am strengsten. Eine genaue Auflistung der Anforderungen findet sich auf der Sonatype Webseite. [60]

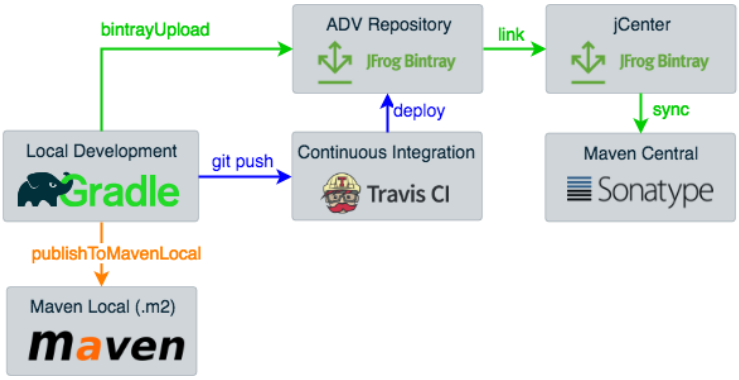


Abbildung I.7: Deployment Setup

Während der Entwicklung wird das Deployment in das lokale Maven Repository sowie in das ADV Bintray Repository empfohlen (Siehe I.6.4: Lokal Entwickeln). Letzteres wird automatisch von Travis durchgeführt. Nach Abschluss eines Moduls sollte im Minimum nach jCenter deployed werden, da dieses Repository sehr einfach in Gradle integriert werden kann (Siehe I.6.4: Major Release / Version Update). Um den Support für Maven zu verbessern kann zudem nach Maven Central deployed werden (Siehe I.6.4: 3. Publizieren nach Maven Central).

Repository Setup

Die benötigten Repositories wurden in allen Projekten im `build.gradle` bereits hinterlegt. Falls das private ADV-Bintray Repository in weiteren Projekten verwendet werden möchte, existiert auf dem Bintray Dashboard [13] eine Anleitung, indem auf «SET ME UP» geklickt wird.

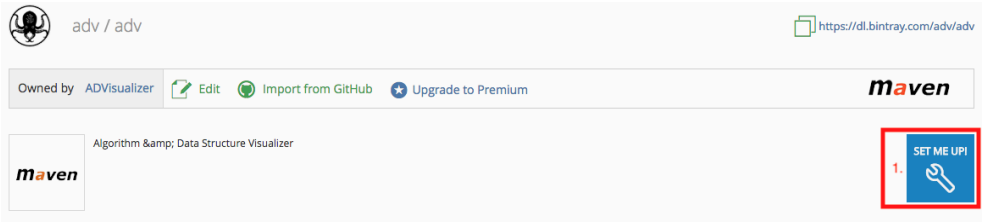


Abbildung I.8: ADV Bintray Repository Setup

Lokal Entwickeln

Bei kleinen Änderungen während der Entwicklung sollte grundsätzlich in das lokale Maven Repository (im Ordner `~/.m2`) deployed werden. Dies geht am einfachsten mit dem Gradle Task `install` oder `publishToMavenLocal`.

Sobald eine Änderung nach Github gepushed wird, publiziert Travis CI das Artefakt automatisch ins private ADV Bintray Repository. Solange die Version eines Artefakts nicht ändert, wird dieses bei jedem Commit überschrieben. Das Bintray Repository könnte deshalb auch als Alternative zu Maven Local verwendet werden. Dies wird wegen der Geschwindigkeit des Deployments jedoch nicht empfohlen.

Hinweis I.6.1: Commit Reihenfolge

Änderungen im ADV Commons Projekt müssen vor den Projekten ADV UI und ADV Lib gepushed und gebuilded werden, da die beiden Projekte vom ADV Commons abhängen.

Major Release / Version Update

Bei einem Major Release wird das Deployment nach jCenter empfohlen, damit die Nutzer des ADV das private ADV Bintray Repository nicht explizit im Projekt definieren müssen. Ein Major Release sollte jeweils am Ende der Modulentwicklung durchgeführt werden.

1. Vorbereitung In einem ersten Schritt muss die Version im *build.gradle* sowie *bintray.json* inkrementiert werden.

Listing I.2: build.gradle

```
group = 'ch.hsr.adv'
version = '0.3'                // increment
```

Listing I.3: bintray.json

```
"version": {
    "name": "0.3",             // increment
    "released": "2018-05-22", // update
    "gpgSign": false
},
"files": [{
    "includePattern": "build/libs/(.*)",
    // increment version of folder
    "uploadPattern": "/ch/hsr/adv/adv-lib/0.3/$1",
    "matrixParams": {
        "override": 1
    }
}],
```

2. Publizieren nach jCenter Nach dem erfolgreichen Deployment können die Artefakte nach jCenter publiziert werden. Diese Prozedur kann einige Stunden dauern, da die Artefakte von Bintray überprüft werden. Sobald die JAR Files bei jCenter angenommen wurden, wird eine Nachricht versendet.

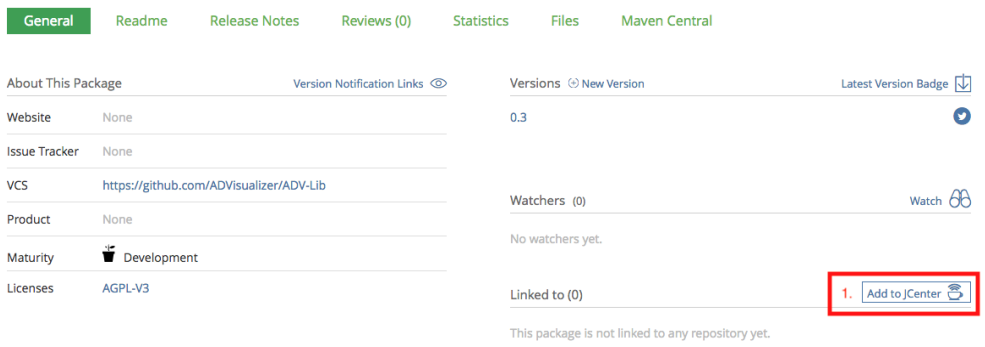


Abbildung I.9: Publizieren nach jCenter

3. Publizieren nach Maven Central Falls zusätzlich nach Maven Central deployed werden möchte, wurde bei Sonatype ein Nexus Repository [59, 58] unter dem *Namespace* `ch.hsr.adv` erstellt. Der Benutzername für das Nexus Repository wurde bereits bei Bintray hinterlegt und zudem dem Betreuer abgegeben. Damit die Artefakte bei Maven Central angenommen werden, müssen sie mit einem GNU Privacy Guard (GPG) Schlüssel signiert werden. Der Schlüssel wurde ebenfalls bei Bintray hinterlegt (siehe I.6.4).

Das Deployment geht am einfachsten mit dem Gradle Task `bintrayUpload`. Für den Upload werden die Anmeldedaten für Bintray, Sonatype (Nexus) sowie die GPG Passphrase benötigt. Diese sind in der Datei `deployment.properties` als Key-Value Felder hinterlegt. Das File ist nicht eingecheckt, wurde aber dem Betreuer übergeben. Die Datei muss im Root-Verzeichnis des Projektes ablegt werden.

Sobald die Artefakte auf Maven Central deployed wurden, sind diese etwa 2 bis 3 Stunden später über die Suche im Maven Central Repository [50] auffindbar.

Achtung I.6.1: API Key einchecken

Das Property File `deployment.properties` darf niemals eingecheckt werden! Es enthält mehrere Passwörter im Klartext.

API Key

Für das Deployment nach Bintray wird ein API-Key benötigt. Der API Key kann dem Bintray Benutzerprofil [13] entnommen werden (siehe Abbildung I.10).

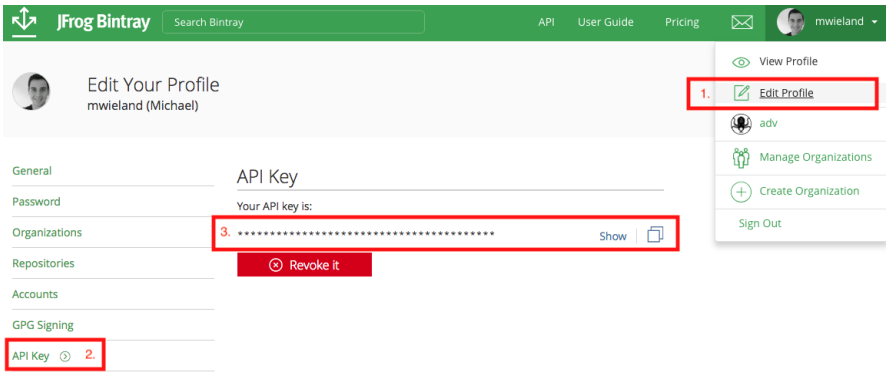


Abbildung I.10: Kopieren des API-Key im Bintray Benutzerprofil

Der API-Key wurde bei Travis CI bereits verschlüsselt hinterlegt. Falls der Schlüssel einmal zurückgezogen werden muss, muss der neue API-Key er-

neut verschlüsselt werden (siehe Codelisting I.4). Die Verschlüsselung muss in jedem Projekt einzeln durchgeführt werden, da das Script projektspezifische Parameter in die Verschlüsselung miteinbezieht. Gleiches gilt für die GPG Passphrase (siehe I.6.4).

Listing I.4: Verschlüsseln des Bintray API Key mit Travis CLI

```
$> travis encrypt <YOUR BINTRAY API KEY> --add deploy.key
```

GPG Key Setup

Der private Schlüssel zum Signieren wurde dem Betreuer übergeben. Der Schlüssel ist für 2 Jahre gültig. Das Expiration Date muss allenfalls aktualisiert werden [29]. Damit die Signatur von Sonatype überprüft werden kann, muss die aktuelle Version des Schlüssels stets auf einem offiziellen Key-Server publiziert sein!

Zudem muss Public und Private Key bei Bintray hinterlegt sein (siehe Abbildung I.11).

Listing I.5: Publizieren des GPG Keys

```
$ gpg2 --keyserver hkp://pool.sks-keyservers.net --send-keys <KEY-ID>
```

Die Artefakte können von Bintray automatisch signiert werden, sofern dieser im Benutzerprofil hinterlegt wurde. Dies wurde für die ADV Organisation bereits gemacht.

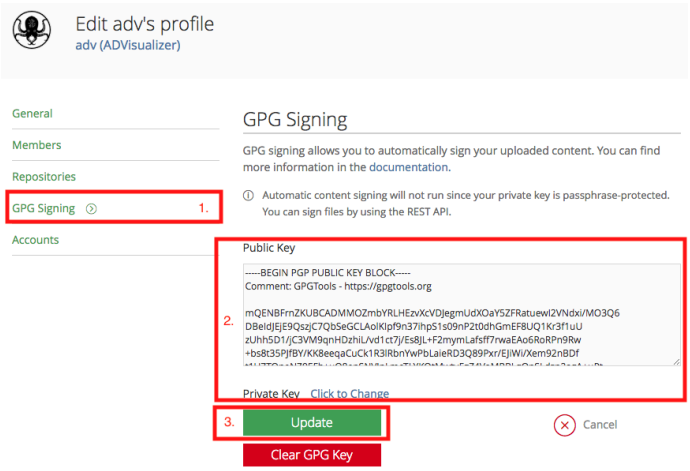


Abbildung I.11: Hinterlegen des GPG Schlüssel bei Bintray

I.7 Modulentwicklung

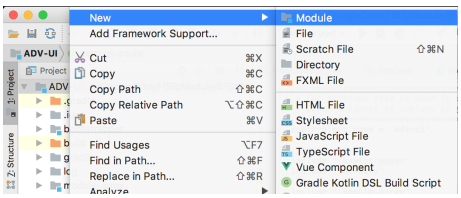
I.7.1 Naming Convention

Bereich	Convention	Beispiel
Module Namen	Als Name des *Modules* versteht sich jeweils der Bezeichner ohne das *"Module"*-Suffix	Der Name des *Test Modules* ist also **Test**
Klassen Namen	Als Präfix den Namen des *Modules* verwenden. Allfällige "ADV" Präfixe dabei ersetzen.	ADVElement Implementierung für das *Test Module*: **TestElement**
Packages	Struktur der core-Projekte übernehmen. Anstelle des `core` Packages soll der Namen des *Modules* verwendet werden	Haupt-UI-Package für das *Test Module*: **ch.hsr.adv.ui.test.***
Module Key	Der einzigartige Schlüssel für ein *Module* entspricht dessen Namen.	Key für das *Test Module*: **test**

Tabelle I.4: Naming Convention

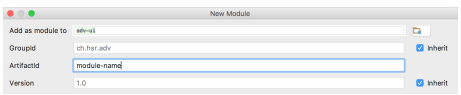
I.7.2 Neues Module erstellen

Der Ablauf für das Erstellen eines neuen *Modules* ist für ADV Lib, ADV UI und ADV Commons identisch. Nachfolgend wird jeweils einer der beiden Abläufe gezeigt.

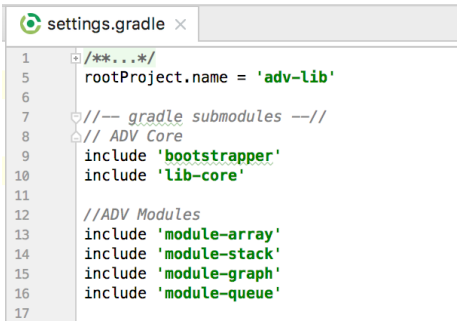


Erstellen eines neuen Modules durch Rechtsklick auf den Root-Ordner.

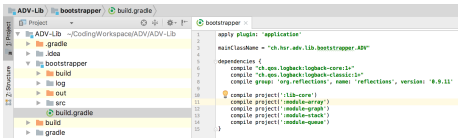
New » Module » Java



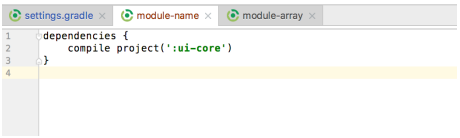
Name des neuen Modules mit dem Präfix "module-" angeben.



Neues *Module* wird mit grundlegender Ordnerstruktur erstellt und automatisch im `settings.gradle` des Root-Projekts registriert.



Das neue *Module* muss manuell im `build.gradle` des **Bootstrapper Gradle Submodule** als Dependency hinterlegt werden.



Im automatisch erstellten `build.gradle` muss mindestens die Abhängigkeit zur Core Komponente definiert werden.

Bei Bedarf können weitere *Module*-spezifische Dependencies mit allfälligen Repositories ins `build.gradle` eingefügt werden (siehe I.5.4).

I.7.3 Module Komponenten

Damit ein neues Modul vom Framework aufgerufen werden kann, müssen folgende Interfaces implementiert werden. Bei der Benennung der neuen Klassen soll die *Naming Convention* eingehalten werden (siehe I.7.1):

name soll mit dem jeweiligen *Module*-Namen (ohne Präfix) ersetzt werden (siehe I.7.1).

ADV Lib

Interface	Ziel-Package	Nutzen
ADVModule	ch.hsr.adv.lib.*name*.logic	Ein Container für die Datenstruktur des *Modules*.
Builder	ch.hsr.adv.lib.*name*.logic	**Service**: Erstellt eine Session aus dem Inhalt des Modules
Stringifyer	ch.hsr.adv.lib.*name*.logic	**[Optional] Service**: Serialisiert eine Session. In den meisten Fällen reicht die Verwendung des *Default-Stringifyer*s.

Tabelle I.5: ADV Lib Interfaces

ADV UI

Interface	Ziel-Package	Nutzen
Layouter	ch.hsr.adv.ui.*name*.presentation	**Service**: Erstellt JavaFX Elemente für die Darstellung der *Module* Elemente
Parser	ch.hsr.adv.ui.*name*.logic	**Service**: Parsed empfangene json-Strings
Stringifyer	ch.hsr.adv.ui.*name*.logic	**Service**: Das Gegenstück des Parsers

Tabelle I.6: ADV UI Interfaces

ADV Commons

Das Ziel-Package beginnt mit: ch.hsr.adv.commons.

Interface	Ziel-Package	Nutzen
ADVElement	*.name*.logic.domain	Entspricht einer Art von Element, welches vom Module dargestellt werden soll. Zum Beispiel einen Knoten eines Graphs.
ADVRelation	*.name*.logic.domain	[Optional] Zeigt Beziehungen zwischen einzelnen Elementen auf. Zum Beispiel Edges in einem Tree.

Tabelle I.7: ADV Commons Interfaces

Neben den oben gelisteten Interfaces macht es Sinn, eine einfache *Constants* Klasse zu erstellen, welche den Namen des Modules als Konstante enthält (vgl. bestehende *Modules*). Diese Konstante kann dann überall verwendet werden, wo das Modul als String registriert werden muss.

I.7.4 Bootstrapping mit Annotationen

Ein wichtiger Aspekt des Frameworks ist die automatische Erkennung der *Module* Services. Dazu wird ein String als Schlüssel verwendet (siehe I.7.3).

Hinweis I.7.1: Module Key

Der gewählte Schlüssel muss in der ADV Lib und im ADV UI exakt gleich lauten! Es bietet sich an, den Schlüssel als statische Konstante im ADV Commons Projekt zu hinterlegen.

Damit die Service Implementationen der *Modules* vom Framework erkannt werden können, müssen diese die Annotation `@Module('name')` nutzen. Services die annotiert werden müssen, sind den Tabellen I.5 und I.6 zu entnehmen.

Das Code Stück I.6 zeigt die Annotation an einer Beispiel-Implementierung des Parsers für das **TestModule**.

Listing I.6: Module Annotation für Framework Services

```
@Module(ConstantsTest.MODULE_NAME)
public class TestParser implements Parser {
    // module-specific parser implementation
}
```

I.8 Framework Umfang

I.8.1 Widgets

Widgets sind eine Sammlung an GUIs Elementen, welche von Modulentwicklern verwendet werden können.

AutoScalePane

Die *AutoScalePane* ist eine Erweiterung der JavaFX Pane. Sie zentriert und zoomt ihren Inhalt automatisch. Wenig Inhalt wird demnach herangezoomt (bis zu einem maximalen Faktor von 2) und viel Inhalt wird herausgezoomt, bis sämtlicher Inhalt auf der Anzeigefläche sichtbar ist.

Listing I.7: AutoScalePane

```
import ch.hsr.adv.ui.core.presentation.widgets.AutoScalePane;

AutoScalePane scalePane = new AutoScalePane();
scalePane.addChildren(container, label);
```

LabeledNode

Die *LabeledNode* Klasse bietet einen Knoten mit einem zentrierten Text. Es kann zwischen einer eckigen und abgerundeten Variante entschieden werden.

Abbildung I.12: LabeledNode

Listing I.8: LabeledNode

```
import ch.hsr.adv.ui.core.presentation.widgets.LabeledNode;

// retrieve style
ADVStyle style = element.getStyle();

//create an angular node
LabeledNode node = new LabeledNode("LabeledNode", style);

//create a rounded node
LabeledNode node = new LabeledNode("LabeledNode", style, true);
```

LabeledEdge

Die *LabeledEdge* Klasse bietet eine Verbindung zwischen zwei *LabeledNodes* mit einem Label, welches mittig an der Verbindung angebracht ist.

Mit sogenannten *ConnectorTypes* kann konfiguriert werden, an welchem Ort die Verbindung angebracht werden soll. `ConnectorType.DIRECT` berechnet dabei den Verbindungspunkt in Bezug zur End-Node in einer direkten Linie.

Eine *Edge* kann zudem eine Richtung besitzen. Diese wird mit einem Pfeilkopf veranschaulicht.

Alle verfügbaren Varianten sind in der Tabelle I.8 gelistet.

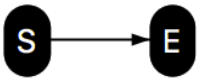
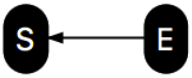
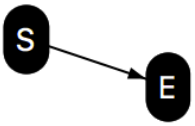
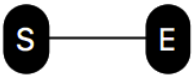
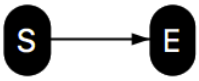
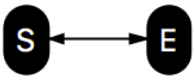
	LabeledEdge
ConnectorType	
TOP	
BOTTOM	
RIGHT	
LEFT	
DIRECT	
LabeledEdge.DirectionType	
NONE	
UNIDIRECTIONAL	
BIDIRECTIONAL	

Tabelle I.8: Verfügbare ConnectorTypes

Listing I.9: LabeledEdge

```
// common ancestor
AutoScalePane scalePane = new AutoScalePane();

// Node Container
VBox vbox = new VBox();
scalePane.addChildren(vbox);

// create nodes
ADVStyle nodeStyle = new ADVDefaultStyle();
LabeledNode start = new LabeledNode("start", nodeStyle);
LabeledNode end = new LabeledNode("end", nodeStyle);
vbox.getChildren().addAll(start, end);

// Create relation
LabeledEdge.DirectionType type = LabeledEdge.DirectionType.UNIDIRECTIONAL;
ADVStyle lineStyle = new ADVDefaultLineStyle();

LabeledEdge relation = new LabeledEdge("start to end",
    start, ConnectorType.DIRECT,
    end, ConnectorType.DIRECT,
    lineStyle, type);

// edge needs to be added to the common ancestor!
scalePane.addChildren(edge);
```

Damit die Berechnung der Start- und End-Koordinaten einer *Edge* korrekt funktionieren sind zwei Dinge **unbedingt** zu beachten:

1. Die *Edge* muss in einem JavaFX Container gezeichnet werden, welcher ein gemeinsamer Parent der Start- und End-Nodes ist (siehe Code Ausschnitt I.9).
2. Auf Grund von JavaFX Problemen mit *WeakListeners* [67], müssen die *Edges* manuell im Layouting Zyklus von JavaFX aktualisiert werden. Dies ist in der AutoScalePane bereits implementiert. Wenn eine andere Pane als Parent verwendet wird, muss diese Mechanik noch ergänzt werden (siehe Code-Ausschnitt I.10)

Listing I.10: Update von Edges in Parent Pane

```
// Verwendung einer Pane als gemeinsames Parent der Nodes und der Edge
    anstelle einer AutoScalePane
Pane parent = new Pane{
    @Override
    protected void layoutChildren() {
        getChildren().forEach(c -> {
            if (c instanceof LabeledEdge) {
                ((LabeledEdge) c).update();
            }
        });
    }}
```

CurvedLabeledEdge

Die *CurvedLabeledEdge* Klasse ist eine Sublasse der *LabeledEdge*, welche eine gekrümmte Verbindung zeichnet.

Die Verwendung sowie verfügbare Features sind identisch zur *LabeledEdge* Klasse.

Abbildung I.13: CurvedLabeledEdge

SelfReferenceEdge

Die *SelfReferenceEdge* Klasse ist eine Sublasse der *CurvedLabeledEdge*, welche eine gekrümmte Verbindung zeichnet. Sie ist speziell ausgelegt für *Edges* welche die gleichen Start- und End-Nodes besitzen.

Die Verwendung sowie verfügbare Features sind identisch zur *LabeledEdge* Klasse.

Abbildung I.14: SelfReferenceEdge

I.8.2 Styles

Styles werden genutzt um *Elemente* und *Relationen* zu stylen. Insbesondere können so gewisse Vorgänge hervorgehoben werden. Zum Beispiel kann ein aktuell aktives Element in einem Suchalgorithmus rot eingerahmt werden. Eine Implementation des *ADVStyle*-Interfaces besteht aus vier Bereichen:

- FillColor: Die Hintergrundfarbe
- StrokeColor: Die Farbe der Umrandung
- StrokeThickness: Die Dicke der Umrandung
- StrokeStyle: der Stil der Umrandung

Gemäss Kommunikationsprotokoll (siehe I.9) werden Farben als hex-Values, Thickness als Pixel-Wert und Style als String übertragen.

Zur angenehmeren Verwendung sind bereits einige vordefinierte Values verfügbar. Die Werte können bei Bedarf aber auch manuell gesetzt werden (siehe ValueStyle I.11).

Colors

STANDARD	BLACK	WHITE
RED_LIGHT	RED	RED_DARK
PURPLE_LIGHT	PURPLE	PURPLE_DARK
BLUE_LIGHT	BLUE	BLUE_DARK
GREEN_LIGHT	GREEN	GREEN_DARK
YELLOW_LIGHT	YELLOW	YELLOW_DARK
ORANGE_LIGHT	ORANGE	ORANGE_DARK
BROWN_LIGHT	BROWN	BROWN_DARK
GRAY_LIGHT	GRAY	GRAY_DARK

Tabelle I.9: Verfügbare ADV Colors

Thickness und Style

	STAN-DARD	THIN	MEDIUM	THICK
SOLID				
DASHED				
DOTTED				

Tabelle I.10: Thickness und Style

I.8.3 Presets

Zur einfacheren Verwendung sind bereits einige fertige Styles verfügbar.

Stylename	Preview
ADVDefaultElementStyle	
ADVSuccessStyle	
ADVInfoStyle	
ADVWarningStyle	
ADVErrorStyle	

Tabelle I.11: Preset Styles

Code Beispiel

Listing I.11: Style Example

```
import ch.hsr.adv.commons.core.domain.styles.*;
import ch.hsr.adv.commons.core.domain.styles.presets.*;

ADVStyle predefinedStyle = new ADVSuccessStyle();
ADVStyle enumStyle = new ADVEnumStyle(ADVColor.ORANGE, ADVStrokeStyle.SOLID,
    ADVStrokeThickness.THIN);
ADVStyle valueStyle = new ADVValueStyle(0xff66ff, 0xff33ff,
    ADVStrokeStyle.SOLID, 4)
```

I.8.4 Events

Events erlauben dem Modulentwickler, in den Arbeitsfluss des Frameworks einzugreifen. Klassen können sich für Events registrieren und werden bei der Auslösung informiert. Jeder Event übergibt den vorherigen sowie zukünftigen Zustand, welcher durch die informierte Klasse verwendet oder gar verändert werden kann.

Die angebotenen Events sind in der Tabelle I.12 gelistet.

ADV Event Handle	Event Trigger
STEP_FIRST	SessionViewModel.navigateSnapshot()
STEP_BACKWARD	SessionViewModel.navigateSnapshot()
STEP_FORWARD	SessionViewModel.navigateSnapshot()
STEP_LAST	SessionViewModel.navigateSnapshot()
CURRENT_SESSION_CHANGED	SessionStore.setCurrentSession()
SESSION_ADDED	SessionStore.addSession()
SESSION_REMOVED	SessionStore.deleteSession()
SNAPSHOT_ADDED	LayoutedSnapshotStore.addLayoutedSnapshot()
NOTIFICATION	Hinweis wird dem Benutzer in der Notification-Bar angezeigt.

Tabelle I.12: Framework Events

Ein Beispiel eines Event-Listener, welche beim Auftreten der registrierten Events notifiziert wird, ist dem Code Listing I.12 zu entnehmen.

Listing I.12: Framework Event in den Modules

```
@Singleton
public class StepEventController implements PropertyChangeListener {
    private static final Logger logger = LoggerFactory.getLogger(
    StepEventController.class);

    // Listen for the following events
    private static final List<ADVEvent> NAVIGATION_EVENTS = List.of(
    ADVEvent.STEP_FIRST, ADVEvent.STEP_BACKWARD,
    ADVEvent.STEP_FORWARD, ADVEvent.STEP_LAST);

    @Inject
    public StepEventController(EventManager eventManager) {
        // EventManager handles the registered events
        eventManager.subscribe(this, NAVIGATION_EVENTS);
    }

    /**
    * Receives the update as the subscribed event happend.
    * @param event occured event
    */
    @Override
    public void propertyChange(PropertyChangeEvent event) {
        logger.info("Received property name: {}", event.getPropertyName());
        logger.info("Received old value: {}", event.getOldValue());
        logger.info("Received new value: {}", event.getNewValue());
    }}
```

I.8.5 I18n Support

Der ADV unterstützt Internationalization (i18n) in den Sprachen Deutsch und Englisch. Alle übersetzbaren Texte sind in Bundle Files ausgelagert. Die Files liegen im Ordner `ui-core ▸ src ▸ main ▸ resources ▸ bundles`

IntelliJ bietet sehr gute Unterstützung für Bundle Files. Es ist daher sehr einfach, einen neuen String für alle vorhanden Sprachen gleichzeitig zu erfassen oder anzupassen (siehe I.15).

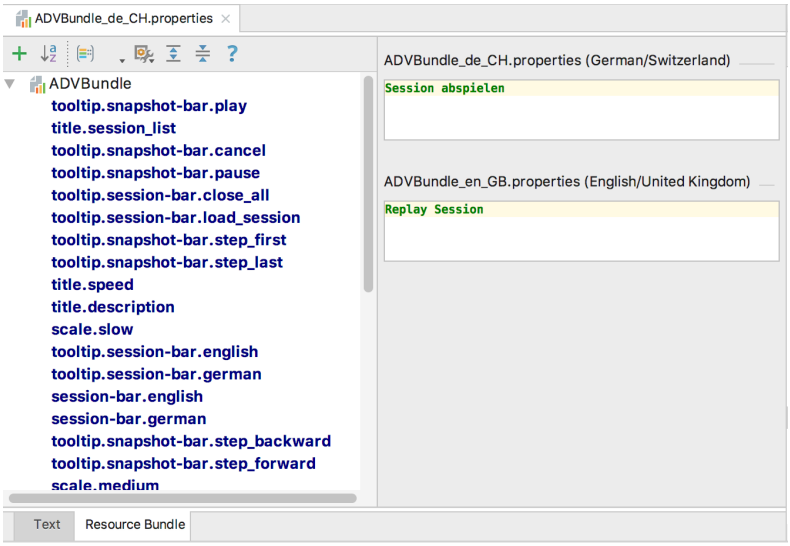


Abbildung I.15: Bundle File in IntelliJ

Hinweis I.8.1: I18n Änderungen nicht sichtbar

Für denn Fall, dass die übersetzten Strings nicht direkt dargestellt werden, müssen die Bundle Files allenfalls mittels `Rechtsklick > Recompile XY.properties` neu kompiliert werden.

Die Util Klasse *ch.hsr.adv.ui.core.presentation.util.I18n* erlaubt es, die Strings im Code zu verwenden und sie zur Laufzeit, je nach aktiver Sprache, zu ersetzen.

Der Code-Ausschnitt I.13 zeigt ein Beispiel zur Verwendung der Hilfsklasse.

Die beiden Methoden zur Erzeugung von String Properties und Tooltips erlauben eine variable Anzahl an Argumenten. Die Argumente können im i18n-Key mit geschweiften Klammern verwendet werden (z.B `tooltip.key.greet = Hallo {0}, mein Name ist {1}`)

Listing I.13: Beispiel für i18n-Übersetzungen im Code

```
@FXML private Label label;
@FXML private Button button;

label.textProperty().bind(I18n.createStringBinding("text.key.label"));
button.setTooltip(I18n.tooltipForKey("tooltip.key.button"));

// with arguments
button.setTooltip(I18n.tooltipForKey("tooltip.key.greet", "Hans", "Paul"));
```

I.9 Kommunikations-Protokoll

Das Framework setzt einen grundlegende Struktur des übertragenen JSON voraus Die Tabelle I.13 gibt eine Übersicht aller Felder, die an den ADV UI Container übertragen werden können.

Feld	Typ	Optional	Beschreibung
`sessionId`	Long		Ein *unique identifier* für die Session. Entspricht der aktuellen Uhrzeit in Millisekunden.
`sessionName`	String		Ein beliebiger Name für die Session
`snapshots`	Array		Umfasst einen oder alle Snapshots der Session
`snapshotId`	Long		Eine inkrementierende Id für den Snapshot. Gibt die Ordnung der Snapshots innerhalb einer Session an.
`snapshotDescription`	String	✓	Eine Erklärung für den Snapshot, welche den ADV User beim Lernen unterstützen soll.
`moduleGroups`	Array		Enthält alle verwendenden *Modules*
`moduleName`	String		Der exakte Name des verwendeten *Modules*
`flags`	Array	✓	Flags können als Strings gesetzt werden, um modul-spezifische Konfigurationen zu übertragen.
`elements`	Array		Umfasst alle Elemente eines Snapshots.

`id`	Long		Ein *unique idendifier* für das Element
`fixedPosX`	Integer	✓	Die x-Koordinate des Elements. Wenn diese Angabe weggelassen wird, sollen die Koordinaten durch den Layouter berechnet werden.
`fixedPosY`	Integer	✓	Die y-Koordinate des Elements. Wenn diese Angabe weggelassen wird, sollen die Koordinaten durch den Layouter berechnet werden.
`content`	Objekt		Der effektive Inhalt des Elements. Modul-spezifisch. Kann weitere `elements` enthalten.
`style`	Object		Das Style-Objekt des Elements
`fillColor`	String		Die Hintergrundfarbe des Elements als Hex-Value
`strokeColor`	String		Die Farbe der Umrandung oder der Linie des Elements als Hex-Value
`strokeStyle`	String		Die Linien-Art der Umrandung oder Linie. Optionen: [dotted | dashed | solid | none]
`strokeThickness`	String		Die Dicke der Umrandung oder der Linie in Pixel
`relations`	Array		Umfasst die Beziehungen zwischen Elementen. Darf Duplikate enthalten.
`sourceElementId`	Long		Id des Start-Elements
`targetElementId`	Long		Id des End-Elements
`label`	String	✓	Eine kurze Beschreibung oder Kennzeichnung für die Beziehung
`isDirected`	Boolean		Gibt an, ob die Relation eine Richtung hat
`style`	Objekt		Aufgebaut gemäss Element-Style

Tabelle I.13: JSON Schema Beschreibung

Listing I.14: Beispiel JSON

```
{
    "sessionId": 1525862380324,
    "sessionName": "Normal Graph Session",
    "snapshots": [
    {
        "snapshotId": 2,
        "snapshotDescription": "Init graph",
        "moduleGroups": [
        {
            "moduleName": "graph",
            "elements": [
            {
                "id": 1,
                "content": "A",
                "fixedPosX": 60,
                "fixedPosY": 50
            },
            {
                "id": 2,
                "content": "B",
                "fixedPosX": 140,
                "fixedPosY": 50
            }
            ],
            "relations": [
            {
                "sourceElementId": 1,
                "targetElementId": 2,
                "isDirected": false
            }
            ],
            "flags": []
        }]
    }]
}
```

I.10 Lessions Learned

In zwei Bereichen gab es bei der Entwicklung des ADV die meisten Hürden zu überwinden: bei anspruchsvolleren GUI Arbeiten mit JavaFX und bei gewissen Testfällen mit Jukito. Die dabei gelernten Tipps und Tricks sind in diesem Kapitel festgehalten.

I.10.1 JavaFX

Koordinaten System

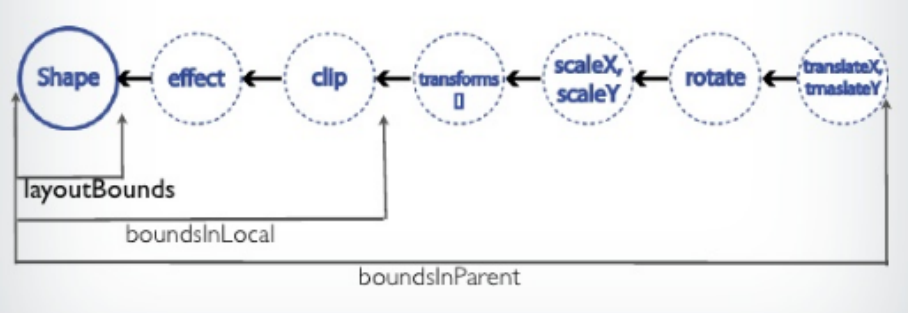
Der Nullpunkt (Ursprung) des verwendeten Koordinatensystems befindet sich oben links. Wichtig zu wissen ist, dass jede Node ihr eige-

nes Bezugssystem hat. Dies bedeutet, dass zum Beispiel eine HBox an ihrer oberen linken Ecke einen eigenen (0,0) Punkt besitzt. Dies wird insbesondere dann wichtig, wenn Elemente in unterschiedlichen Containern miteinander in Bezug gesetzt werden. Beispielsweise, wenn von einer Node in der einen HBox einen Pfeil zu einer Node in einer anderen HBox gezogen wird (vgl. Objektreferenzen in *Array Module*). Da in diesem Fall beide Nodes ihre Koordinaten im Bezugssystem ihrer HBoxen haben, müssen die Koordinaten für den Pfeil umgerechnet werden:

```
private Bounds getRelativeBounds(Node node, Node relativeNode) {
    Bounds nodeBoundsInScene =
        node.localToScene(node.getBoundsInLocal());
    return relativeNode.sceneToLocal(nodeBoundsInScene);
}
```

Bounds

In JavaFX gibt es drei Arten von Bounds: `layoutBounds(); boundsInLocal(); boundsInParent();`. Diese unterscheiden sich im Bezug auf das verwendete Koordinatensystem. `boundsInParent()` gibt als `minX()` die X-Koordinate im Bezugssystem des Parents, wohingegen `boundsInLocal().minX()` die Koordinaten im eigenen Bezugssystem zurück gibt. `layoutBounds()` entspricht den logischen Bounds der Node. Oftmals, aber nicht immer, sind diese identisch zu den `boundsInLocal()`.



Layouting Process

Für kompliziertere Positionierungen von Elementen sind die Bounds und die Koordinatensysteme extrem relevant. Neben dem grundlegenden Verständnis, kommt in der Praxis ein weiterer Aspekt hinzu: Timing. Insbesondere die Bounds werden erst vollständig berechnet, wenn ein Node in einer Scene bzw. im Scene-Graph eingebettet wurde. Dies führt dazu, dass oftmals die benötigten Werte (wie zum Beispiel die Breite einer Node) nicht vorhanden oder falsch sind. Abhilfe schaft oftmals ein Listener auf `boundsInParent()`. Sollte dies nicht klappen, hat es sich als nützlich erwiesen, solche Berechnungen in einer überschriebenen `layoutChildren()` Methode zu tätigen.
(siehe ch.hsr.adv.ui.core.presentation.widgets.AutoScalePane)

I.10.2 Jukito

JukitoModule und GuiceModule

Leider ist der Begriff *Module* im ADV Projekt sehr überladen. Neben ADV Modules und Gralde Modules gibt es Guice Modules und Jukito Modules. Ein Jukito Module ist ein Wrapper um ein Guice Module. So können in beiden Modulen zum Beispiel ein Interfaces auf konkrete Klassen gebunden oder Guice Factories installiert werden. Ein Jukito Module kann direkt in der Test Klasse als private statische innere Klasse erstellt werden (1). Alternativ kann auch ein Guice Module für den Test-Run verwendet werden (2). Sobald ein Guice Module verwendet wird, wird das Jukito Module leider ignoriert!

Mocken von Klassen

Interfaces werden von Jukito automatisch gemockt. Es macht keinen Sinn nur den Tests zu Liebe für jede Klasse ein Interface einzuführen. Umso besser also, dass Jukito auch Klassen mocken kann (3)

Injections

Field-Injections (4) werden manchmal nicht so injected wie dies erwartet werden könnte. Insbesondere wenn verifiziert werden muss, ob eine Instanzvariable des "System under Test" verwendet wurde, sollte die entsprechende Klasse der Variable direkt in die Testmethode injected werden. (5)

```
@RunWith(JukitoRunner.class)
@UseModules({GuiceCoreModule.class}) // 2
public class ArrayLayouterTest extends ApplicationTest {
    @Inject //4
    private ArrayParser testParser;
    @Inject
    private ArrayLayouter sut;

    @Test                                              // 5
    public void layoutDefaultTest(ArrayDefaultLayouter mockDefaultLayouter) {
        // WHEN
        sut.layout(moduleGroup, null);

        // THEN
        verify(mockDefaultLayouter).layout(moduleGroup);
    }
    // 1
    public static class Module extends JukitoModule {
        @Override
        protected void configureTest() {
            forceMock(ArrayDefaultLayouter.class); // 3
        }
    } }
```

Hinweis I.10.1: Test Beispiele

Viele weitere Beispiele für Tests finden sich in den ADV Lib und ADV UI Projekten. Zum Beispiel unter: `ui-core‣src‣test`

Anhang J

Zeitauswertung

J.1 Zeitauswertung nach Kategorien

Testing
Vorbereitung und Durchführung von Usability- und Systemtests

Analyse
Analyse von Requirements

Design
Erarbeiten von Architektur und Design

Dokumentation
Schreiben und Überarbeiten der Dokumentation

Administratives
Sitzungen, Schreiben von Protokollen

Implementierung
Schreiben von produktivem Code und Unit-Tests, Code Reviews, Aufsetzen des Toolings (Travis, Bintray, Codacy etc.)

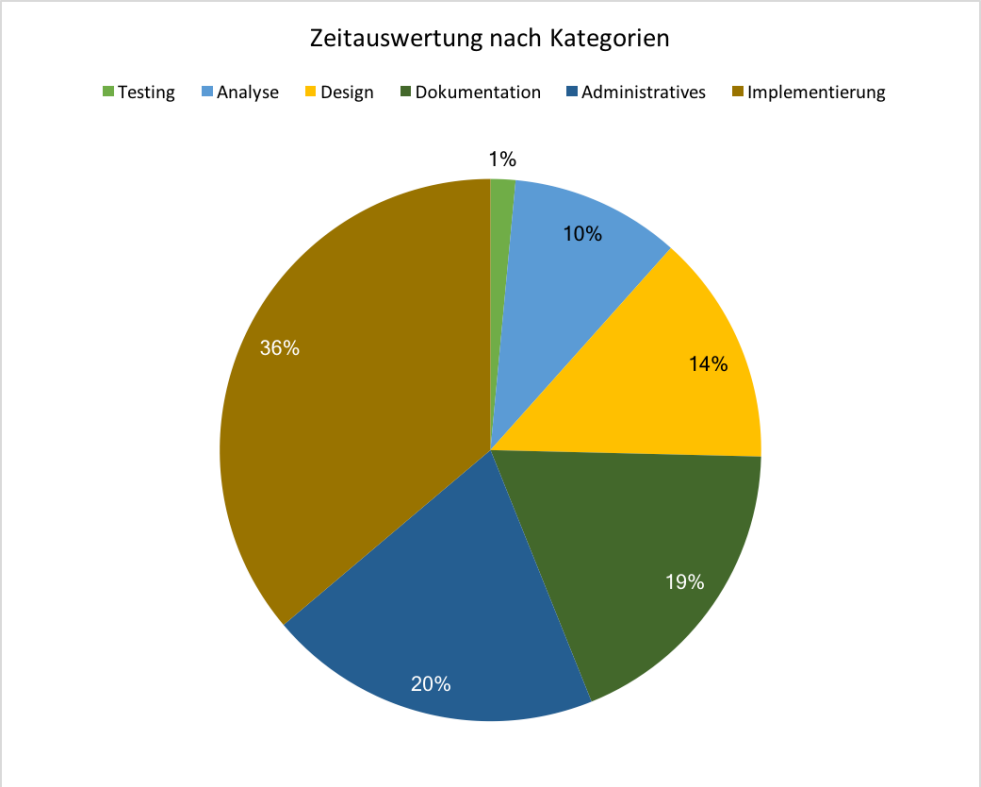


Abbildung J.1: Zeitauswertung nach Kategorien

J.2 Zeitauswertung nach Meilensteinen

Auswertung gemäss Meilensteine (siehe 4.3).

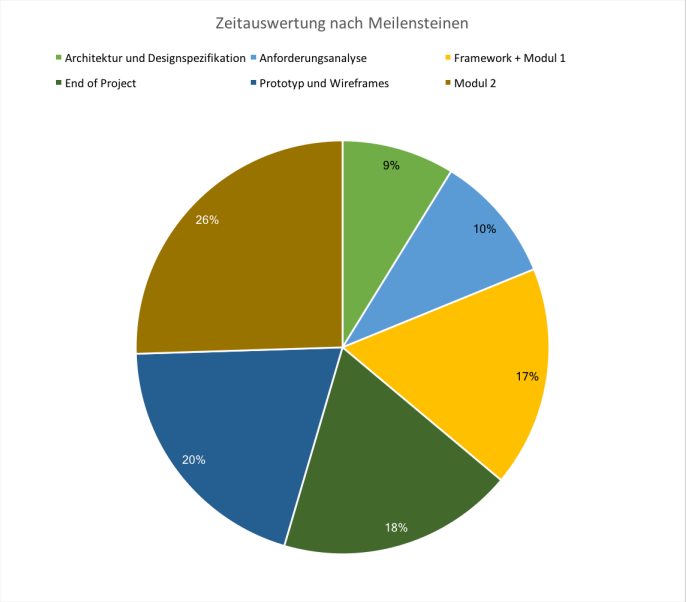


Abbildung J.2: Zeitauswertung nach Phasen

J.3 Zeitauswertung nach Teammitgliedern

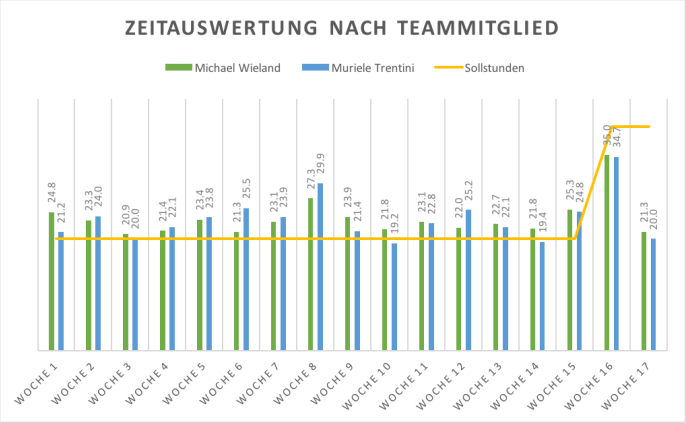


Abbildung J.3: Zeitauswertung nach Teammitgliedern

J.4 Zeitauswertung Soll-Ist-Vergleich

Insgesamt wurde für die vorliegende Arbeit 802 Stunden aufgewendet. Das Ziel-SOLL für die Arbeit lag bei 760 Stunden (15 Wochen * 20 Stunden + 2 Wochen * 40 Stunden pro Teammitglied).

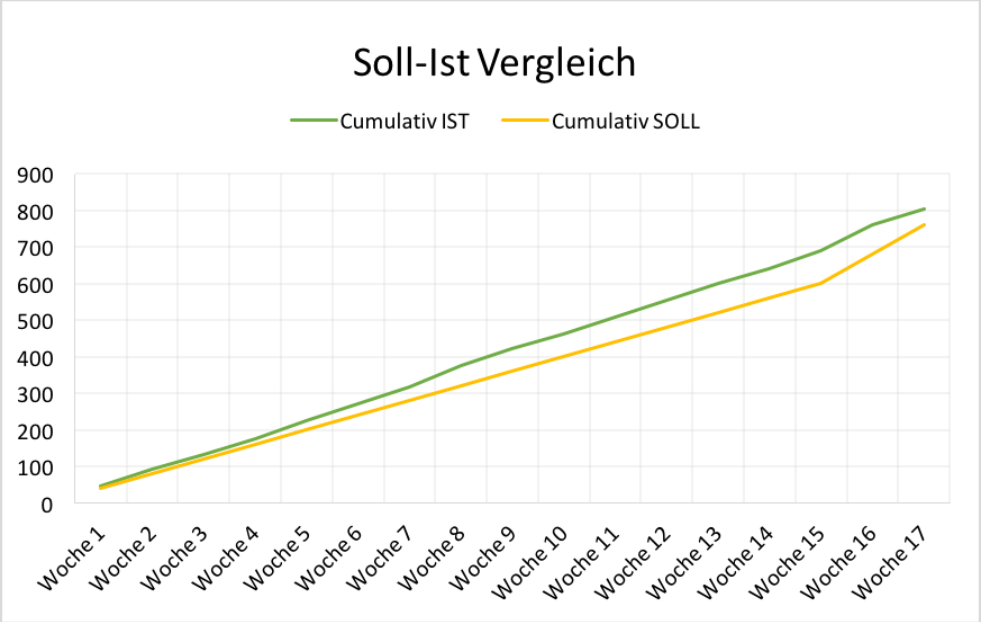


Abbildung J.4: Zeitauswertung Soll-Ist-Vergleich

Literaturverzeichnis

[1] Dirk Riehle. «Framework Design: A Role Modeling Approach». Ph.D. Thesis. ETH Zürich, 2000. URL: http://dirkriehle.com/computer-science/research/dissertation/diss-a4.pdf (besucht am 05. 06. 2018).

[2] Andreas Egli und Michael Koller. «Graphs-Visualization-Service». Diplomarbeit. Hochschule für Technik Rapperswil, 15. Dez. 2005.

[3] Christoph Buchheim, Michael Jüunger und Sebastian Leipert. «Drawing rooted trees in linear time». In: *Wiley InterScience* (2006). URL: https://onlinelibrary.wiley.com/doi/pdf/10.1002/spe.713 (besucht am 05. 06. 2018).

[4] J.D. Meier u. a. *Microsoft Application Architecture Guide. Chapter 5: Layered Application Guidelines, 2nd Edition*. Microsoft. 12. Juni 2013. URL: https://docs.microsoft.com/en-us/previous-versions/msp-n-p/ee658109%28v%3dpandp.10%29 (besucht am 03. 06. 2018).

[5] J.D. Meier u. a. *Microsoft Application Architecture Guide. Chapter 8: Data Layer Guidelines, 2nd Edition*. Microsoft. 12. Juni 2013. URL: https://docs.microsoft.com/en-us/previous-versions/msp-n-p/ee658127%28v%3dpandp.10%29 (besucht am 03. 06. 2018).

[6] Anton Epple. *JavaFX 8. Grundlagen und fortgeschrittene Techniken*. dpunkt.verlag, 2015, S. 19–35.

[7] Gernot Starke. *Effektive Software-Architekturen. Ein praktischer Leitfaden*. Carl Hanser Verlag, 2015, S. 117.

[8] *MVVM. Xamarin*. 8. Juli 2017. URL: https://docs.microsoft.com/en-us/xamarin/xamarin-forms/enterprise-application-patterns/mvvm (besucht am 03. 06. 2018).

[9] Muriele Trentini und Michael Wieland. «Graphs-Visualization-Service GVS 2.0». Studienarbeit. Hochschule für Technik Rapperswil, Abteilung Informatik, 21. Dez. 2017.

[10] Muriele Trentini und Michael Wieland. «Framework zur Visualisierung von Algorithmen und Datenstrukturen». Bachelorarbeit. Hochschule für Technik Rapperswil, Abteilung Informatik, 15. Juni 2018.

[11] *Algorithm Visualizer*. URL: http://algo-visualizer.jasonpark.me (besucht am 05. 06. 2018).

[12] *Atlassian Marketplace: Tempo Timesheets for Jira*. URL: https://marketplace.atlassian.com/plugins/is.origo.jira.tempo-plugin/cloud/overview (besucht am 05. 06. 2018).

[13] *Bintray*. URL: https://bintray.com/adv/adv (besucht am 22. 05. 2018).

[14] *C4 Model*. URL: https://c4model.com/ (besucht am 05. 06. 2018).

[15] *Checkstyle*. URL: http://checkstyle.sourceforge.net/ (besucht am 05. 06. 2018).

[16] *Checkstyle Idea*. URL: https://plugins.jetbrains.com/plugin/1065-checkstyle-idea (besucht am 04. 06. 2018).

[17] *Checkstyle Sun*. URL: https://github.com/checkstyle/checkstyle/blob/master/src/main/resources/sun_checks.xml (besucht am 05. 06. 2018).

[18] *Class Application. Threading*. URL: https://docs.oracle.com/javase/8/javafx/api/javafx/application/Application.html (besucht am 04. 06. 2018).

[19] *Codacy Dashboard*. URL: https://app.codacy.com/app/ADV/ADV-UI/dashboard (besucht am 05. 06. 2018).

[20] *Codecov*. URL: https://codecov.io/gh/ADVisualizer (besucht am 31. 05. 2018).

[21] *ControlsFX*. URL: http://fxexperience.com/controlsfx/ (besucht am 05. 06. 2018).

[22] *Dropbox*. URL: https://www.dropbox.com (besucht am 05. 06. 2018).

[23] *Eclipse Buildship*. URL: http://projects.eclipse.org/projects/tools.buildship (besucht am 04. 06. 2018).

[24] *FontAwesome*. URL: https://fontawesome.com/ (besucht am 03. 06. 2018).

[25] Martin Fowler. *Inversion Of Control*. URL: https://martinfowler.com/bliki/InversionOfControl.html (besucht am 05. 06. 2018).

[26] *FXGL: avaFX Game Development Framework*. URL: https://almasb.github.io/FXGL (besucht am 05. 06. 2018).

[27] *Git Repository des ADV*. URL: https://github.com/ADVisualizer (besucht am 03. 06. 2018).

[28] *Gitflow*. URL: http://nvie.com/posts/a-successful-git-branching-model/ (besucht am 24. 09. 2017).

[29] *GPG Extend Expiration Date*. URL: http://central.sonatype.org/pages/working-with-pgp-signatures.html (besucht am 19. 05. 2018).

[30] *Gradle*. URL: https://gradle.org/ (besucht am 05. 06. 2018).

[31] *Gradle Multi-project Builds*. URL: https://guides.gradle.org/creating-multi-project-builds/ (besucht am 05. 06. 2018).

[32] *GraphStream*. URL: http://graphstream-project.org (besucht am 05. 06. 2018).

[33] *Gson*. URL: https://github.com/google/gson (besucht am 05. 06. 2018).

[34] *Guice*. URL: https://github.com/google/guice (besucht am 05. 06. 2018).

[35] *Guice MapBinder*. URL: https://google.github.io/guice/api-docs/latest/javadoc/index.html?com/google/inject/multibindings/MapBinder.html (besucht am 05. 06. 2018).

[36] *Hamcrest*. URL: http://hamcrest.org/JavaHamcrest/ (besucht am 04. 06. 2018).

[37] *IntelliJ IDEA*. URL: https://www.jetbrains.com/idea/ (besucht am 05. 06. 2018).

[38] *IPC Dimensions*. URL: https://www.nginx.com/blog/building-microservices-inter-process-communication/ (besucht am 05. 06. 2018).

[39] *JaCoCo Java Code Coverage Library*. URL: http://www.eclemma.org/jacoco/ (besucht am 05. 06. 2018).

[40] *JConsole*. URL: https://docs.oracle.com/javase/7/docs/technotes/guides/management/jconsole.html (besucht am 05. 06. 2018).

[41] *Jira*. URL: https://project.redbackup.org/projects/GVS/ (besucht am 05. 06. 2018).

[42] *Jukito*. URL: https://github.com/ArcBees/Jukito (besucht am 05. 06. 2018).

[43] *JUNG: Java Universal Network/Graph Framework*. URL: http://jung.sourceforge.net (besucht am 05. 06. 2018).

[44] *Junit*. URL: https://junit.org/junit4/ (besucht am 05. 06. 2018).

[45] *Kraken*. URL: https://de.wikipedia.org/wiki/Kraken?section=8#Lernf%C3%A4higkeit (besucht am 05. 06. 2018).

[46] *Layered Architecture*. URL: https://www.safaribooksonline.com/library/view/software-architecture-patterns/9781491971437/ch01.html (besucht am 04. 06. 2018).

[47] *Log4J*. URL: https://logging.apache.org/log4j/2.x/ (besucht am 05. 06. 2018).

[48] *Logback*. URL: https://logback.qos.ch/ (besucht am 05. 06. 2018).

[49] Yves Lucet. *Data Structure Visualizations*. URL: https://people.ok.ubc.ca/ylucet/DS/Algorithms.html (besucht am 05. 06. 2018).

[50] *Maven Central Repository*. URL: https://search.maven.org/#search%7Cga%7C1%7Cch.hsr.adv (besucht am 31. 05. 2018).

[51] *Mockito*. URL: http://site.mockito.org/ (besucht am 05. 06. 2018).

[52] *Monocle*. URL: https://github.com/TestFX/Monocle (besucht am 04. 06. 2018).

[53] *OpenJDK JEP 253*. URL: http://openjdk.java.net/jeps/253 (besucht am 04. 06. 2018).

[54] *OpenJDK Project Jigsaw*. URL: http://openjdk.java.net/projects/jigsaw/ (besucht am 04. 06. 2018).

[55] *Reflections*. URL: https://github.com/ronmamo/reflections (besucht am 05. 06. 2018).

[56] *Risikomanagement*. URL: https://risikomanager.org/methodenassistent/risikodiagramm-risikograph-risikolandschaft-risikoportfolio-risikomatrix/ (besucht am 05. 06. 2018).

[57] *SLF4J*. URL: https://www.slf4j.org/manual.html (besucht am 05. 06. 2018).

[58] *Sonartype Jira*. URL: https://issues.sonatype.org (besucht am 19. 05. 2018).

[59] *Sonartype Nexus Repository*. URL: https://www.sonatype.com/nexus-repository-oss (besucht am 19. 05. 2018).

[60] *Sonatype Requirements*. URL: http://central.sonatype.org/pages/requirements.html (besucht am 19. 05. 2018).

[61] *Spotbugs*. URL: https://spotbugs.github.io/ (besucht am 05. 06. 2018).

[62] *Story Points verständlich erklärt*. URL: http://www.ksimons.de/2011/06/story-points-verstandlich-erklart/ (besucht am 05. 06. 2018).

[63] *Strategy Pattern*. URL: https://sourcemaking.com/design_patterns/strategy (besucht am 05. 06. 2018).

[64] *Structure 101*. URL: https://structure101.com/ (besucht am 05. 06. 2018).

[65] *TestFX*. URL: https://github.com/TestFX/TestFX (besucht am 05. 06. 2018).

[66] *The secrets behind story points and agile estimation*. URL: https://www.atlassian.com/agile/estimation (besucht am 05. 06. 2018).

[67] *The Trouble with Weak Listeners*. URL: https://tomasmikula.github.io/blog/2015/02/10/the-trouble-with-weak-listeners.html (besucht am 16. 05. 2018).

[68] *Travis CI*. URL: https://travis-ci.org/ADVisualizer/ (besucht am 05. 06. 2018).

[69] *VisuAlgo*. URL: https://visualgo.net (besucht am 05. 06. 2018).

[70] Prof. Dr. Olaf Zimmermann. *Y-Statements*. URL: https://resources.sei.cmu.edu/library/asset-view.cfm?assetID=31345 (besucht am 05. 06. 2018).