Bachelorarbeit
zur Erlangung des akademischen Grades
Bachelor of Science

Modellierung einer Software-Architektur mithilfe zweidimensionaler Strichgesten

Modelling Software Architecture Using Two-Dimensional Stroke Gestures

Nico Weiser
Matrikelnummer: 4221740

2. November 2021

Erstgutachter: Dr. Prof. Rainer Koschke
Zweitgutachter: Sabine Kuske

E R K L Ä R U N G

Ich versichere, den Bachelor-Report oder den von mir zu verantwortenden Teil einer Gruppenarbeit*) ohne fremde Hilfe angefertigt zu haben. Ich habe keine anderen als die angegebenen Quellen und Hilfsmittel benutzt. Alle Stellen, die wörtlich oder sinngemäß aus Veröffentlichungen entnommen sind, sind als solche kenntlich gemacht.

*) Bei einer Gruppenarbeit muss die individuelle Leistung deutlich abgrenzbar und bewertbar sein und den Anforderungen entsprechen.

Bremen, den _____

_____
(Unterschrift)

Aus Gründen der besseren Lesbarkeit wird im Text verallgemeinernd das generische Maskulinum verwendet. Diese Formulierungen umfassen gleichermaßen weibliche und männliche Personen; alle sind damit selbstverständlich gleichberechtigt angesprochen.

Inhaltsverzeichnis

1 Einleitung **8**
- 1.1 Ziele der Arbeit . . . 8
- 1.2 Aufbau der Arbeit . . . 9

2 Grundlagen **10**
- 2.1 Software-Architektur . . . 10
- 2.2 Unity . . . 11
- 2.3 Graph eXchange Language . . . 12
- 2.4 JSON . . . 13
- 2.5 Projekt SEE . . . 14
- 2.6 Wacom Cintiq 16 . . . 14
- 2.7 Axivion Gravis . . . 15
- 2.8 Ähnliche Arbeiten . . . 17
  - 2.8.1 A collaborative Multi-Touch UML Design Tool . . . 17
  - 2.8.2 VR-UML: The unified Modeling Language in Virtual Reality . 17
  - 2.8.3 Modellierung von Softwarearchitektur in Virtual Reality mithilfe der Leap Motion . . . 18

3 Entwurf **22**
- 3.1 Entwurfsidee . . . 22
  - 3.1.1 Gestenerkennung . . . 22
  - 3.1.2 Darstellung der Architektur . . . 24
  - 3.1.3 Erzeugen von Architekturkomponenten . . . 24
  - 3.1.4 Bearbeiten von Architekturkomponenten . . . 24
  - 3.1.5 Laden & Speichern der Software-Architektur . . . 24

4 Implementierung **25**
- 4.1 Unity Setup . . . 25
  - 4.1.1 Darstellungsoptionen . . . 25
- 4.2 Gestenerkennung . . . 27
  - 4.2.1 Stifteingabe . . . 27
  - 4.2.2 Strichgesten . . . 28
  - 4.2.3 Algorithmus . . . 29
  - 4.2.4 Umsetzung . . . 31
  - 4.2.5 Sampling . . . 32
- 4.3 Architekturmodus . . . 32

4.3.1 UI . . . 33
4.3.2 Darstellung der Architektur . . . 33
4.3.3 Erzeugen von Architekturkomponenten . . . 35
4.3.4 Bearbeiten von Architekturkomponenten . . . 35
4.3.5 Laden & Speichern der Softwarearchitektur . . . 36
4.3.6 Zusätzliche Funktionen . . . 36
4.4 Probleme . . . 36

5 Evaluation **38**
5.1 Planung . . . 38
5.1.1 DECIDE-Framework . . . 38
5.2 Durchführung . . . 42
5.2.1 Aufbau . . . 42
5.2.2 Ablauf . . . 42
5.3 Ergebnisse . . . 43
5.3.1 Teilnehmer . . . 43
5.3.2 Quantitative Daten . . . 43
5.3.3 Qualitatives Feedback . . . 46
5.4 Auswertung . . . 47
5.4.1 Hypothesen . . . 47
5.4.2 Quantitative Daten . . . 48
5.4.3 Qualitatives Feedback . . . 50
5.4.4 Ergebnis der Evaluation . . . 51

6 Fazit und Ausblick **52**
6.1 Fazit . . . 52
6.2 Ausblick . . . 52
6.2.1 Radial-Menü . . . 52
6.2.2 Onscreen Tastatur . . . 53

Quell und Literaturverzeichnis **54**

Anhang **57**

Abbildungsverzeichnis

2.1 Unity Editor 11
2.2 GameObject mit Komponenten 12
2.3 SEE 15
2.4 Wacom Cintiq 16 Stift-Display[10] 16
2.5 Wacom Pro Pen slim Stift[10] 16
2.6 Axivion Gravis 17
2.7 Kollaboratives Multi-Touch UML Design Tool[14] 18
2.8 Hyperplanes unterschiedlicher Diagrammtypen[18] 19
2.9 Virtuelles Tablet[18] 19
2.10 LeftToRightIndex Handgeste [20] 20
2.11 Visualisierung einer alten Teil-Architektur von Twitter [20] 21

3.1 Mockup in Microsoft Whiteboard App 23
3.2 UI Mockup in Microsoft Whiteboard App 23

4.1 Unity Szene 26
4.2 SEECityArchitecture Settings 27
4.3 Architecture Input Actions 28
4.4 UI im Architekturmodus 33
4.5 Bedeuetung der UI Elemente 34
4.6 Label der Architekturkomponenten 34

5.1 Bewertungsskala nach Bangor et al.[30] 40
5.2 Digitale Einwilligungserklärung der Evaluation 41
5.3 Einführung in die Thematik 44
5.4 Vergleich der Antworten des SUS-Fragebogen 45
5.5 Vergleich der Bearbeitungszeit je Tool 46

6.1 Wacom Expressmenu 53

Tabellenverzeichnis

5.1 Bereinigte Messergebnisse 45
5.2 Effizienz: Shapiro-Wilk-Test auf Normalverteilung 48
5.3 Effizienz: Wilcoxon-Test 48
5.4 Effektivität: Shapiro-Wilk-Test auf Normalverteilung 49
5.5 Effektivität: Wilcoxon-Test 49
5.6 Sicherheit: Shapiro-Wilk-Test auf Normalverteilung 49
5.7 Sicherheit: Wilcoxon-Test 49
5.8 SUS-Score: gepaarter t-Test 50

Listings

4.1 Input Action Logik im Update() 27
4.2 Event-driven Input Action Logik 28
4.3 Extrahieren der Strichgestenpunkte 28
4.4 Serialisierte Gesten 32

Kapitel 1

Einleitung

Der Entwurf einer Software-Architektur ist ein kritischer Schritt für die erfolgreiche Implementierung eines komplexen Software-Systems. Die Architektur bildet die Brücke zwischen den Anforderungen und der Implementation. Sie abstrahiert die komplexen Anforderungen als eine leicht verständliche Darstellung von Strukturen. Indem sie Zugehörigkeiten und Abhängigkeiten festlegt, dient sie gewissermaßen als Bauplan für das Software-System, an dem sich die Entwickler orientieren können.

Doch wie modelliert man eine solche Software-Architektur? In der Regel verwendet man eine Modellierungssprache wie die Unified Modeling Language (UML)[1] für die Erstellung von Diagrammen. Dafür werden meist spezielle UML-Editoren wie UMLet[2] oder ArgoUML[3] genutzt, deren Verwendung sich grundsätzlich immer ähnelt: Die Nutzer platzieren UML-Elemente mithilfe von Maus & Tastatur auf einer Zeichenfläche, um ihnen so eine Bedeutung zu geben.

Als Software-Entwickler nutzt man im Alltag ebenfalls gerne Abstraktionen, um schnell komplexe Zusammenhänge verständlich darzustellen. Ein gern genutztes Medium sind Whiteboards, an denen dann mit primitiven Formen und Pfeilen Zusammenhänge veranschaulicht werden. Was wäre, wenn man mithilfe eines Stift-Displays diese natürliche und intuitive Art der Modellierung als digitales Tool abbildet?

1.1 Ziele der Arbeit

Diese Forschungsarbeit befasst sich mit dem Ziel, das SEE Projekt um eine Lösung zur Modellierung von Software-Achitektur zu erweitern, die es dem Benutzer erlaubt mithilfe von Strichgesten Architekturkomponenten und Strukturen zu erzeugen. Die so modellierten Architekturen sollen in Zukunft für eine Architekturprüfung mittels *Incremental Reflexion Analysis* genutzt werden können, welche bestehende Implementierungen auf Verletzungen der Architekturvorgaben überprüft. [4]. Als Eingabegerät soll das Stift-Display von Wacom genuztzt werden, welches in Kapitel 2.6 genauer vorgestellt wird. Die Strichgesten in

Gestalt primitiver Formen. wie Vierecken oder Kreise, sollen zum Erzeugen verschiedener Typen von Architekturkomponenten genutzt werden können. Abhängigkeiten können durch simple Linien zwischen Strukturen und Komponenten modelliert werden. Der Benutzer soll so in der Lage sein die Architektur, ähnlich wie an einem Whiteboard, skizzieren zu können. Anhand dieser Überlegungen lassen sich folgende Aufgaben formulieren, die zum Erreichen des Zieles notwendig sind:

1. Eine Strichgestenerkennung implementieren
2. Eine Steuerung entwickeln, damit der Benutzer Strichgesten auf dem Stift-Display zeichnen kann
3. Eine Funktion entwickeln mit derer auf Basis der Strichgesten Architekturkomponenten erzeugt werden
4. Eine Steuerung entwickeln, damit der Benutzer Komponenten bearbeiten kann
5. Eine Funktion entwickeln, damit der Benutzer die erzeugte Softwarearchitektur abspeichern kann
6. Eine Funktion entwickeln, damit der Benutzer bereits bestehende Architekturen laden und bearbeiten kann.

1.2 Aufbau der Arbeit

In Kapitel 2 werden die für die Forschungsarbeit benötigten Grundkenntnisse vermittelt. Kapitel 3 erläutert den Entwurf und die getroffenen Design-Entscheidungen, die zur Umsetzung der Funktionen benötigt werden. Im nächsten Kapitel 4 werden die Details der technischen Umsetzung erläutert. In Kapitel 5 wird die entwickelte Erweiterung für das SEE-Projekt hinsichtlich Effizienz, Sicherheit und Effektivität analysiert. Das Kapitel 6 schließt die Forschungsarbeit mit dem Fazit ab und gibt einen Ausblick auf die weitere Zukunft.

Kapitel 2

Grundlagen

2.1 Software-Architektur

Eine Software-Architektur ist ein wichtiges Werkzeug für die Planung und Umsetzung eines Software-Systems. Sie bildet die Brücke zwischen den komplexen Anforderungen des Systems und der konkreten Implementierung. Nach Bass et al.[5] ist eine Software-Architektur wie folgt definiert:

> „The software architecture of a system is the set of structures needed to reason about the system. These structures comprise software elements, relations among them, and properties of both.“

Die Architektur abstrahiert das System mithilfe einer Vielzahl an Strukturen bestehend aus Elementen und deren Abhängigkeiten. Diese Abstraktion vernachlässigt nicht relevante Informationen, wie Details der Implementierung, um die Komplexität der Darstellung zu verringern. Bass et al. teilen die Strukturen in drei Kategorien ein, mit denen sich jeweils spezifische Fragen über das System beantworten lassen:

Die **Component-and-connector(C&C) structures** beschreiben das Laufzeitverhalten von Elementen und deren Interaktionen untereinander. Innerhalb der *C&C structures* wird zwischen *Components* und *Connectors* unterschieden. *Components* repräsentieren Recheneinheiten wie zum Beispiel *Services* oder *Clients* die über die sogenannten *Connectors* miteinander interagieren. Ein *Connector* stellt die Kommunikationsart der Interaktion dar. Beispiel für eine Frage:

> „How does data progress through the system?“

Die **Allocation structures** beschreiben die Verbindung zu Nicht-Software-Strukturen, wie zum Beispiel dem Unternehmen oder der Laufzeitumgebung. Beispiel für eine Frage:

> „In which directories or files is each element stored during development, testing, and system building?“

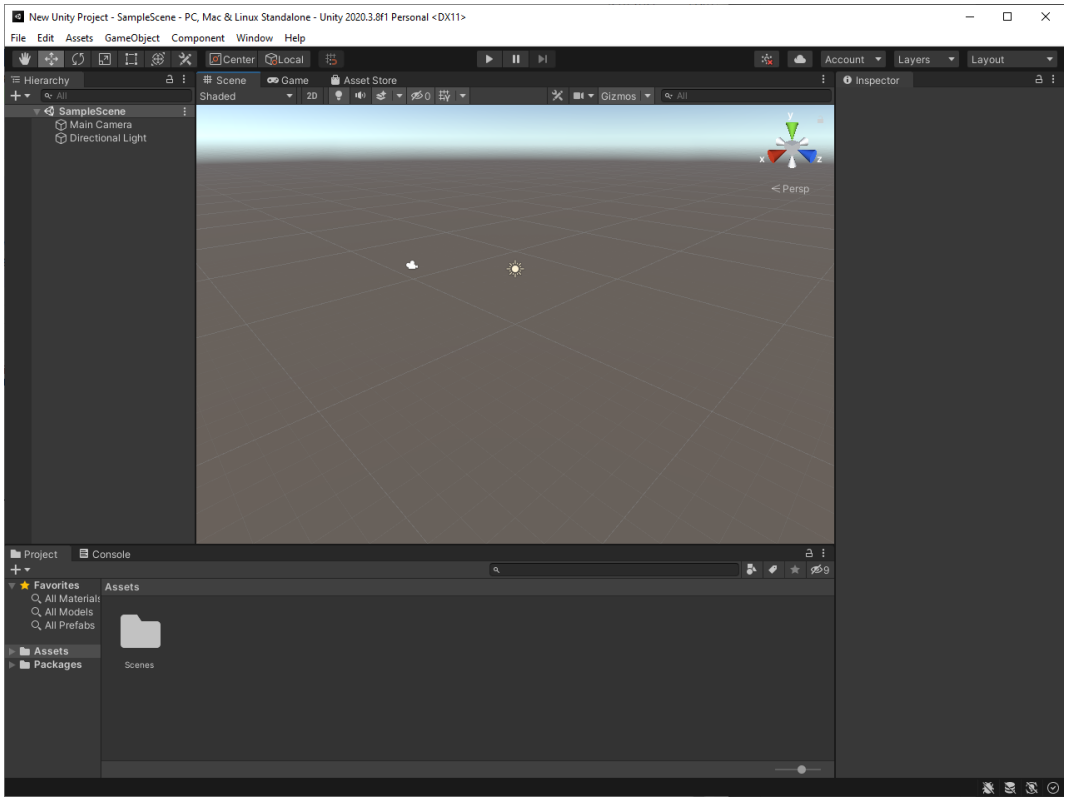


Abbildung 2.1: Unity Editor

Die **Module structures** unterteilen das Software-System in Module. Sie stellen die Strukturierung des Systems als eine Menge von Quellcode-Einheiten dar. Module werden verschiedenem Typen wie *Classes, Packages* oder *Layers* zugeordnet und können in Relation zueinander gesetzt werden. Die Notation der *Module structures* kann zum Beispiel in Form der Unified Modeling Language (UML)[1] erfolgen. Beispiel für eine Frage:

„What other software elements is a module allowed to use?“

Im Rahmen dieser Forschungsarbeit sollen aus forschungsökonomischen Gründen nur *Module structures* berücksichtigt und verwendet werden.

2.2 Unity

Unity [6] (siehe Abb. 2.1) ist eine Entwicklungsplattform für die Erstellung interaktiver Echtzeitinhalte in 2D, 3D, AR und VR. Neben der primären Nutzung als Game Engine für die Entwicklung von Spielen, bietet Unity aber auch Funktionen für industrielle Anwendungen. Beispielsweise die Visualisierung und Interaktion von Produkten auf Basis von CAD Daten. Neben den gängigen PC-Betriebssystemen (Windows, Linux & MacOS) und mobilen Plattformen (iOS & Android) bietet Unity die Möglichkeit Inhalte ebenso für Konsolen (Playstation, XboX und Nintendo Switch) und Webbrowser zu erstellen.

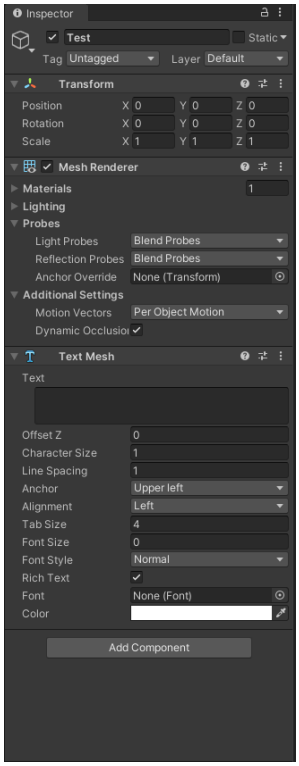


Abbildung 2.2: GameObject mit Komponenten

Objekte werden in Unity durch sogenannte GameObjects (siehe Abb. 2.2) repräsentiert und besitzen Informationen über Position, Größe und Rotation im 2D & 3D Raum. Das Verhalten eines Objektes kann durch *Komponenten* verändert werden, die einem GameObject zugewiesen sind. *Komponenten* sind Skriptinstanzen, die das Verhalten des GameObjects durch Logik beschreiben, auch genannt *MonoBehaviour*. Die zum implementieren von Skripten primär verwendete Programmiersprache ist C#.

2.3 Graph eXchange Language

Die *Graph eXchange Lanugage*[7] (GXL) ist ein standardisiertes Format für den Austausch von Daten in Form von Graphen. Das Format basiert auf der *Extensible Markup Language* (XML), deren Syntax von einer eigenen *Document Type Defintion* (DTD) vorgegeben wird. Diese ermöglicht eine einheitliche Notation von gerichteten Graphen, Hypergraphen und hierarchischen Graphen. Das Format entstand aus der Zusammenführung mehrerer einzelner Dateiformate mit dem Ziel die Interoperabilität zwischen graph-basierten Software-Reengineering-Tools zu verbessern. Das SEE-Projekt (Kapitel 2.5) verwendet aus einer Code-Analyse erzeugte Graphen zur Visualisierung von Software-Systemen. Im Rahmen dieser Forschungsarbeit wird das GXL Format zum Speichern und Laden von modellierten Software-Architekturen benutzt.

2.4 JSON

JSON(JavaScript Object Notation) ist ein leichtgewichtiges Format für den Austausch von Daten [8][9], das sowohl von Maschinen, als auch Menschen gelesen und geschrieben werden kann. JSON basiert auf einer Teilmenge der *JavaScript Programming Language* und definiert eine Reihe von Strukturen für die Repräsentation von Daten:

Objekt

Ein Objekt beschreibt eine ungeordnete Menge von Name/Value-Paaren deren Notation mit **{** beginnt und mit **}** endet. Jedes Paar besteht aus dem Namen und dem Value getrennt durch ein **:**. Mehrere Paare werden durch **,** getrennt.

```
1 {
2     "stringValueName": "stringValue",
3     "stringValueName2": "stringValue2"
4 }
```

Array

Ein Array repräsentiert eine geordnete Menge von Values, die jeweils durch ein **,** getrennt werden. Arrays beginnen mit **[** und enden mit **]**.

```
1 [
2     "value1",
3     "value2",
4 ]
```

Value

In JSON können Values sieben verschiedene Typen haben: String, Number, Object, Array oder einer der Ausdrücke true, false oder null. Dadurch, dass auch die Strukturen Object und Array als Werte gelten, können so außerdem Verschachtelungen erzeugt werden.

```
1  {
2      "stringValueName": "stringValue",
3      "numberValueName": 250,
4      "objectValueName": {
5
6      },
7      "arrayValueName": [
8          "value1",
9          {
10             "name": "valueObject2"
11         },
12         [
13             "value31",
14             "value32"
```

```
        ],
        true,
        false,
        null
    ],
    "trueValueName": true,
    "falseValueName": false,
    "nullValueName": null
}
```

Innerhalb dieser Forschung wird JSON als Datenformat für die Serialisierung von Strichgesten verwendet.

2.5 Projekt SEE

Projekt SEE, abgekürzt für Software Engineering Experience, wird innerhalb der Arbeitsgrupe AG Softwaretechnik der Universität Bremen unter der Leitung von Dr. Prof. Rainer Koschke entwickelt. SEE visualisiert Daten und Informationen, die aus Software-Systemen per statischer Code-Analyse extrahiert wurden, dargestellt als hierarchische Graphen in Form von Code Cities (siehe Abb. 2.3). Neben der klassischen Desktopsteuerung durch Maus & Tastatur werden ebenfalls Touch, AR und VR Eingabegeräte unterstützt.

Neben der Darstellung statischer Informationen eines Software-Systems zu einem bestimmten Zeitpunkt unterstützt SEE ebenfalls die Visualisierung und Animationen der Änderungen dieser über einen Zeitraum hinweg. Außerdem kann das Laufzeitverhalten einer Software aufgezeichnet und anschließend postmortem animiert dargestellt werden.

2.6 Wacom Cintiq 16

Für die Erzeugung von Strichgesten wird für die Durchführung der Untersuchung, ein Stift-Display als Eingabegerät genutzt. Bei dem verwendeten Gerät handelt es sich um das Cintiq 16 (Siehe Abb. 2.4) der Firma Wacom [10]. Das Cintiq 16 ist ein 15,6 Zoll großes Full HD Stift-Display, das per HDMI und USB an einen Computer oder Laptop angeschlossen werden kann. Die Eingabe erfolgt durch Interaktion mit dem mitgelieferten Wacom Pro Pen Slim Stift (Siehe Abb. 2.5).

Zur Erkennung der Stiftposition verwendet das Cintiq die EMR-Technik[11][12] von Wacom. Unterhalb des LCD-Displays sind elektromagnetische Sensoren gitterartig verbaut, welche mehrmals pro Sekunde ihren Betriebsmodus wechseln. Im ’Power mode’ wird ein Magnetfeld erzeugt, das mit einer gewissen Frequenz schwingt. Diese Schwingungen werden von dem im Stift verbauten Schaltkreis aufgenommen und erzeugen eine Alternativ-Schwingung. Die Sensoren wechseln anschließend in den ’Listening mode’ und können diese Schwingung wahr-

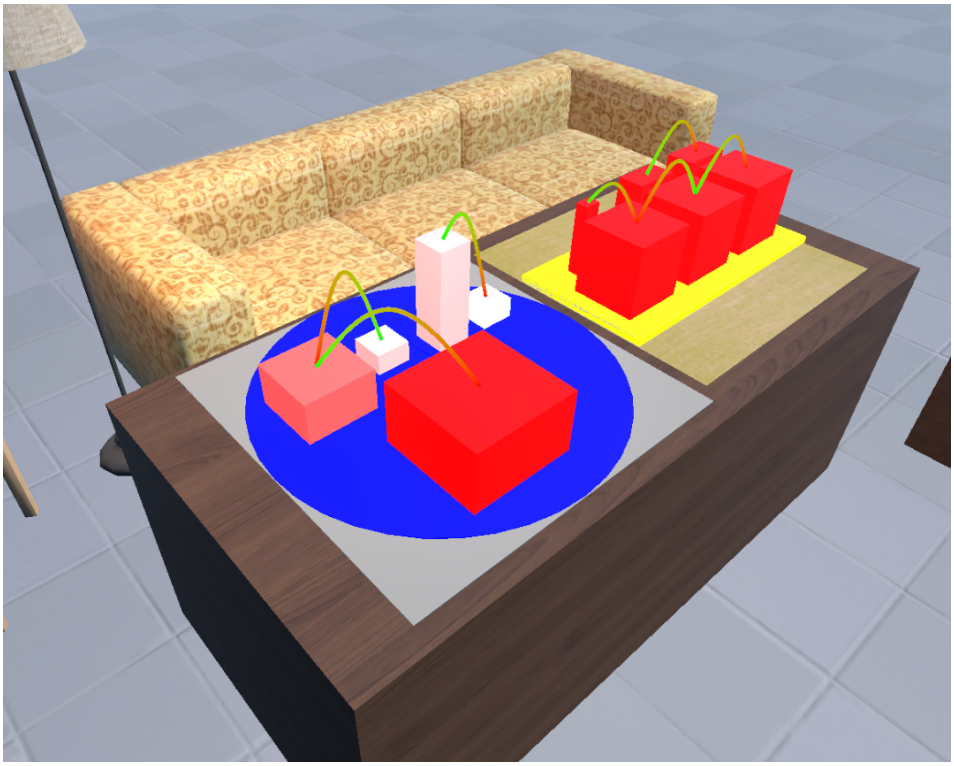


Abbildung 2.3: SEE

nehmen. Indem an mehreren Punkten entlang des Sensorgitters die Signalstärke der im Stift erzeugten Schwingung gemessen wird, können Position und die Druckstärke der Stiftspitze berechnet werden.

2.7 Axivion Gravis

Axivion Gravis ist ein Tool zur Bearbeitung und Betrachtung hierarchischer Graphen und ist Bestandteil der vom gleichnamigen Unternehmen entwickelten Axivion Suite[13], welche eine Reihe an Analyse-Tools für die Softwarequalitätssicherung bereitstellt. In Gravis modellieren Nutzer Softwarearchitektur auf einer zweidimensionalen Zeichenfläche, auf der sie Knoten und Kanten erzeugen, anordnen und bearbeiten können (siehe Abb. 2.6). Neben der Modellierung unterstützt Axivion Gravis auch die Durchführung einer Architekturprüfung, um Fehler bei der Implementierung aufzudecken. Für den Im- und Export von Graphen wird wie in SEE, das GXL-Format genutzt. Es wird außerdem der Export von Layout-Dateien im GVL-Format, basierend auf XML, unterstützt, welches auch von SEE gelesen werden kann. Aus diesem Grund bietet sich Axivion Gravis als optimales Vergleichstool für die im Kontext dieser Forschung durchgeführten Evaluation an.

Abbildung 2.4: Wacom Cintiq 16 Stift-Display[10]

Abbildung 2.5: Wacom Pro Pen slim Stift[10]

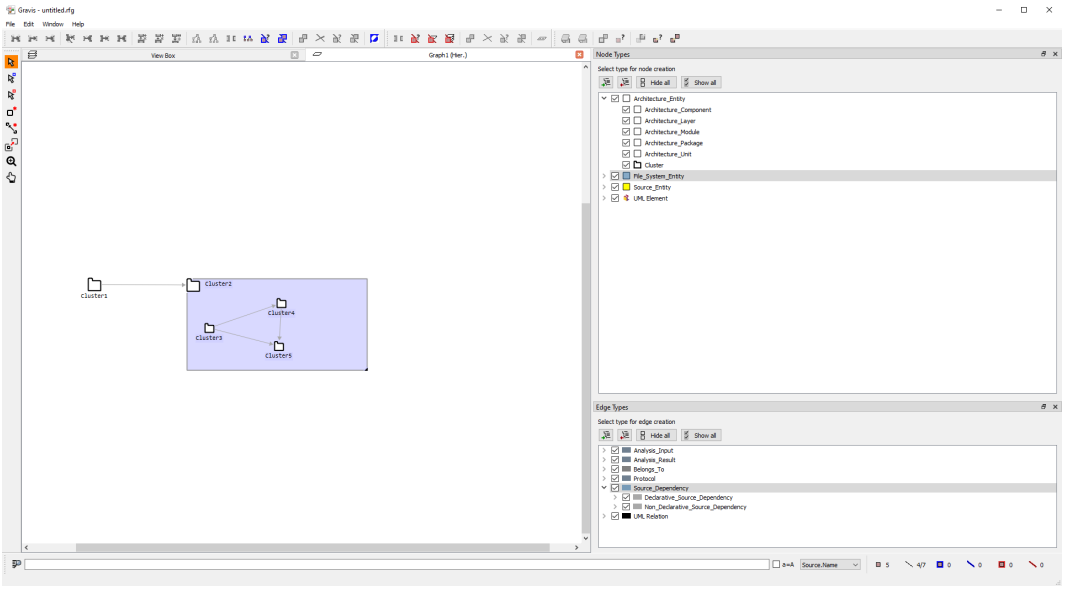


Abbildung 2.6: Axivion Gravis

2.8 Ähnliche Arbeiten

2.8.1 A collaborative Multi-Touch UML Design Tool

Kopf et al.[14] haben in ihrer Arbeit ein kollaboratives Multi-Touch UML Design Tool entwickelt (siehe Abb. 2.7). Sie nutzen einen speziellen Multi-Touch Tisch mit Display, welcher aus Kostengründen auf ein optisches Infrarot-System zur Erkennung von Benutzereingaben setzt. Als Modellierungssprache verwenden Kopf et al. die bekannte Unified Modelling Language (UML)[1]. Allerdings werden nur Klassen, Assoziationen und Pakete als Element-Typen unterstützt. Elemente werden durch Gesten der Benutzer*innen auf dem Display erzeugt. Klassen werden durch Rechtecke, Assoziationen durch Linien zwischen zwei Klassen und Pakete durch das Umschließen von Klassen mit einem Rechteck erzeugt. Die Darstellung der Elemente und Assoziationen orientiert sich direkt an UML. Für die Erkennung der Gesten wurde ein von Wobbrock et al.[15] entwickeltes Verfahren implementiert, das die Gestenpfade der Benutzer*innen normalisiert und anschließend die optimale Punktdistanz im Vergleich zu einer vorab klassifizierten Template-Geste errechnet. Die automatische Anordnung der UML-Elemente nutzt einen Graphalgorithmus von Sugiyama et al.[16], der das Diagramm anhand ästhetischer Kriterien von Eichelberger[17] optimiert. Neben dem Im- und Export der Diagramme im XML-Format unterstützt das Tool ebenso den Export des UML-Diagramms als Java Code Skelett.

2.8.2 VR-UML: The unified Modeling Language in Virtual Reality

Oberhauser stellt in *VR-UML: The unified Modeling Langauge in Virtual Reality*[18] ein Konzept zur Modellierung von UML-Diagrammen in einer Virtual-Reality Umgebung vor. Die Arbeit erweitert das *Virtual Reality - Modeling Fra-*

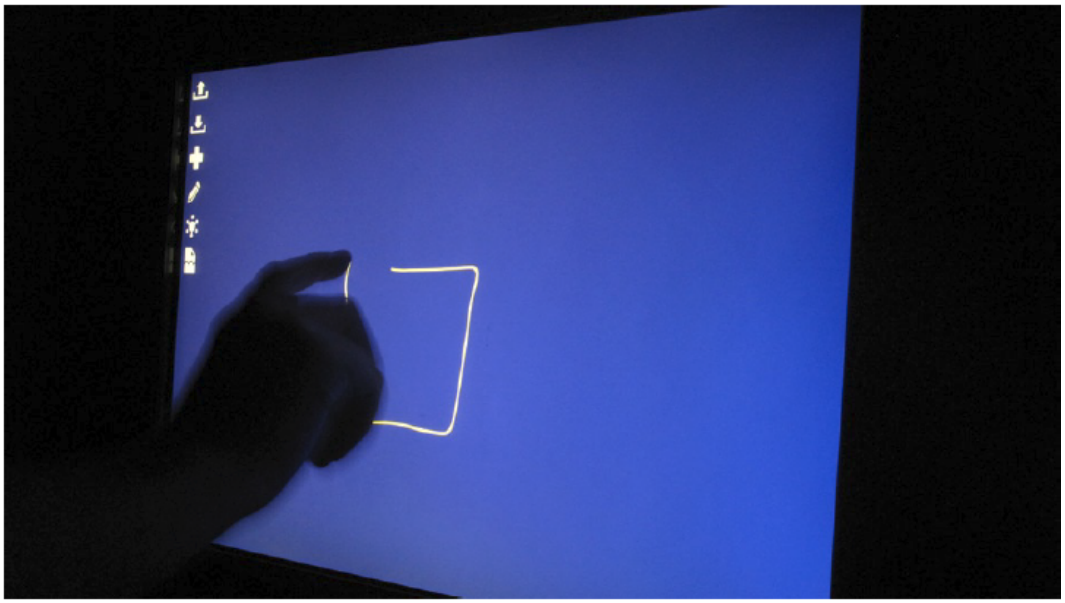


Abbildung 2.7: Kollaboratives Multi-Touch UML Design Tool[14]

mework (VR-MF), welches aus einer vorherigen Forschung von Oberhauser et al.[19] stammt. Die Umsetzung erfolgte in der Unity Engine mit Verwendung des SteamVR-Plugins. Im Gegensatz zu Kopf et al. unterstützt *VR-UML* mehrere UML-Diagrammtypen: Anwendungsfall, Klassen, Sequenz und Verteilungsdiagramme. Die einzelnen Diagramme werden auf den sogenannten *Hyperplanes* (siehe Abb. 2.8) visualisiert. Die UML-Elemente werden als generische Rechtecke dargestellt, deren Typen über Icons an den Ecken des Rechtecks zu identifizieren sind. Assoziationen werden als dreidimensionale Pfeile zwischen den Elementen visualisiert. Benutzer*innen interagieren mit den Diagrammen und ihren Elementen über ein virtuelles Keyboard (siehe Abb. 2.9) das CRUD-Operationen (Create, Retrieve, Update, Delete) abhängig des Typ bereitstellt. Der Im- und Export der Diagramme erfolgt in Form eines JSON-Formats.

2.8.3 Modellierung von Softwarearchitektur in Virtual Reality mithilfe der Leap Motion

SEE wurde im Rahmen der Abschlussarbeit von Döhl[20] bereits um eine Möglichkeit zur Modellierung von Software-Architektur in Virtual Reality erweitert. Die Implementierung erlaubt es Benutzer*innen mithilfe von einfachen Handgesten Softwarearchitektur zu modellieren. Für das Tracking der Hände wird ein Leap Motion[21] Sensor verwendet, bei dem es sich um eine stereoskopische ultraweitwinkel Infrafrotkamera handelt. Sie wird über ein Unity-Plugin integriert und erlaubt es die Position der Hände und Finger im dreidimensionalen Raum zu erkennen. Es wurden verschiedene Handgesten für die folgenden Funktionen implementiert:

- Erzeugen von Knoten

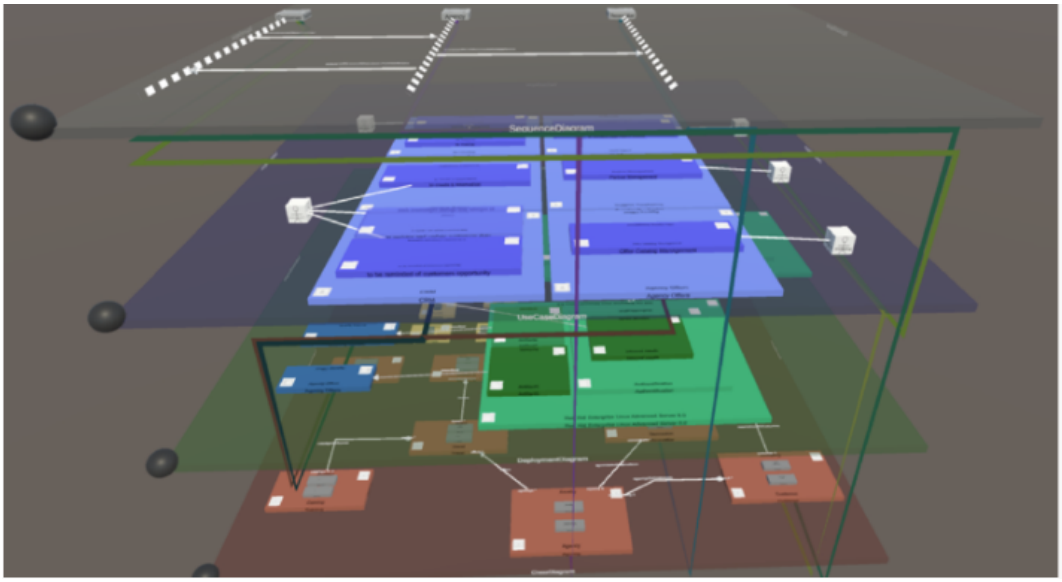


Abbildung 2.8: Hyperplanes unterschiedlicher Diagrammtypen[18]

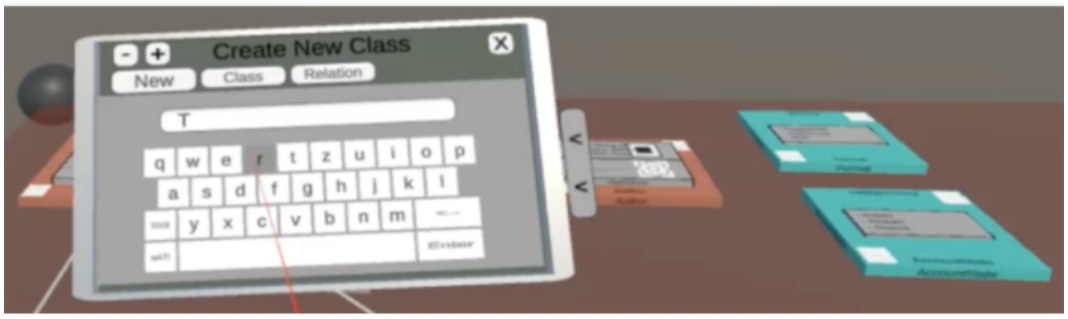


Abbildung 2.9: Virtuelles Tablet[18]

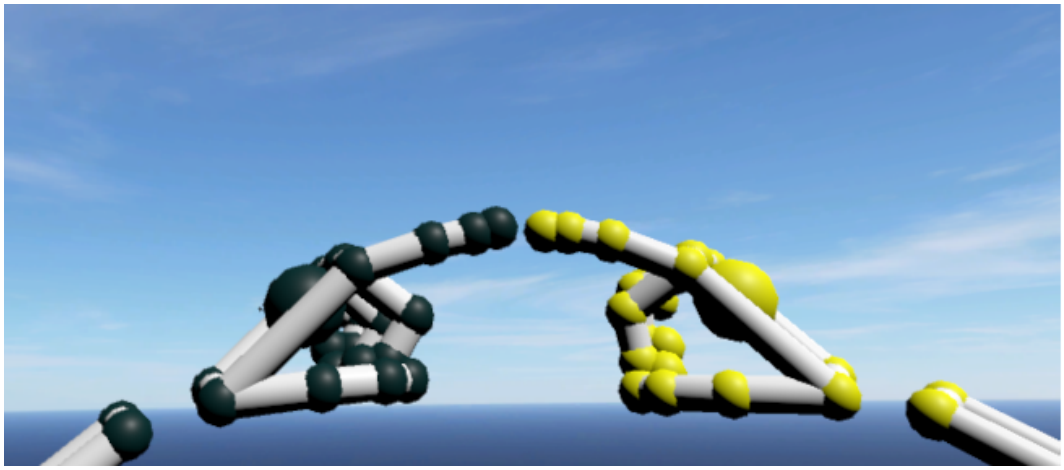


Abbildung 2.10: LeftToRightIndex Handgeste [20]

- Bearbeiten/Verschieben der Knoten
- Erzeugen/Entfernen von Kanten
- Import/Export einer Software-Architektur

In Abbildung 2.10 kann die Geste zum Erzeugen von Elementen betrachtet werden. Abbildung 2.11 zeigt die Darstellung von Komponenten und Abhängigkeiten am Beispiel der alten Twitter-Architektur. Architekturelemente bestehen aus einem grauen Quadrat mit dem Namen an der oberen Kante. Die Abhängigkeiten werden durch farbige Kanten zwischen den Elementen dargestellt. Anhand der Farbe der Kante können Ursprung (grün) und Ziel (rot) der Abhängigkeit identifiziert werden. Der Im- und Export der Architekturen geschieht als GXL-Datei.

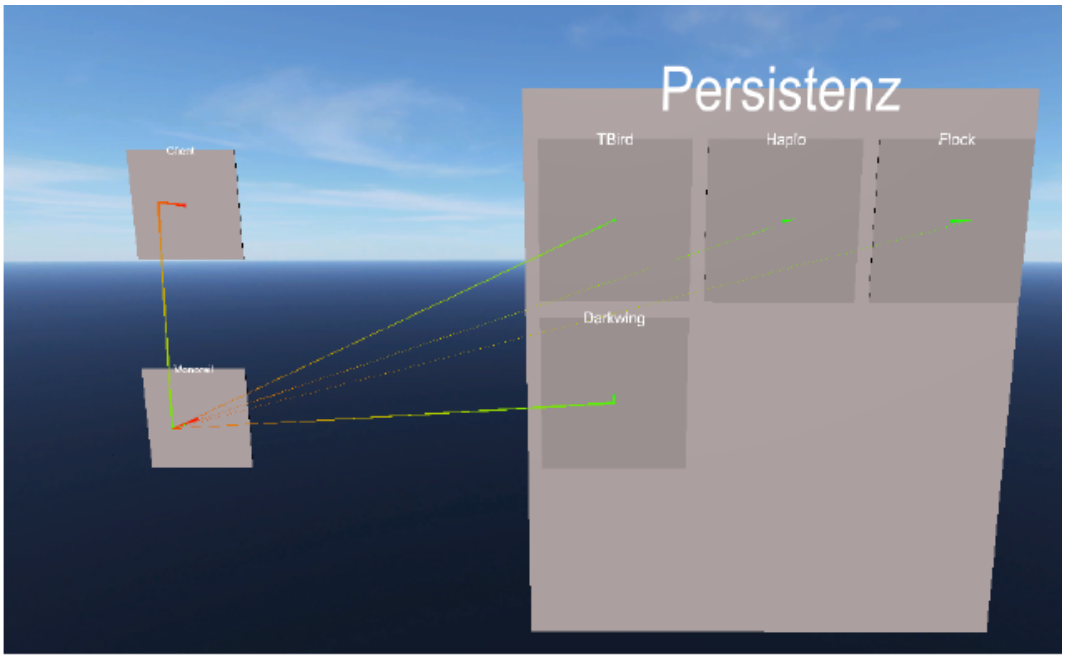


Abbildung 2.11: Visualisierung einer alten Teil-Architektur von Twitter [20]

Kapitel 3

Entwurf

Im folgenden Kapitel werden die Entwurfsidee und die daraus resultierenden Anforderungen an die Implementierung des Modellierungstools für SEE erläutert.

3.1 Entwurfsidee

Das Design der Erweiterung soll sich an einem Whiteboard orientieren und dem Nutzer mithilfe des Stift-Displays die Möglichkeit geben durch Strichgesten Architekturelemente und Abhängigkeiten zu erzeugen (siehe Abb. 3.1). Die Form der Strichgeste entscheidet über den Typ der erzeugten Architekturkomponente und deren visuelle Darstellung. Im Rahmen dieser Forschung sollen vorerst nur zwei verschiedene Typen von Komponenten berücksichtigt werden: *Cluster* und *Component*. *Cluster* sollen durch Rechtecke erstellt werden können und *Components* durch Kreise. Die Größe und Position der Strichgeste wird direkt auf die generierte Architekturkomponente angewandt. Abhängigkeiten werden durch direkte Linien zwischen den Formen oder durch Verschachtelung erzeugt. Die so erzeugten Architekturkomponenten können im nächsten Schritt vom Nutzer bewegt, bearbeitet oder gelöscht werden. Außerdem soll der Nutzer die Möglichkeit haben seine modellierte Software-Architektur als GXL zu speichern. Die Allgemeinen Funktionen sind per Menüleiste und die für den jeweiligen Typ von Architekturkomponenten spezifischen Funktionen sind über ein Kontextmenü erreichbar (siehe Abb. 3.2).

3.1.1 Gestenerkennung

Um die Strichgesten des Nutzers einer Architekturkomponente zuordnen zu können, bedarf es einem Algorithmus der jene Gesten eindeutig klassifiziert und identifiziert. Dieser sollte möglichst effizient agieren, um für den Nutzer sichtbare Verzögerungen bei der Verarbeitung der Daten zu minimieren. Außerdem sollte der Algorithmus die Gesten mit einer hohen Genauigkeit zuordnen können. Ebenso wichtig ist die Robusheit gegenüber der Varianz der Art und Weise wie Nutzer eine Strichgeste zeichnen, bspw. Nutzer A beginnt beim Quadrat links unten, Nutzer B aber oben rechts.

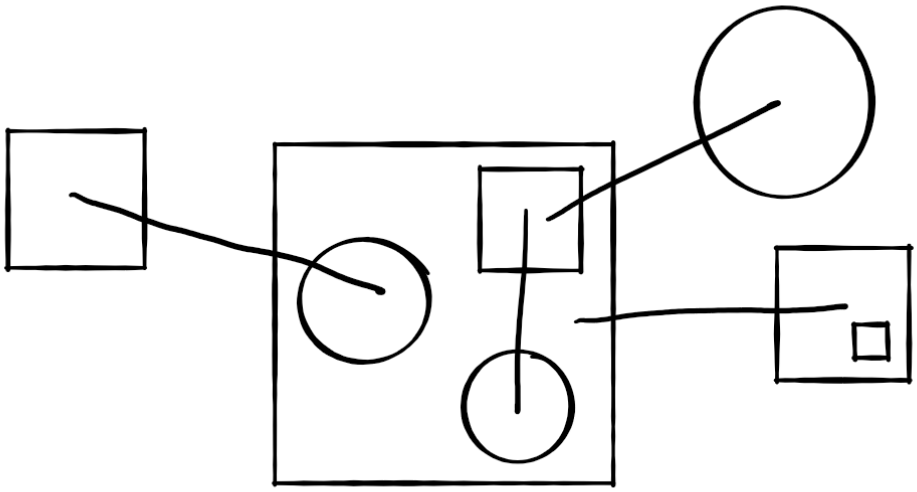


Abbildung 3.1: Mockup in Microsoft Whiteboard App

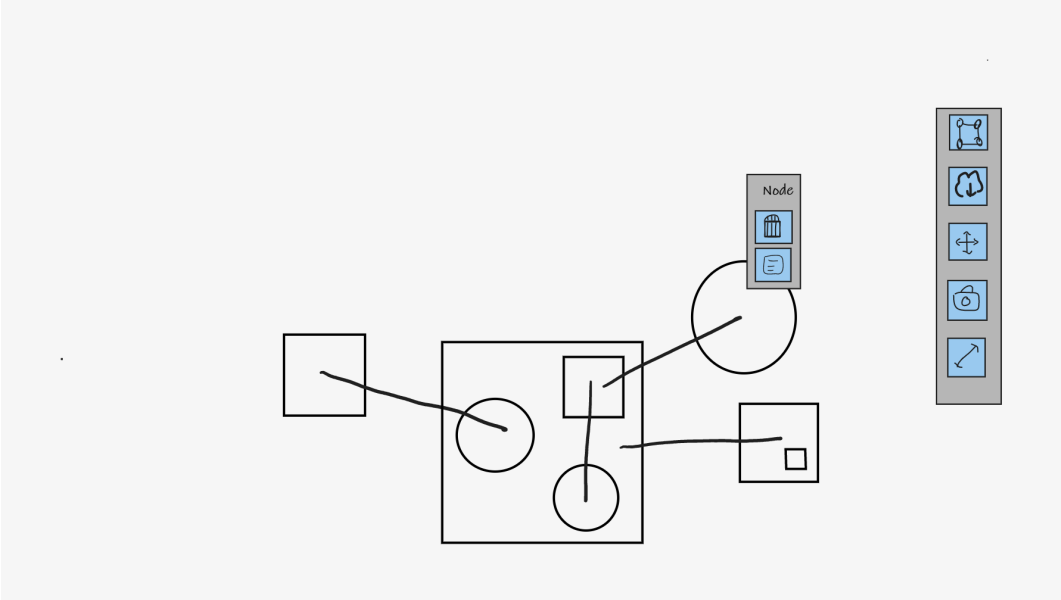


Abbildung 3.2: UI Mockup in Microsoft Whiteboard App

3.1.2 Darstellung der Architektur

Die Architektur soll auf einer Whiteboard-Fläche dargestellt und platziert werden. Komponenten der Architektur werden anhand ihres Types optisch differenziert präsentiert. Deren Abhängigkeiten werden entweder durch Verschachtelung in Form von aufeinander gestapelten Komponenten oder durch Kanten als direkte Linien angezeigt. Die Tiefe der Verschachtelung wird farblich abgestuft dargestellt.

3.1.3 Erzeugen von Architekturkomponenten

Durch zeichnen von Strichgesten werden neue Architekturkomponenten generiert. Strichgesten auf der Whiteboard-Fläche schaffen neue freistehende Komponenten ohne Abhängigkeiten, während Gesten auf bestehenden Komponenten explizit Beziehungen erzeugen. Quadrate oder Rechtecke als Strichgesten erzeugen Cluster Komponenten. Der Typ Components wird durch kreisförmige Gesten generiert. Abhängigkeiten können ebenfalls durch Verbinden zweier Komponenten mithilfe einer Linie erschaffen werden.

3.1.4 Bearbeiten von Architekturkomponenten

Architekturkomponenten können einzeln durch den Nutzer ausgewählt und bearbeitet werden. Es besteht die Möglichkeit über einen Dialog den Namen und den Typ der Komponenten zu ändern, sie auf dem Whiteboard zu bewegen und in andere Komponenten zu verschieben, um sie zu verschachteln. Ebenso können Komponenten gelöscht werden.

3.1.5 Laden & Speichern der Software-Architektur

Der Nutzer muss die Möglichkeit haben seine bestehende Architektur im GXL Format in Kombination mit einer bestehenende Layoutdefinition im SLD oder GVL Format in das Modellierungstool laden zu können. Darüberhinaus sollte er ebenfalls in der Lage sein, seine Änderungen in einer Datei abzuspeichern.

Kapitel 4

Implementierung

4.1 Unity Setup

Zu aller erst müssen alle Standard Objekte aus der Unity Szene entfernt werden. Anschließend wird ein leeres GameObject erzeugt und die *Player Settings* Komponente hinzugefügt. Über den Button *Add required objects to scene* werden anschließend die benötigten Objekte und Komponenten zur Szene hinzugefügt. Um die Visualiserung der Software-Architektur konfigurieren zu können, muss nun ein Objekt mit der *SEECityArchitecture* Komponente erzeugt werden, welches ebenfalls über die *PlayerSettings* getan wird. Im Abschnitt *Create a new code city* muss als *City type* die *SEECityArchitecture* ausgewählt und als Name *Architecture* eingetragen werden. Anschließen wird mit dem Button *Create City* das entsprechende Objekt in der Szenenhierarchie erzeugt. Damit der Nutzer beim Start von SEE das richtige Objekt fokussiert, muss im Inspektor der *PlayerSettings* als *Focus Plane* noch die *Plane* Komponente des eben erzeugten Architektur Objekts zugewiesen werden. Im Anschluss sollte die Szene in Unity wie in Abb. 4.1 aussehen.

4.1.1 Darstellungsoptionen

In Abbildung 4.2 sieht man die Architektur-spezifischen Darstellungsoptionen des Graphen:

1. Hier können Architekturgraph und dessen Layoutdatei gewählt werden. Das Tool unterstützt die im Rahmen von SEE verwendeten .sld als auch die von Axivion Gravis generierten .gvl Layoutdefinitionen für Graphen.

2. Mit dem Button *New Graph* kann ein neuer Archtitekturgraph erzeugt werden.

3. Der Button *Load Graph* lädt die unter *Architecture Data Settings* gewählten Daten.

4. Im Abschnitt *Architecture Element settings* kann die Darstellung der Knoten angepasst werden.

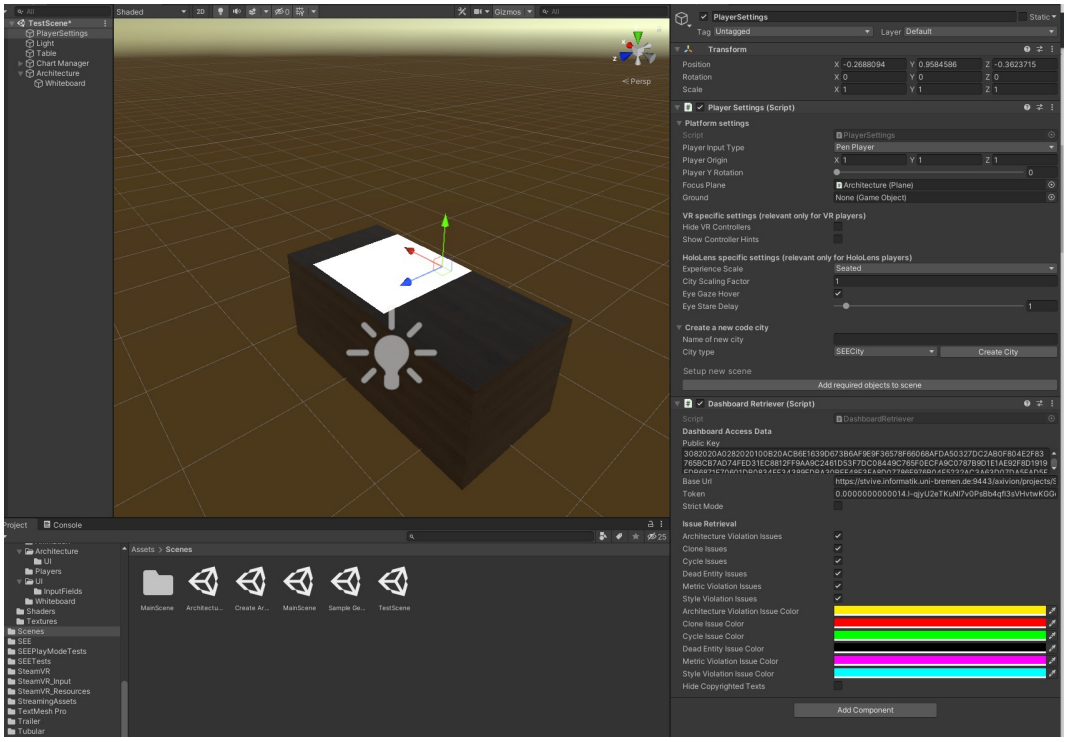


Abbildung 4.1: Unity Szene

- **Element height**: Höhe der Knoten
- **Lower color - Upper color**: Start und Endfarbe des Gradienten zur Visualisierung der einzelnen Verschachtelungsebenen
- **#Colors**: Anzahl der einzelnen Farbabstufungen

5. Der Abschnitt *Edges and edge layout* steuert die Darstellung der Kanten:

   - **Edge with**: Strichstärke der Kante.
   - **Edges above blocks**: Kanten können entweder unterhalb der Knoten oder oberhalb dargestellt werden. Diese Option sollte immer aktiviert sein, da es ansonsten zu Problemen beim Auswählen von Kanten kommen kann.
   - **Bunding tension**: Keine Bedeutung für Architekturgraphen
   - **RDP**: Keine Bedeutung für Architekturgraphen
   - **Tubular Segments**: Steuert die Menge die Segmentierung des Meshes entlang der Kante.
   - **Radius**: Steuert den Radius des erzeugten Meshes.
   - **Radial Segments**: Steuert die Menge der radialen Segmentierung des Meshes.
   - **Edge selectable**; Sollte immer aktiviert sein, ansonsten können keine Kanten ausgewählt und gelöscht werden.

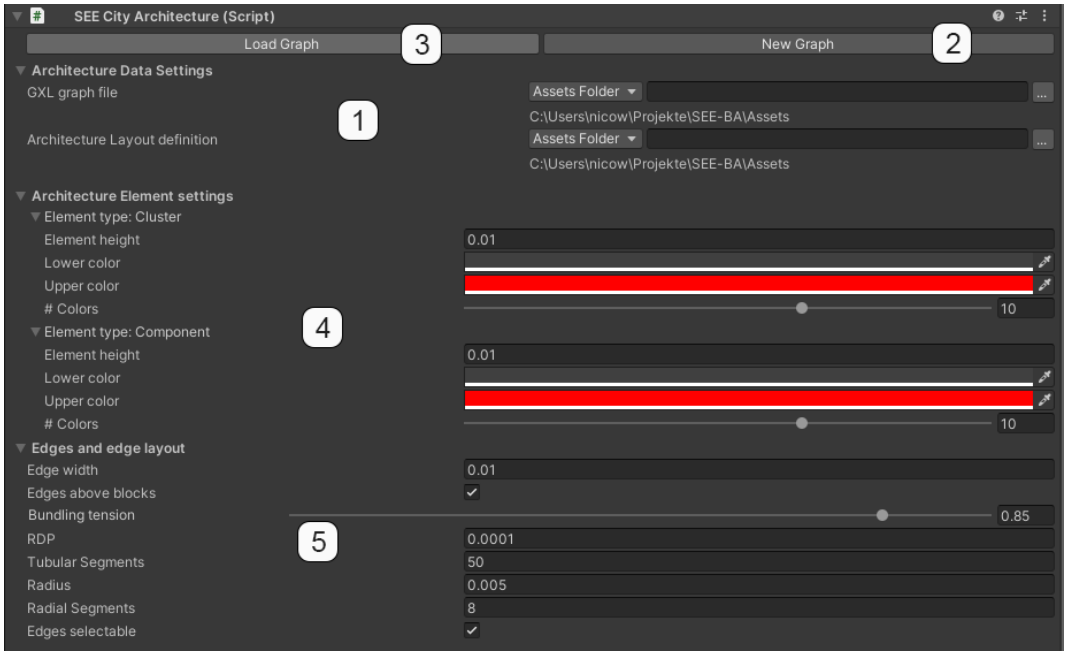


Abbildung 4.2: SEECityArchitecture Settings

4.2 Gestenerkennung

4.2.1 Stifteingabe

Damit der Stift des Wacom Cintiq Pro 16 als Eingabgerät genutzt werden kann, musste zusätzlich zum klassischen Unity Input system *UnityEngine.Input* das erst kürzlich veröffentlichte neue Input System [22] eingesetzt werden. Dieses bietet offiziellen Support für Stifteingaben und bietet die Möglichkeit im C#-Code direkt auf den aktuell verbundenen Stift und dessen State zuzugreifen (siehe Listing 4.1). Dies führt allerdings dazu, dass die verarbeitende Logik in der *Update()* implementiert werden muss, was schnell zu unübersichtlichen Code führen kann. Unity bietet zusätzlich die Möglichkeit Logik Event-driven zu implementieren. Dazu muss vorab ein **InputActionAsset** (siehe Abb. 4.4) erzeugt werden, das es ermöglicht seperate InputActions auf Stifteingaben zu mappen. Anschließend kann im Code direkt auf die Events der vorab definierten InputActions reagiert werden (siehe Listing. 4.2).

Listing 4.1: Input Action Logik im Update()

```
1 public void Update()
2 {
3     if(Pen.current.tip.wasPressedThisFrame)
4     {
5         // Logik
6     }else if(Pen.current.tip.wasReleasedThisFrame)
7     {
8         // Logik
9     }else if(Pen.current.tip.isPressed)
```

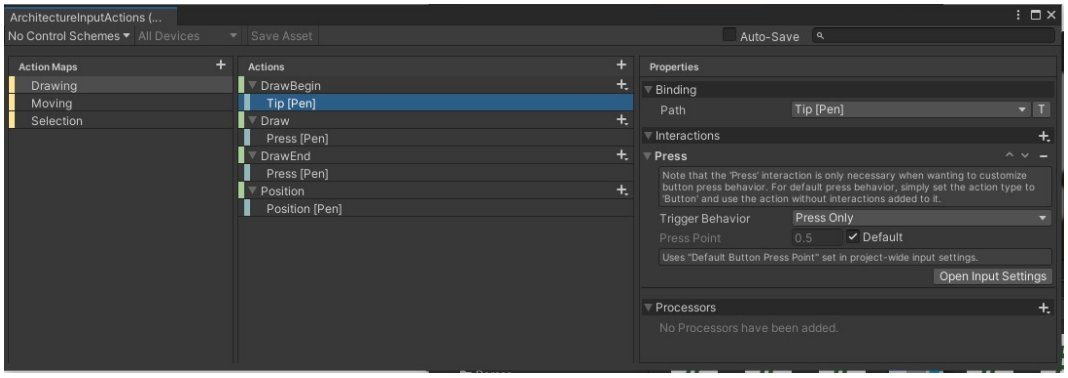


Abbildung 4.3: Architecture Input Actions

```
10      {
11          // Logik
12      }
13  }
```

Listing 4.2: Event-driven Input Action Logik

```
1  public void Awake()
2  {
3      ArchitectureInputActions actions = new ArchitectureInputActions();
4      actions.Drawing.DrawBegin.performed += OnDrawBegin;
5      actions.Enable();
6  }
7  private void OnDrawBegin(InputAction.CallbackContext _)
8  {
9      //Logik
10 }
```

Listing 4.3: Extrahieren der Strichgestenpunkte

```
1  public static Vector3[] ExtractGesturePoints(GameObject gestureGO)
2  {
3      TrailRenderer renderer = gestureGO.GetComponent<TrailRenderer>();
4      Vector3[] points = new Vector3[renderer.positionCount];
5      renderer.GetPositions(points);
6      return points;
7  }
```

4.2.2 Strichgesten

Bei einer Strichgeste handelt es sich im Folgenden um eine durch den Benutzer ausgeführte Geste in Form einer oder mehrerer zusammenhängender Linien. Die Linien werden durch Bewegungen der Finger oder eines Eingabegeräts wie einer Maus erzeugt. Im Rahmen dieser Arbeit passiert dies durch die Bewegung des Stifts auf dem Grafiktablett. Die Linien der Strichgesten werden durch Mengen von Knoten beschrieben.

Die in SEE verwendeten Strichgesten werden mithilfe der Unity **TrailRenderer** Komponente erzeugt. Hinzugefügt zu einem **GameObject** erlaubt es die Komponente Linien anhand dessen Bewegung zu erzeugen. Die Klasse **CreateArchitectureAction** instantiiert ein solches Prefab, sobald der Stift das Display berührt. Berührt der Stift auch weiterhin das Display wird die Position des **GameObject** in jedem **Update()** an dessen Position verschoben. Dazu müssen die Positionsvektoren vom Viewport Koordinatenraum, in welchem sich der Stift befindet, in den World Koordinatenraum umgewandelt werden. Sobald der Stift nicht mehr das Display berührt ist die Strichgeste beendet. Die Punkte können anschließend über die **GetPositions()** Methode des **TrailRenderer** extrahiert werden (siehe Listing 4.3), um sie im nächsten Schritt durch den in 4.2.3 beschriebenen Algorithmus erkennen zu lassen.

4.2.3 Algorithmus

Auf der Suche nach einem Algorithmus der die in 3.1.1 definierten Anforderungen erfüllt, findet man häufig die $-Family von Algorithmen zur Erkennung von zweidimensionalen Strichgesten. Die $-Family vereint den Ansatz nur einfache geometrische Berechnungen und einfach zu verstehende Datenmodelle für die Erkennung von Gesten zu verwenden, so dass eine Vielzahl von Projekten und Endgeräte daraus einen Nutzen ziehen können[23].

$1

Die Basis der $-Familie von Algorithmen bildet $1 von Wobbrock et al. [24]. $1 ist in vier Schritte unterteilt.

In Schritt eins werden die Punkte M der Strichgeste resampled, so dass sie durch N gleichmäßig verteilte Punkte dargestellt wird. Nach Wobbrock et al. sollte $N \leq 32 \leq 256$ gewählt werden, um einen Kompromiss zwischen Präzision und Ausführungsdauer des Algorithmus zu erreichen.

Im zweiten Schritt wird die Strichgeste so rotiert, dass deren Überlappung mit einer anderen Strichgeste möglichst genau ist. Dazu definieren Wobbrock et al. den indikativen Winkel einer Strichgeste als den Winkel der vom ersten Punkt x_0 und deren Mittelpunkt $(\overline{x}, \overline{y})$ beschrieben wird. Anschließend wird die Rotation der Geste so angepasst, dass der indikative Winkel 0 beträgt.

Schritt drei des Algorithmus' skaliert so, dass die Strichgeste in ein einheitliches Referenzquadrat passt. Im Anschluss werden die Punkte der Strichgeste als ganzes verschoben, damit der Mittelpunkt der Geste $(\overline{x}, \overline{y})$ im Punkt $(0, 0)$ platziert ist.

Alle Algorithmen der $-Familie vergleichen eine durch den Nutzer erzeugte Strichgeste (im Folgenden Candidate) mit einer Menge von vorab aufgenom-

menen Strichgestenpunkten (im Folgenden Templates) und versuchen die beste Übereinstimmung zu finden. Die ersten drei Schritte des Algorithmus' müssen für den Candidate und die Templates durchgeführt worden sein, um in Schritt vier mit der Klassifizierung starten zu können.

In Schritt vier vergleicht $1 den Candidate C mit jedem Template $T_i \in T$ und errechnet die durchschnittliche Distanz d_1 zwischen den zusammengehörigen Knoten (siehe Formel 4.1). Das Minimum aller errechneten Durchschnitte d_i ist das Ergebnis des Algorithmus' und wird mithilfe der Formel 4.2 in einen numerischen Wert von $[0..1]$ umgewandelt. $size$ ist die Länge einer Seite des in Schritt zur Skalierung verwendeten Quadrats. Der errechnete Score gibt Auskunft über die Genauigkeit der Übereinstimmung mit 1 als 100% Übereinstimmung und 0 als 0% Übereinstimmung.

$$d_i = \frac{\sum_{k=1}^{N} \sqrt{(C[k]_x - T_i[k]_x)^2 + (C[k]_y - T_i[k]_y)^2}}{N} \tag{4.1}$$

$$score = 1 - \frac{d_i^*}{\frac{1}{2}\sqrt{size^2 + size^2}} \tag{4.2}$$

In ihrem Paper haben Wobbrock et al. zeigen können, dass die Genauigkeit von $1 mit nur einem geladenen Template bei bereits 97% beträgt und sich ab drei Templates auf über 99% erhöht.

$N

Der von Anthony entwickelte $N-Algorithmus [25] erweitert das Grundkonzept von $1 um eine Möglichkeit zur Erkennung von Multistrichgesten. Dazu bildet $N alle möglichen Einzelstrich Permutationen der Multistricheingabe, indem er die Striche in allen möglichen Reihenfolgen künstlich miteinander verbindet und so effektiv Einzelstrichgesten erzeugt, die wiederum vom bestehenden $1-Algorithmus klassifiziert werden können. Das Erzeugen der Permutationen führt allerdings zu einem deutlich höheren Bedarf an Speicher und Ausführungsdauer als bei $1 [26]und wird daher für die Implementierung nicht in Betracht gezogen.

$P

$P [26] ist die nächste Stufe der Weiterentwicklungen der $-Family und versucht die Probleme von $N zu adressieren. $P unterstützt ebenfalls die Klassifizierung von Multistrichgesten, verfolgt aber einen anderen Ansatz als $N es tut. Anstatt die Strichgesten als einzelne geordnete Mengen von Punkten zu sehen, stellt $P sie als eine ungeordnete Menge von Punkten dar, eine sogenannte Punktewolke oder auch Pointcloud. Das Ziel ist es das Minimum der benötigten Kosten zu finden, um die Punkte zweier Wolken aufeinander abzubilden. Dazu wird ein Greedy Algorithmus verwendet, der zu jedem Punkt in der ersten Wolke C_i den nächsten noch nicht gematcheten Punkt aus der zweiten Wolke sucht. Solange bis zu jedem Punkt ein passender Partner gefunden wurde. Zusätzlich werden

die Ergebnisse der euklidischen Distanz für die ersten Matches höher gewichtet, als für die letzten. Das liegt daran, dass mit hoher Wahrscheinlichkeit der gefundene Punkt aus der zweiten Wolke auch wirklich der nächste Nachbar ist, während für die letzten Punkte nicht mehr die volle Auswahl an möglichen Matches zur Verfügung steht. Die Gewichtung berechnet sich anhand der Formel 4.3 und die Summe der Kosten für die Abbildung aus Formel 4.4.

$$\omega_i = 1 - \frac{i-1}{n} \tag{4.3}$$

$$\sum_i \omega_i \cdot \parallel C_i - T_j \parallel \tag{4.4}$$

Das Ergebnis vom Greedy Algorithmus ist abhängig der Reihenfolge der Punktewolken, weshalb die Berechnung mit getauschten Wolken wiederholt werden muss. 4.5

$$min(GREEDY - X(C,T), GREEDY - X(T,C)) \tag{4.5}$$

Im direkten Vergleich mit $1 und $N konnte Vatavu feststellen [26], dass die Genauigkeit von $P sowohl für Einzel als auch Multistrichgesten höher ist, bei gleichzeitig deutlich geringerem Implementierungsaufwand. Daher wurde sich bei der Implementierung der Gestenerkennung für $P entschieden.

4.2.4 Umsetzung

GesturePoint

Die Klasse **GesturePoint** stellt einen Punkt in einer zweidimensionalen Punktewolke dar, dessen Position durch die Koordinaten x und y beschrieben wird. Das Attribut *strokeID* definiert welchem Strich einer Multistrichgeste der Punkt angehört.

Gesture

Die Klasse **Gesture** repräsentiert eine Strichgeste als normalisierte zweidimensionale Punktewolke der Größe n = 32. Beim instantiieren der Klasse werden die Punkte der Strichgeste gemäß der in 4.2.3 erläuterten Schritte 1-3 von $1 mit der privaten Methode **Normalize** normalisiert.

DollarPGestureRecognizer

Die Funktionen des $P-Algorithmus werden durch die Klasse **DollarPGestureRecognizer** bereitgestellt. Die Methode **TryRecognizeGesture** extrahiert die Punkte, erzeugt die für $P notwendigen Datenstrukturen und führt diesen auch aus. Die für die Klassifizierung verwendeten Templates befinden sich als JSON im Ordner *\Assets\StreamingAssets\Architecture\GestureSet* (siehe Listing 4.4) und können über die Klasse **GestureIO** geladen und gespeichert werden. Im Abschnitt 4.2.5 wird zudem erläutert wie man als SEE Entwickler neue Templates für die Gestenerkennung erzeugen kann.

Listing 4.4: Serialisierte Gesten

```
{
    "Points": [
        {
            "X": 0.45,
            "Y": 0.63,
            "StrokeID": 0
        },
        ...
    ],
    "Name": "circle"
}
```

4.2.5 Sampling

Um als SEE Entwickler neue Gesten Templates zu erzeugen kann die bereitgestellte Unity Scene **Sample Gesture Data**, die sich unter *\Assets\Scenes*befindet, verwendet werden. Diese beinhaltet alle nötigen GameObjects und Komponenten um die Gestenerkennung außerhalb der SEE Umgebung zu testen oder um neue Templates zu generieren. Das Verhalten kann über die Komponente **GestureRecorder** konfiguriert werden:

- **leanPlane**: Referenz auf die verwendete **LeanPlane** Zeichenfläche
- **sample**: Ist dieses Flag aktiviert, so werden aus Strichgesten neue Templates erzeugt, serialisiert und im Dateiverzeichnis abgelegt. Andernfalls werden Strichgesten mithilfe des $P-Recognizers gegen die bereits existierenden Templates getestet
- **gestureName**: Name der Geste für die neue Templates erzeugt werden sollen

Über das UI-Panel in der oberen rechten Ecke werden außerdem Informationen zu den erzeugten Gesten bereitgestellt.

Zum Zeitpunkt der Abgabe befinden sich insgesamt 10 Templates für die Kreisgesten, 33 Templates für die Liniengeste und 22 Templates für die Viereckgeste im Projekt.

4.3 Architekturmodus

Im Folgenden Abschnitt werden der in SEE integrierte Architekturmodus und dessen Funktionen vorgestellt. Der Wechsel geschieht über den Architecture Eintrag im PlayerMenu, das über den Menu Button unten links geöffnet werden kann. Eine Unity Scene, die wie in Abschnitt 4.1 eingerichtet wurde, befindet sich mit dem Namen *Create Architecture* im Verzeichnis *\Assets\Scenes* .

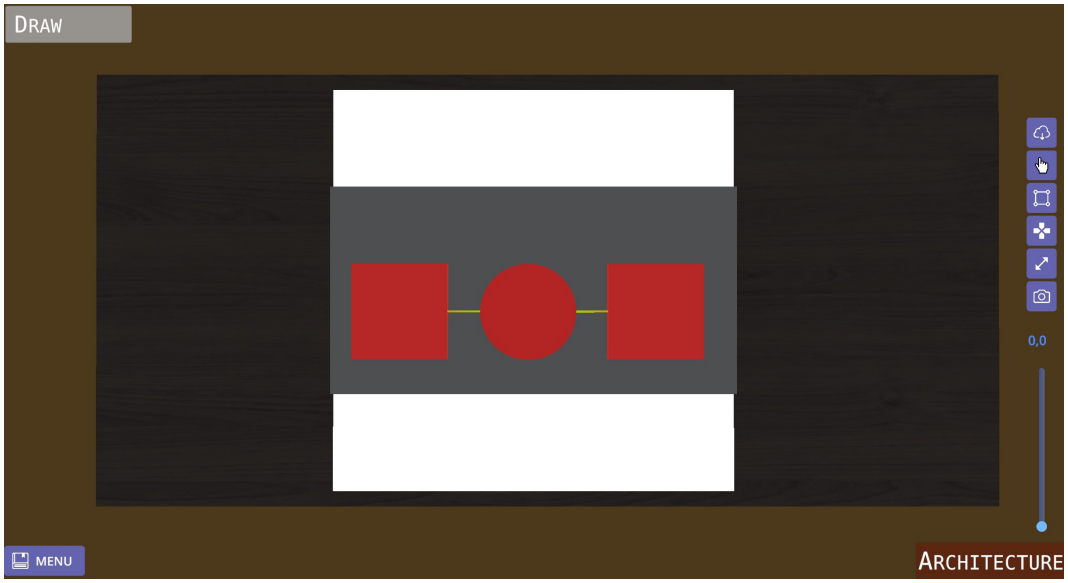


Abbildung 4.4: UI im Architekturmodus

4.3.1 UI

Im Architekturmodus befindet sich oben links der Indikator für das aktuell aktivierte Tool. Bei erstmaliger Aktivierung startet der Nutzer automatisch im Draw Tool. Auf der rechten Seite befindet sich die Menüleiste mit den Button zur Auswahl des Modellierungstool die der Nutzer wählen kann. Unterhalb dieser befindet sich ein Slider zur Vergrößerung der Architektur. Die UI wird von der Klasse **ArchitectureAction** erzeugt und verwaltet. Die einzelnen UI-Elemente liegen als vorkonfigurierte Prefabs im Verzeichnis *\Assets\Resources \Prefabs\Architecture\UI*. Die Bedeutungen der einzelnen Buttons werden in Abb. 4.5 gezeigt und in den folgenden Abschnitten genauer erläutert.

4.3.2 Darstellung der Architektur

Bei der Visualisierung der Softwarearchitektur wurde sich zum Großteil an der existierenden Implementierung von SEE orientiert und, wo sinnvoll, auch bestehender Code wiederverwendet. Der **GraphRenderer** und die **SEECity** für die Darstellung der CodeCities werden durch den **ArchitectureRenderer** und die **SEECityArchitecture** ersetzt. Der neue Renderer bedient sich der bereits vorhandenen Cube-und CylinderFactories für die Darstellung der Knoten anhand ihres Types. Die Kanten werden außerdem als gerade Linien zwischen den Punkten gerendert und die Knoten, wie in 3.1.2 definiert, auf einer weißen Whiteboard Fläche platziert. Die Namen der Knoten und der Kanten werden durch Hovern des Cursors angezeigt (siehe Abb. 4.6).

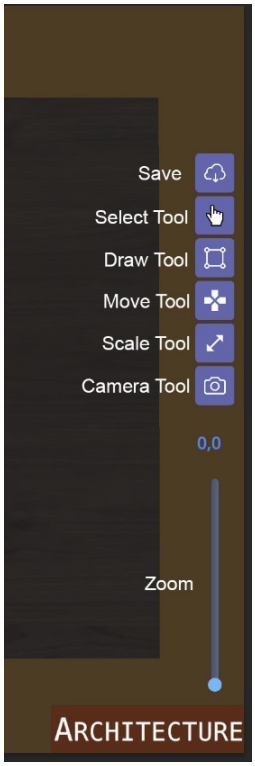


Abbildung 4.5: Bedeuetung der UI Elemente

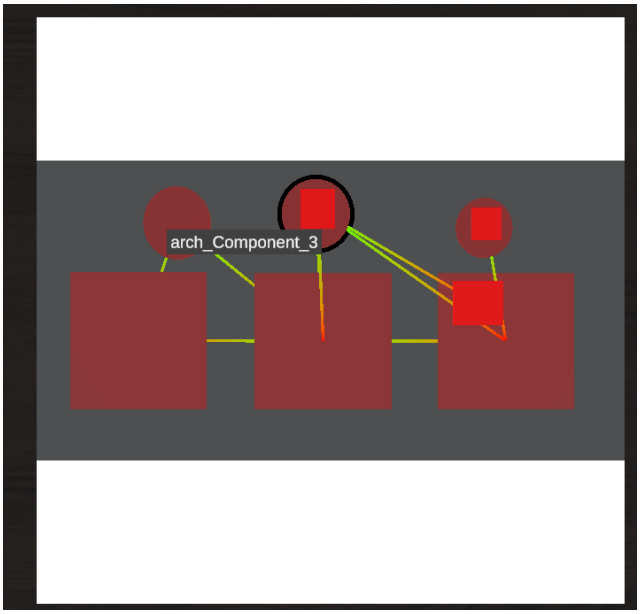


Abbildung 4.6: Label der Architekturkomponenten

4.3.3 Erzeugen von Architekturkomponenten

Architekturkomponenten können mithilfe des **Draw Tools** hinzugefügt werden. Als gültige Flächen für die Strichgesten gelten die Whiteboard-Zeichenfläche und die Flächen bereits existierender Knoten. Nachdem die Strichgesten wie in 4.2.2 beschrieben erzeugt und mit Hilfe des in 4.2.3 vorgestellten Algorithmus erfolgreich klassifiziert werden konnten, wird das Ergebnis in der Klasse **GestureHandler** mit der Methode **HandleGesture** ausgewertet und die dazugehörige Architekturkomponente generiert. Im Falle von Knoten, wird außerdem die Breite der gezeichneten Geste mittels zweidimensionaler BoundingBox errechnet und im nächsten Schritt auf die gerenderte Komponente angewandt.

Mithilfe von geraden Linien die zwischen zwei Knoten gezogen werden, können außerdem Abhängigkeiten generiert werden. Der zu erst berührte Knoten bildet dabei immer den Startknoten, der Knoten der zuletzt berührt wurde ist der Zielknoten.

4.3.4 Bearbeiten von Architekturkomponenten

Bewegen

Um Knoten bewegen zu können, muss der Nutzer über die Menüleiste zum **Move Tool** wechseln. Das Tool wird von der **MoveArchitectureAction** implementiert. Indem der Nutzer die Spitze des Stiftes auf den gewünschten Knoten setzt und anschließend den ersten Button am Stift gedrückt hält , kann anschließend über Bewegung des Stiftes die Position des Knoten verändert werden. Lässt der Nutzer den ersten Button am Stift los, so wird der Drag&Drop Modus beendet. Es besteht die Möglichkeit Knoten per Drag& Drop auf andere Knoten fallen zu lassen und sie so zu verschachteln. Um eine Verschachtelung rückgängig zu machen, können Knoten auch erneut auf eine andere Fläche fallen gelassen werden.

Zusätzlich besteht die Möglichkeit den anzuzeigenden Ausschnitt der Architektur zu verändern. Dazu kann der Nutzer den Stift auf eine Fläche ohne Knoten setzen und ähnlich zur Bewegung von Knoten per Drag&Drop den Auschnitt seinem Wunsch entsprechend anpassen.

Bei der Implementierung ist ein Bug im neuen Unity Input System aufgefallen, der dazu führt, dass der erste Button am Stift nicht vor anderen Interaktionen, wie z.b aufsetzen der Stiftspitze, gedrückt werden darf. Drückt der Nutzer den ersten Button dennoch bevor er den Stift auf dem Display aufsetzt, wird letzteres nicht mehr vom Input System aufgezeichnet.

Skalieren

Das **Scale Tool** erlaubt es einzelne Knoten in der Breite und\oder Höhe zu skalieren. Zur Auswahl des Knotens muss dieser vom Nutzer mit der Stiftspitze aus-

gewählt werden. Anschließen werden durch die Klasse **ScaleArchitectureAction** acht Ankerpunkte an den Rändern des Knotens erzeugt. Durch Drag&Drop der einzelnen Ankerpunkte kann der Nutzer anschließend die Größe des Knoten verändern.

Editieren & Löschen

Tippt der Nutzer nach Auswahl des **Select Tool** auf einen Knoten oder eine Kante, so öffnet sich ein Kontextmenü mit zusätzlichen Optionen. Für Knoten stehen die Aktionen Editieren und Löschen zur Verfügung. Im Falle von Editieren, öffnet sich ein Dialog zum Ändern des Knotennamen und des Knotentyps. Wählt der Nutzer die Löschaktion aus, so wird ein Bestätigungsdialog geöffnet, der eine Bestätigung erfordert um den Knoten zu löschen. Beim Bestätigen des Dialogs werden alle verbundenen Kanten und alle bisher untergeordneten Knoten mit gelöscht. Das Kontextmenü für Kanten stellt nur die Funktion zum Löschen zur Verfügung.

4.3.5 Laden & Speichern der Softwarearchitektur

Laden einer bestehenden Softwarearchitektur

Um eine bestehende Softwarearchitektur zu laden, muss der Nutzer seine GXL und die zugehörige Layoutdatei entweder im SLD oder GVL Format über die **SEECityArchitecture** Komponente auswählen und den Button Load Graph betätigen.

Speichern einer Softwarearchitektur

Änderungen an der Softwarearchitektur können vom Nutzer direkt aus dem Architekturmodus heraus gespeichert werden. Nachdem betätigen des Save Button, kann der Nutzer in einem Dialog den Namen der Architektur vergeben. Die Daten werden als GXL und SLD Dateien im Verzeichnis `StreamingAssets\Architecture\Output` abgelegt.

4.3.6 Zusätzliche Funktionen

Kamera

Der Nutzer kann das **Camera Tool** verwenden, um die Position der Kamera über der Zeichenfläche zu verändern. Darüber hinaus, besteht in jedem Tool die Möglichkeit über den slider das Zoomlevel der Architektur zu verändern und so einen detaillierteren Einblick zu gewinnen.

4.4 Probleme

Um das Grafiktablett überhaupt als Eingabegerät nutzen zu können, ist wie bereits in 4.2.1 erwähnt, das neue Input System von Unity nötig. Dieses ist erst seit

April 2020 als vollwertiges Release über den Unity Package Manager beziehbar. Während der Implementierung wurde ein Bug identifiziert, der dazu führt, dass das Drücken des Buttons an der Seite des Stiftes den kompletten State in Unity überschreibt und eine eventuell gedrückte Stiftspitze nicht mehr erkannt wird. Das hat besonders bei der Implementierung eines Drag&Drop Systems beim Bewegen der Architekturkomponenten zu Schwierigkeiten geführt.

Der $P-Algorithmus wurde außerdem um einen erwarteten Minimalwert für die Genauigkeit erweitert, so dass nur Ergebnisse mit einer Genauigkeit von mehr als 85% (> 0.85) als erfolgreich gewertet werden. Der Grund dafür ist die Ähnlichkeit von Strichgesten von Quadraten und Kreisen. Es kann andernfalls vermehrt zu fehlerhaften Klassifizierungen von Kreisen, Quadraten und anderen ähnlichen Formen kommen. Ebenso wird so verhindert, dass bei weniger als zwei vorhandenen Templates, auch komplett andere Gesten erfolgreich klassifiziert werden, aber nur einen Score von 10% (0.10) besitzen. Beispiel: Die Templates enthalten nur Daten für einen Kreis und der Benutzer zeichnet ein Dreieck. Dann würde das Dreieck als Kreis mit einem niedrigen Score erkannt werden.

Die Darstellung der Knoten basiert auf dem zugehörigen Typ. Für Knoten die über Strichgesten erzeugt oder über eine bestehende Architektur im GXL Format geladen werden, wird dies korrekt umgesetzt. Der Nutzer hat darüber hinaus allerdings die Möglichkeit über den Editierdialog den Typ der Komponente manuell zu ändern. In diesem Fall ändert sich die visuelle Darstellung der Komponente nicht. Aus Zeitgründen konnte dieses Problem nicht behoben werden.

Bei der Skalierung von Knoten kann es dazu kommen, das die Ankerpunkte so klein sind, das sie durch den Nutzer nicht mehr greifbar sind. In dem Fall kann das skalieren nur abgebrochen und neu gestartet werden. Hier wurde ebenfalls aus Zeitgründen noch keine Lösung für gefunden.

Zum Zeitpunkt des Schreibens dieses Abschnitts ist die Evaluation bereits abgeschlossen und die Arbeit kurz vor ihrer Abgabe. Während der Evaluation konnten keinerlei Komplikationen beim Erzeugen von Architekturkomponenten mit dem angezeigten Namens-Label festgestellt werden. Mittlerweile scheint die Performance der Anwendung aber soweit abgenommen zu haben, so dass die Bewegung des Labels immer ein wenig verzögert dargestellt wird. Das führt dazu, dass das Hinzufügen von Komponenten nicht zuverlässig klappt. Aus zeitlichen Gründen konnte dafür vor Abgabe keine Lösung mehr gefunden werden.

Kapitel 5

Evaluation

5.1 Planung

5.1.1 DECIDE-Framework

Zur Planung und Durchführung der Evaluation wird das *DECIDE*-Framework [27][28] verwendet. Das von **Preece, Rogers et al.** entwickelte Framework stellt eine sechsteilige Checkliste zur Verfügung, anhand derer sich zur Planung und Durchführung der Evaluation orientiert werden kann. Sie bietet besonders für unerfahrenere Studienleiter einen einfach zu befolgenden roten Faden.

- **D**etermine the overall *goals* that evaluation addresses
- **E**xplore the specific *questions* to be answered
- **C**hoose the *evaluation paradigm* and *techniques* to answer the questions.
- **I**dentify the *practical issues* that must be addressed, such as selecting participants
- **D**ecide how to deal with the *ethical issues*
- **E**valuate, interpret, and present the *data*.

Determine the overall goals that the evaluation addresses

Das Ziel der Evaluation ist es die SEE-Erweiterung hinsichtlich der Effizienz, Sicherheit und Effektivität mit Axivion Gravis zu vergleichen und deren Usability zu bewerten.

Explore the specific questions to be answered

Auf Basis der vorab definierten Ziele lassen sich Fragen formulieren, die durch die Evaluation beantwortet werden sollen:

- Modellieren Nutzer*innen Software-Architektur effizienter als mit Axivion Gravis?

- Modellieren Nutzer*innen Software-Architektur effektiver als mit Axivion Gravis?
- Modellieren Nutzer*innen Software-Architektur sicherer als mit Axivion Gravis?
- Empfinden Nutzer*innen eine höhere Usability als bei der Verwendung von Axivion Gravis?

Choose the evaluation paradigm and techniques to answer the questions

Die Probanden nehmen an einem Usability-Test unter kontrollierten Bedingungen teil. Die zu bearbeitenden Aufgaben entsprechen einem realistischen Nutzungsszenario und werden je in SEE oder Axivion Gravis bearbeitet. Zur Eliminierung eines Lerneffektes wird die Stichprobe der Probanden in zwei Gruppen unterteilt, die jeweils mit einem anderen Tool starten. Es gibt zwei Aufgaben A und B (siehe Anhang) die jeweils in zwei Teile unterteilt sind:

- **Teil 1** beinhaltet die Abbildung einer fiktiven Software-Architektur als UML-Paketdiagramm. Ziel ist es die jeweils abgebildete Architektur logisch und inhaltlich korrekt in dem vorliegenden Tool zu modellieren. Die punktgenaue Anordnung der einzelnen Architekturkomponenten ist zu vernachlässigen.
- **Teil 2** enthält eine Liste von Modifizierungsanweisungen, die auf das Ergebnis aus Teil 1 angewandt werden sollen.

Die Dauer, die zum Lösen der Aufgabe benötigt wurde, wird gemessen und anschließend für die Beurteilung der *Effizienz* des Tools genutzt. Nach Beendigung der Aufgabe wird die modellierte Architektur als *GXL* gespeichert und anschließend ausgewertet. Jede Architekturkomponente die falsch modelliert wurde oder gar fehlt, wird als Fehler gewertet. Mithilfe der so ermittelten Fehlerrate kann eine Aussage über die *Effektivität* des Tools getroffen werden. Zusätzlich werden jegliche Hilfestellungen, die zur Lösung der Aufgabe notwendig waren, vermerkt. Die *Sicherheit* des Tools kann anschließend aus der Anzahl der Hilfestellungen abgeleitet werden.

Um die letzte Frage beantworten zu können, wird der *System Usability Scale*(SUS)-Fragebogen verwendet, dem de-facto Standard zur Messung der allgemeinen Usability. [29]. Der Fragebogen besteht aus 10 alternierend positiv und negativ formulierten Aussagen, welche die Proband*innen auf einer Skala von 1 -”Stimme gar nicht zu” bis 5 -”Stimme voll zu” bewerten können. Aus den Bewertungen des*r Proband*in lässt sich anschließend ein numerischer Wert, der SUS-Score, errechnen. Seien U = Menge der ungeraden Aussagen und G = Menge der geraden Aussagen, so ist der SUS-Score definiert als: $(\sum_{u \in U} u - 1 + \sum_{g \in G} 5 - g) * 2,5$. Der so errechnete numerische SUS-Score kann mithilfe der Skala (siehe Abb. 5.1) von Bangor et al.[30] bewertet und verglichen werden.

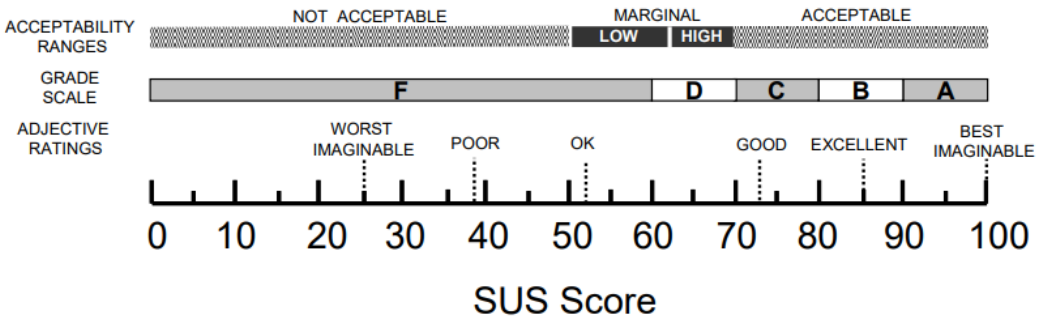


Abbildung 5.1: Bewertungsskala nach Bangor et al.[30]

Identify the practical issues

Die Evaluation ist aufgrund der herrschenden Covid-19 Pandemie von einigen Einschränkungen betroffen, die berücksichtigt werden müssen. Die Durchführung kann aufgrund des anhaltenden Notbetriebs der Universität nicht dort stattfinden. Es muss also auf die eigene Wohnung oder andere Alternativen ausgewichen werden. In Absprache mit meinem Arbeitgeber besteht die Möglichkeit die Evaluation in einem Konferenzraum vor Ort unter Einhaltung eines Hygiene-Konzepts durchzuführen. Die Wahl der Proband*innen beschränkt sich dadurch auf Arbeitskollegen, da der Zugang zum Büro aktuell nur Mitarbeitern gestattet ist. Sollten sich weitere externe Teilnehmer melden, so muss die Evaluation mit diesen in der eigenen Wohnung durchgeführt werden.

Das zu erfüllende Hygienekonzept beinhaltet folgende Punkte, die berücksichtigt werden müssen:

- Desinfizieren aller Eingabegeräte nach jedem Durchgang
- Ausreichende Belüftung der Räumlichkeiten durch Stoßlüften
- Tragen von Mund-Nase-Bedeckungen

Es muss berücksichtigt werden, dass nicht alle Arbeitskollegen bereit sind an der Evaluation nach Feierabend teilzunehmen, was dazu führt, dass einige der Evaluationen im Tagesgeschäft durchgeführt werden müssen. Die Evaluation wird daher mit einer maximalen Länge von 45 Minuten geplant.

Decide how to deal with the ethical issues

Die Zustimmung der Probanden zur Verarbeitung der gesammelten Daten wird zu Beginn der Evaluation in Form einer digitalen Einwilligungserklärung erfasst (siehe Abb. 5.2). Diese informiert die Teilnehmer über Umfang und Nutzung der Daten.

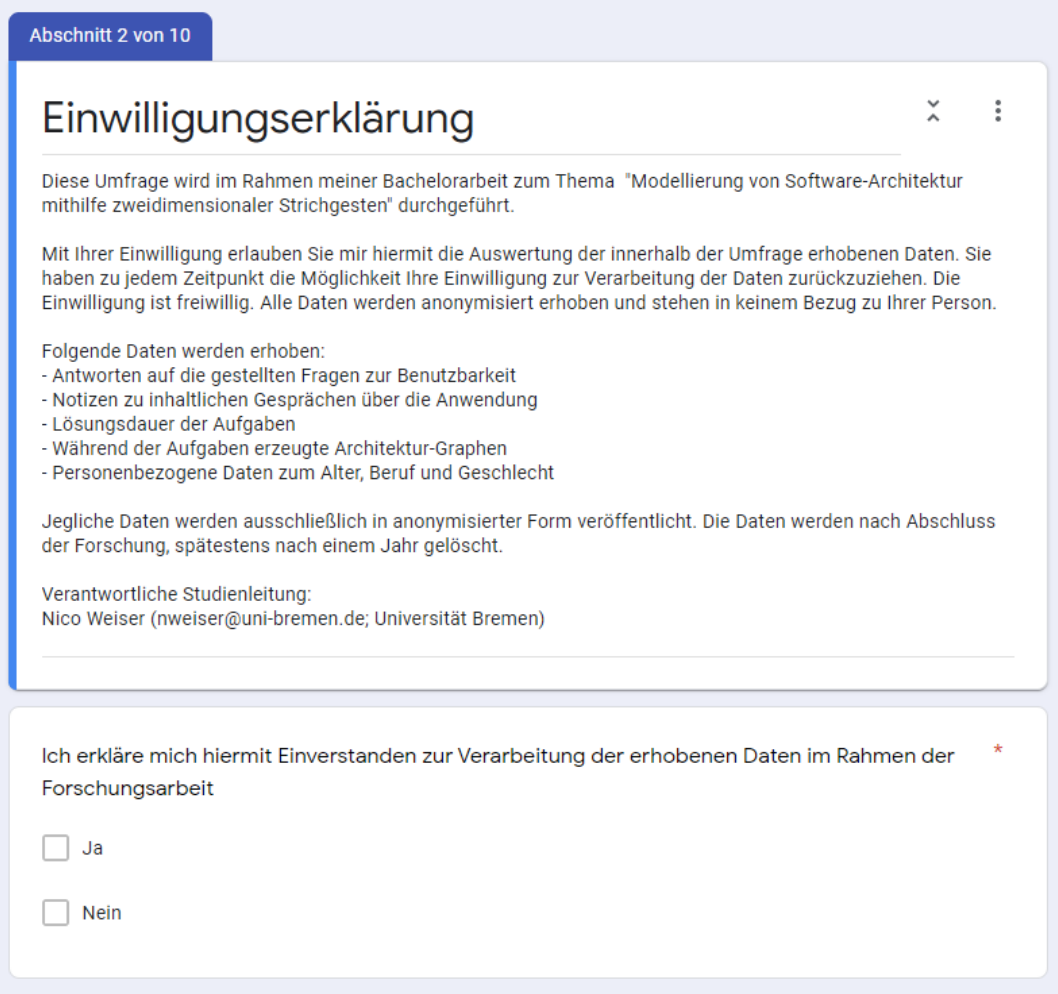


Abbildung 5.2: Digitale Einwilligungserklärung der Evaluation

Die Daten der Probanden werden innerhalb des Datensatzes ausschließlich anhand eines zufälligen Universal Unique Identfiers (UUID)[31] referenziert, so dass kein Rückschluss auf eine explizite Person getroffen werden kann.

Evaluate, interpret, and present the data

Im Folgenden werden die experimentellen Variablen der Evaluation erläutert:

- Die **unabhängigen Variablen** werden nicht von anderen Faktoren der Evaluation beeinflusst. Sie werden bewusst verändert, um die abhängigen Variablen des Evaluationsmodells zu manipulieren. Im Rahmen dieser Forschung entspricht die unabhängige Variable dem verwendeten Tool und hat zwei Ausprägungen: SEE und Axivion Gravis.
- Die **abhängigen Variablen** werden durch die Manipulation der unabhängigen Variablen des Evaluationsmodells beeinflusst und sind damit abhängig. Die abhängigen Variablen dieser Forschung sind die empfundene Usability der Probanden in Form des SUS-Score, die Bearbeitungsdauer der

Aufgaben, die Anzahl der vom Probanden gemachten Fehler und die Anzahl der benötigten Hilfestellungen.

Um die Beeinflussung der abhängigen Variablen durch äußere Faktoren zu minimieren muss darauf geachtet werden, dass alle Durchläufe unter gleichen Bedingungen stattfinden.

Die Wahl der statistischen Methode ist abhängig der zugrundeliegenden Wahrscheinlichkeitsverteilung der Stichprobe. Die Daten des Sample werden vorab auf Normalverteilung unter Verwendung des Shapiro-Wilk-Test geprüft. Die Hypothesen einer normalverteilten Stichprobe werden durch den t-Test für abhängige Stichproben überprüft. Ist die Stichprobe nicht normalverteilt wird der gepaarte Wilcoxon-Test verwendet.

5.2 Durchführung

5.2.1 Aufbau

Für die Evaluation wurde ein Lenovo ThinkPad mit einem Intel Core i5-10210U Prozessor, einer integrierten Intel UHD Grafikkarte, 16 GB Arbeitsspeicher und Windows 10 genutzt. Zusätzlich wurden Maus und Tastatur für die Nutzung von Axivion Gravis bereitgestellt. Für SEE stand das in Kapitel 2.6 vorgestellte Wacom Cintiq 16 Stift-Display mit dem Wacom Pro Pen Slim zur Verfügung. Als Monitor wurde für beide Tools das Stift-Display genutzt.

5.2.2 Ablauf

Zu Beginn wurde der Proband gebeten der Verarbeitung und anonymisierten Veröffentlichung der erhobenen Daten in Form einer digitalen Einwilligungserklärung zuzustimmen (siehe Abb. 5.2). Sie informiert den Teilnehmer über Art und Menge der gesammelten Daten und räumt ihm das Recht ein, zu jedem Zeitpunkt der Verarbeitung zu widersprechen. Anschließend wurden über die digitale Umfrage demografische Fragen zu Alter, Geschlecht und Job und eine Frage zu Vorkenntnissen mit Software-Architektur Modellierungstools gestellt. Um zu gewährleisten, dass alle Teilnehmer das gleiche Wissen über Software-Architektur und die verwendete Darstellungsform der Architektur verfügen, erhielten sie einen Text der die Punkte in Kürze beschreibt (siehe Abb. 5.3). In den nächsten Schritten wurden die Aufgaben in den nacheinander in SEE und Axivion Gravis bearbeitet. Die Reihenfolge der verwendeten Tools war abhängig der dem Probanden zugeordneten Gruppe. Für jedes Tool erhielt der Proband eine Einweisung in die Steuerung und 5 Minuten Zeit um sich selbständig mit dem auseinander setzen zu können. Wurden alle offenen Fragen geklärt konnte mit der Bearbeitung der Aufgaben begonnen werden. Aufgabe A wurde mit Tool 1 und Aufgabe 2 mit Tool 2 bearbeitet. War der Teilnehmer der Meinung, die Aufgabe vollständig bearbeitet zu haben, so konnte er diese abgeben. Anschließend wurde der digitale SUS-Fragebogen für das verwendete Tool ausgefüllt. Nach

der Bearbeitung beider Aufgaben und dem Ausfüllen der SUS-Fragebögen wurden die Teilnehmer nach ihrem präferierten Tool und den Gründen ihrer Wahl gefragt. Abschließend konnten noch Anmerkungen und Verbesserungsvorschläge genannt werden.

5.3 Ergebnisse

Im folgenden Abschnitt werden die im Rahmen der Evaluation erhobenen Daten präsentiert. Es wird mit den Teilnehmern und einigen demografischen Daten begonnen. Darauf folgen die quantitativen Daten und zum Schluss die gesammelten qualitativen Daten

5.3.1 Teilnehmer

Es haben an der Evaluation insgesamt 10 Probanden teilgenommen davon waren acht Arbeitskollegen und zwei Personen aus dem Freundeskreis. Es haben acht Probanden und zwei Probandinnen teilgenommen. Sechs Probanden hatten bereits Erfahrung mit graph-basierten Modellierungstools. Das Durchschnittsalter lag bei 27,6 Jahren mit einer Standardabweichung von 1,46 Jahren. Acht der Probanden waren Software-Developer, einer Software-Architekt und einer Projektmanager.

5.3.2 Quantitative Daten

Die durchschnittliche Bearbeitungszeit der Aufgaben in SEE betrug 579,7 Sekunden. Neun Teilnehmer bearbeiteten die Aufgaben fehlerfrei, ein Teilnehmer hat drei Fehler gemacht. Vier Teilnehmer benötigten keinerlei Hilfe bei der Bearbeitung der Aufgaben, drei Teilnehmer haben ein Mal Hilfe in Anspruch genommen, ein Teilnehmer zwei Mal, ein Teilnehmer drei Mal und ein Teilnehmer vier Mal. Der durchschnittliche SUS-Score ergab 68 Punkte und entspricht damit einer akzeptablen Usability[30].

In Axivion Gravis betrug die durchschnittliche Bearbeitungszeit der Aufgaben 527,1 Sekunden. Acht Teilnehmer begingen keinen Fehler bei der Bearbeitung. Ein Teilnehmer machte einen Fehler und einer acht Fehler. Fünf Teilnehmer haben keine Hilfe in Anspruch genommen Zwei Teilnehmer haben ein Mal Hilfe benötigt, zwei nahmen zwei Mal und einer drei Mal Hilfe in Anspruch. Der durschnittliche SUS-Score ergab 52,75 Punkte und entspricht damit einer gerade noch akzeptablen Usability[30].

Betrachtet man die Daten fallen eindeutig die extremen Ausreißer auf. Die Maxima aller gemessenen Werte stammen von einer einzigen Person. Bei jenem Probanden war es der Fall, dass für die Evaluation nur genau 45 Minuten bis zu einem Nachfolgetermin zur Verfügung standen. Dadurch war der Proband

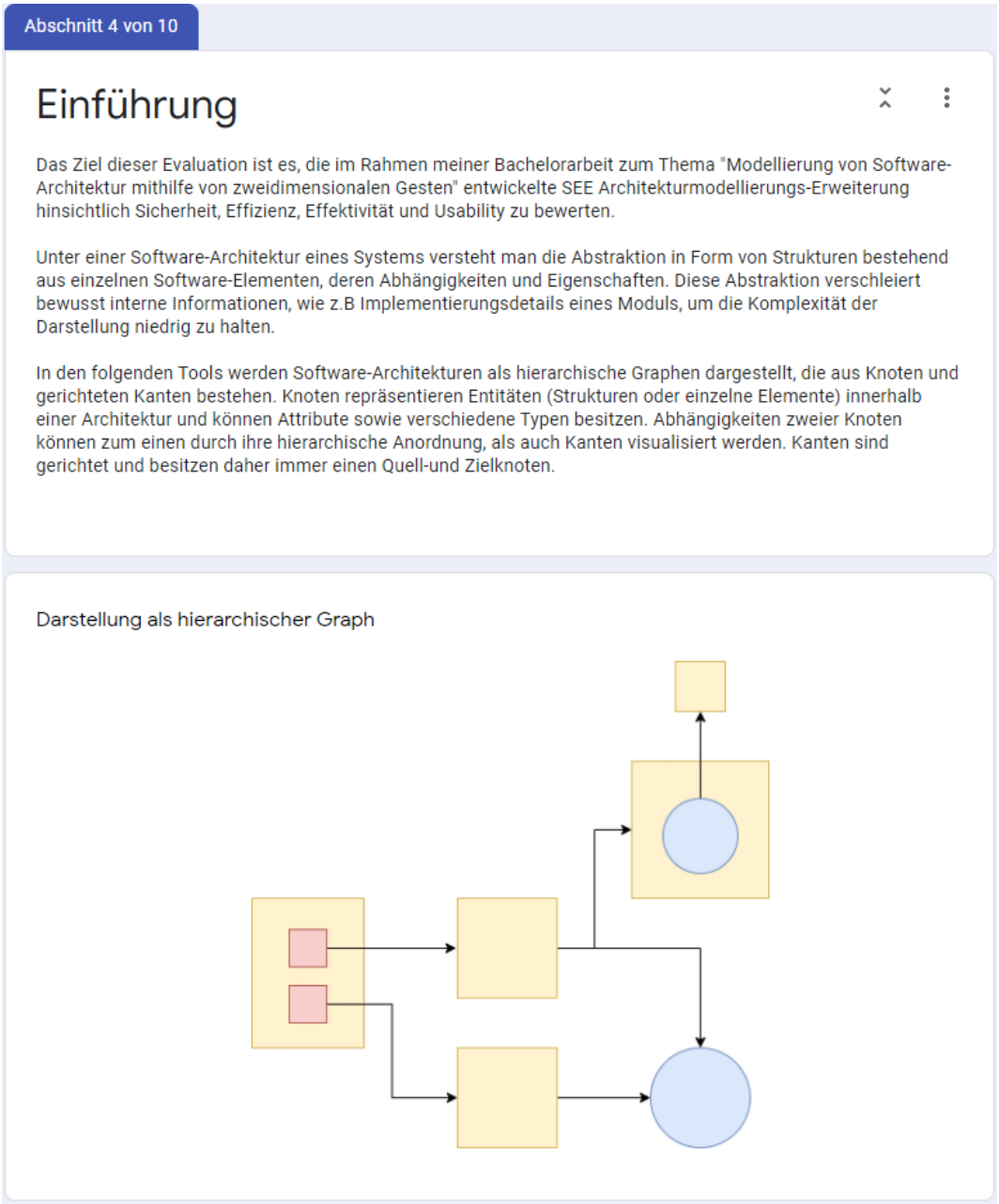


Abbildung 5.3: Einführung in die Thematik

unkonzentrierter und gestresster als in einem üblichen Szenario. Da diese Daten nicht in den geplanten Rahmenbedingungen erhoben wurden und extreme Ausreißer darstellen, werden diese für die weitere Auswertung nicht berücksichtigt. Tabelle 5.1 zeigt die bereinigten Messergebnisse.

	Mittelwert SEE	Standardabweichung SEE
Frage 1	3,44	0,24
Frage 2	2,33	0,44
Frage 3	3,88	0,26
Frage 4	2,22	0,43
Frage 5	3,33	0,37
Frage 6	2,88	0,45
Frage 7	4,44	0,18
Frage 8	2,55	0,50
Frage 9	4,22	0,22
Frage 10	1,88	0,38
SUS-Score	68,61	14,04

	Mittelwert Gravis	Standardabweichung Gravis
Frage 1	3,0	0,40
Frage 2	3,44	0,34
Frage 3	3,11	0,26
Frage 4	2,11	0,39
Frage 5	2,78	0,28
Frage 6	2,78	0,15
Frage 7	3,67	0,33
Frage 8	3,22	0,28
Frage 9	4,0	0,24
Frage 10	3,0	0,5
SUS-Score	51,25	13,91

Zeit	564,1	19,41
Fehler	0	0
Hilfe	0,88	0,351

Zeit	519	14,02
Fehler	0,11	0,11
Hilfe	0,78	0,36

Tabelle 5.1: Bereinigte Messergebnisse

Anhand der bereinigten Daten aus Tabelle 5.1 ergibt sich für SEE weiterhin eine akzeptable Usability mit 68,61 Punkten[30]. Axivion Gravis fällt mit 51,25 Punkten auf eine schlechte Usability[30] herab. Abbildung 5.4 zeigt noch einmal den direkten Vergleich der SUS-Fragebogen Antworten als Plot. Der Vergleich der Bearbeitungszeit der Probanden ist in Abbildung 5.5 zu sehen.

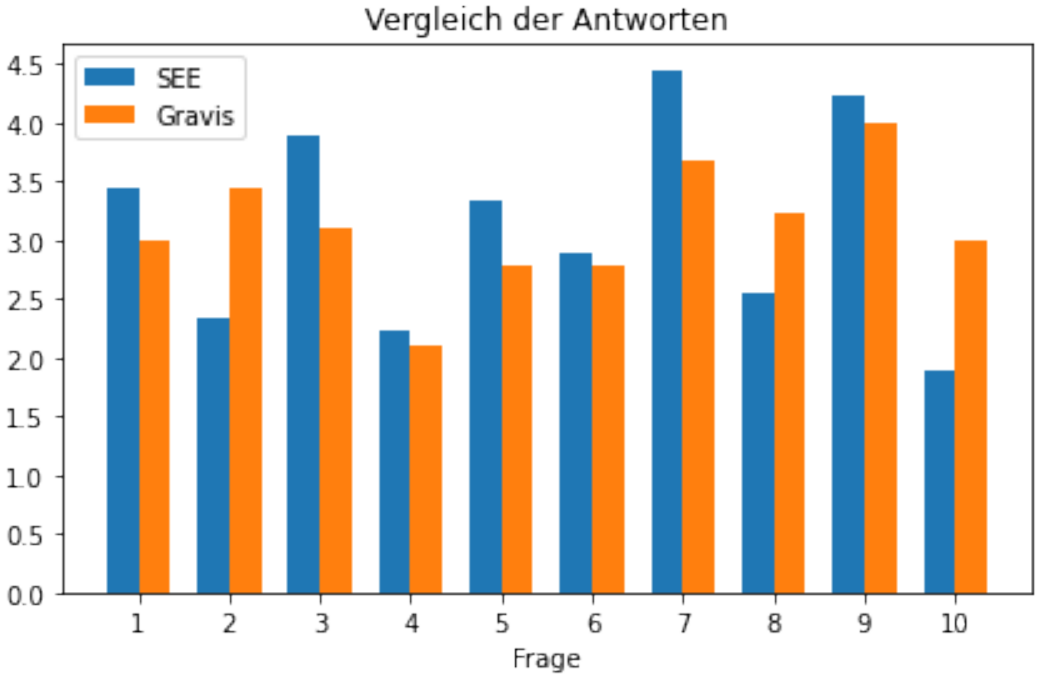


Abbildung 5.4: Vergleich der Antworten des SUS-Fragebogen

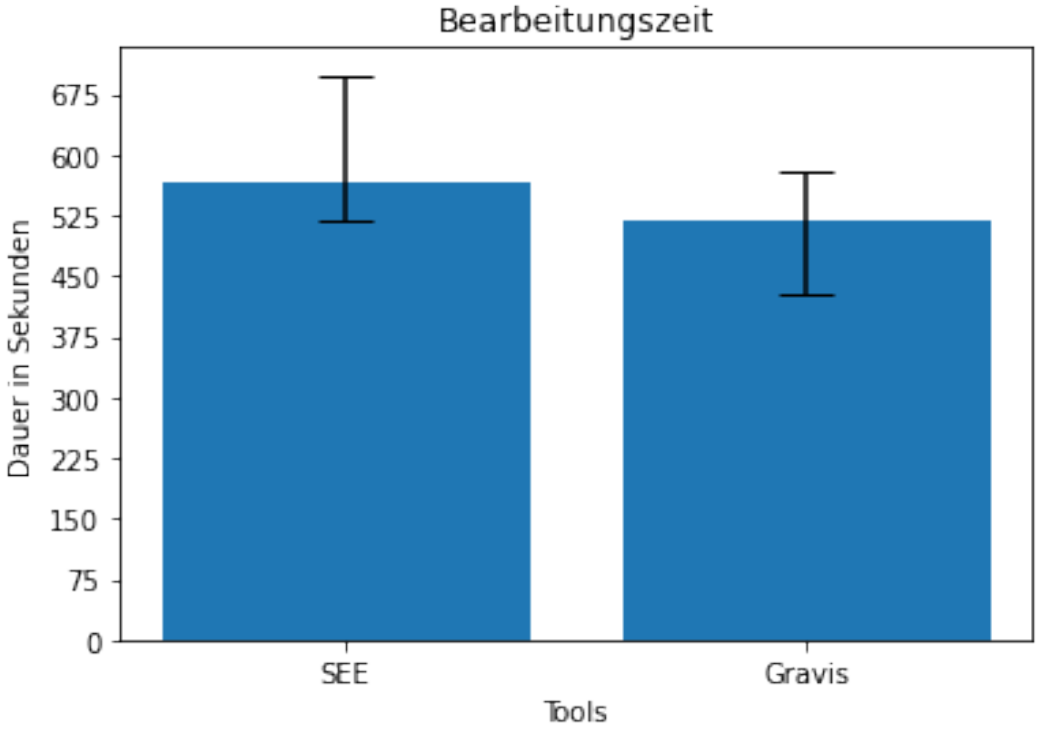


Abbildung 5.5: Vergleich der Bearbeitungszeit je Tool

5.3.3 Qualitatives Feedback

Nach der Bearbeitung der Aufgaben wurden die Probanden gefragt welches Tool ihnen besser gefallen hat und warum sie sich so entschieden haben. Acht der 10 Probanden gaben an, dass Ihnen die SEE Erweiterung besser gefallen hat, zwei Teilnehmer wählten Axivion Gravis. Im Folgenden werden die von den Probanden genannten Gründe gegenübergestellt:

- SEE
  - Moderneres und übersichtlicheres UX-Design
  - Das ”Malen” von Komponenten macht Spaß
  - Die Nutzung fühlt sich intuitiver an
  - Einfache Auswahl von Funktionen
- Axivion Gravis
  - Einfachere Bedienung
  - Icons und Namen haben die Orientierung erleichtert

Die Teilnehmer konnten außerdem eigene Verbesserungsvorschläge und Anmerkungen nennen, welche im Folgenden aufgelistet wurden:

Kritik

- SEE
  - Der Name sollte immer auf den Knoten dargestellt werden
  - Die Eingabe der Namen ist umständlich
  - Es ist nicht sichtbar welches Element ausgewählt wurde
  - Die Eingabe der Namen ist umständlich
  - Anwendungsfeedback beim Drag&Drop
  - Fehlende Zurück-Funktion
- Gravis
  - Unübersichtliches UX-Design
  - Das verschachteln von Knoten ist unintuitiv

SEE Verbesserungsvorschläge

- TreeMap-Ansicht aller Architekturelemente
- Kopierfunktion für Knoten
- Multi-Select und multi-Edit von Elementen
- Radialmenü für die Auswahl der Funktion
- Darstellung von Kanten als Pfeile

5.4 Auswertung

5.4.1 Hypothesen

Für die Auswertung der Messergebnisse werden die in Kapitel 5.1.1 aufgestellten Leitfragen als Hypothesen umformuliert.

Modellieren Nutzer Software-Architektur effizienter als mit Axivion Gravis?

- **H0:** Es gibt keinen Unterschied in der Effizienz der Modellierung zwischen SEE und Axivion Gravis
- **H1:** Es gibt einen Unterschied in der Effizienz der Modellierung zwischen SEE und Axivion Gravis

Modellieren Nutzer Software-Architektur effektiver als mit Axivion Gravis?

- **H0:** Es gibt keinen Unterschied in der Effektivität der Modellierung zwischen SEE und Axivion Gravis
- **H1:** Es gibt einen Unterschied in der Effektivität der Modellierung zwischen SEE und Axivion Gravis

Modellieren Nutzer Software-Architektur sicherer als mit Axivion Gravis?

- **H0:** Es gibt keinen Unterschied in der Sicherheit der Modellierung zwischen SEE und Axivion Gravis
- **H1:** Es gibt einen Unterschied in der Sicherheit der Modellierung zwischen SEE und Axivion Gravis

Empfinden Nutzer eine höhere Usability als bei der Verwendung von Axivion Gravis?

- **H0:** Es gibt keinen Unterschied in der empfundenen Usability der Modellierung zwischen SEE und Axivion Gravis
- **H1:** Es gibt einen Unterschied in er empfundenen Usability der Modellierung zwischen SEE und Axivion Gravis

5.4.2 Quantitative Daten

Hypothese 1

- **H0:** Es gibt keinen Unterschied in der Effizienz der Modellierung zwischen SEE und Axivion Gravis
- **H1:** Es gibt einen Unterschied in der Effizienz der Modellierung zwischen SEE und Axivion Gravis

Statistik	Signifikanz p
0,885	0,031

Tabelle 5.2: Effizienz: Shapiro-Wilk-Test auf Normalverteilung

Gesamtzahl	9
Teststatistik	4
Standardfehler	12,840
Standardisierte Teststatistik	-2,194
Asymptotische Sig. (zweiseitiger Test)	0,028

Tabelle 5.3: Effizienz: Wilcoxon-Test

Die Effizienz war gemäß Shapiro-Wilk-Test (siehe Tabelle 5.2) nicht normalverteilt, $p < 0,05$. Für die Prüfung der Hypothesen wird daher der gepaarte Wilcoxon-Test angewandt. Es konnte ein signifikanter Unterschied zwischen der Effizienz der Modellierung in SEE und Axivion Gravis festgestellt werden, demnach wird die Nullhypothese verworfen. Die Effizienz der Modellierung in Axivion Gravis (Median = 522) ist signifikant höher als in SEE (Median = 541), $z = -2,19, p < 0,05$.

Hypothese 2

- **H0:** Es gibt keinen Unterschied in der Effektivität der Modellierung zwischen SEE und Axivion Gravis
- **H1:** Es gibt einen Unterschied in der Effektivität der Modellierung zwischen SEE und Axivion Gravis

Statistik	Signifikanz p
0,252	0,000

Tabelle 5.4: Effektivität: Shapiro-Wilk-Test auf Normalverteilung

Gesamtzahl	9
Teststatistik	0
Standardfehler	0,317
Standardisierte Teststatistik	-1
Asymptotische Sig. (zweiseitiger Test)	1

Tabelle 5.5: Effektivität: Wilcoxon-Test

Gemäß Shapiro-Wilk-Test (siehe Tabelle 5.4) ist die Effektivität nicht normalverteilt, weswegen der gepaarte Wilcoxon-Test für die Prüfung der Hypothese verwendet wird. Es konnte mit dem gepaarten Wilcoxon-Test kein signifikanter Unterschied zwischen der Effektivität in SEE und Axivion Gravis gefunden werden, daher wird die Nullhypothese beibehalten, $z = -1, p \geq 0,05$.

Hypothese 3

- **H0:** Es gibt keinen Unterschied in der Sicherheit der Modellierung zwischen SEE und Axivion Gravis
- **H1:** Es gibt einen Unterschied in der Sicherheit der Modellierung zwischen SEE und Axivion Gravis

Statistik	Signifikanz p
0,774	0,000

Tabelle 5.6: Sicherheit: Shapiro-Wilk-Test auf Normalverteilung

Gesamtzahl	9
Teststatistik	17
Standardfehler	0,246
Standardisierte Teststatistik	-0,14
Asymptotische Sig. (zweiseitiger Test)	0,943

Tabelle 5.7: Sicherheit: Wilcoxon-Test

Der Shapiro-Wilk-Test (siehe Tabelle 5.6) zeigt, dass die Sicherheit nicht normalverteilt ist. Aus diesem Grund wird der gepaarte Wilcoxon-Test verwendet. Anhand des gepaarten Wilcoxon-Test konnte kein signifikanter Unterschied zwischen der Sicherheit in SEE und Axivion Gravis festgestellt werden, daher wird die Nullhypothese beibehalten, $z = -0,14, p \geq 0,05$.

Hypothese 4

- **H0:** Es gibt keinen Unterschied in der empfundenen Usability der Modellierung zwischen SEE und Axivion Gravis
- **H1:** Es gibt einen Unterschied in er empfundenen Usability der Modellierung zwischen SEE und Axivion Gravis

	Gepaarte Differenzen							
				95% Konfidenzintervall der Differenz				
	Mittelwert	Standardabweichung	Standardfehler des Mittelwertes	Untere	Obere	T	df	Sig. (2-seitig)
SEE - Axivion Gravis	13,61111	21,36407	7,12136	-2,81077	30,03299	1,911	8	0,092

Tabelle 5.8: SUS-Score: gepaarter t-Test

Anhand des Shapiro-Wilk-Test (siehe Tabelle 5.6) erkennt man, dass der SUS-Score normalverteilt ist, weshalb zur Überprüfung der Hypothese der gepaarte t-Test verwendet wird. Mithilfe des t-Test konnte keine signifikante Differenz der SUS-Score Mittelwerte ermittelt werden, es wird daher die Nullhypothese angenommen und die Alternativhypothese verworfen.

5.4.3 Qualitatives Feedback

Der am häufigsten genannte Kritikpunkt bei SEE war das Fehlen von Namen auf den Knoten. Das führte dazu, dass ein anhand des Namen gesuchter Knoten nicht auf einen Blick erkennbar war und die Probanden sich neu orientieren mussten. Die Probanden empfanden außerdem die Eingabe der Namen umständlich, da durch das fehlende automatische Fokussieren des Inputfeldes nicht direkt mit der Eingabe des Namens begonnen werden konnte. Ebenso häufig wurde kritisiert, dass nicht eindeutig war welches Element per Selektion ausgewählt wurde. Einige Teilnehmer merkten zudem an, dass ihnen eine Möglichkeit zum rückgängig machen von Änderungen fehlte.

Die Probanden kritisierten an Axivion Gravis das ihrem Eindruck nach alte UX-Design und empfanden dies als zu komplex und unübersichtlich. Außerdem empfanden alle Teilnehmer die Notwendigkeit zur Nutzung des *Hierarchy Tools* zum Verschachteln von Knoten, als äußerst unintuitiv.

Besonders hervorgehoben wurde die User-Experience beim Erzeugen von Architekturelementen in SEE. Eine häufige Aussage der Teilnehmer war, dass das

Erzeugen durch die Strichgesten ”Spaß” mache. Ebenso wurde die visuelle Darstellung und das übersichtlichere UX-Design gelobt.

5.4.4 Ergebnis der Evaluation

Die Auswertung der quantitativen Daten konnte keinen signifikanten Unterschied zwischen der Effektivität, Sicherheit und empfundenen Usability feststellen. Im Hinblick auf die Effizienz konnte jedoch gezeigt werden, dass Nutzer mit Axivion Gravis schneller Softwarearchitektur modellieren. Die Anmerkungen und Kritikpunkte der Probanden bestätigen dieses Ergebnis, ebenfalls. Dennoch haben 7 von 9 Teilnehmern angegeben, dass sie SEE aufgrund der intuitiveren Benutzung bevorzugen würden.

Kapitel 6

Fazit und Ausblick

6.1 Fazit

Die im Kontext dieser Arbeit entstandene Möglichkeit zur Modellierung von Software-Architektur mithilfe zweidimensionaler Strichgesten wurde erfolgreich in das SEE Projekt integriert. Auch wenn die Mehrzahl der Probanden sich dafür ausgesprochen hat, dass sie SEE aufgrund der intuitiven Bedienung bevorzugen würden, befriedigt das Ergebnis der Evaluation meine persönlichen Erwartungen nicht. Rückblickend erachte ich besonders die Entscheidung die Namen der Knoten nicht direkt auf den Objekten, sondern nur per Hover als Label anzeigen zu lassen, als großen Fehler. Ebenso die fehlende Möglichkeit Aktionen wieder rückgängig zu machen. Dies hätte besonders im Bezug auf die Sicherheit der Verwendung ein deutliches Improvement ermöglicht. Für eine zukünftige Verbesserung der entwickelten Erweiterung, sehe ich diese Aspekte mit der höchsten Priorität.

6.2 Ausblick

Im folgenden Abschnitt werden einige Verbesserungsvorschläge und Funktionen erläutert, die im Rahmen der Bearbeitung dieser Arbeit aufgekommen sind und für zukünftige Weiterentwicklung von Interesse sein könnten.

6.2.1 Radial-Menü

Aktuell müssen Nutzer jeden Toolwechsel manuell über die Menüleiste an der Seite durchführen. Bei vielen kleinen Anpassungen die häufige Toolwechsel mit sich ziehen, wird viel Zeit für die Bewegung hin und zurück zum Menü verloren. Inspiriert von dem Wacom eigenen Expressmenü für Grafiktabletts (siehe Abb. 6.1), könnte man ein ähnliches Menü für den Wechsel des aktiven Tools in SEE implementieren.

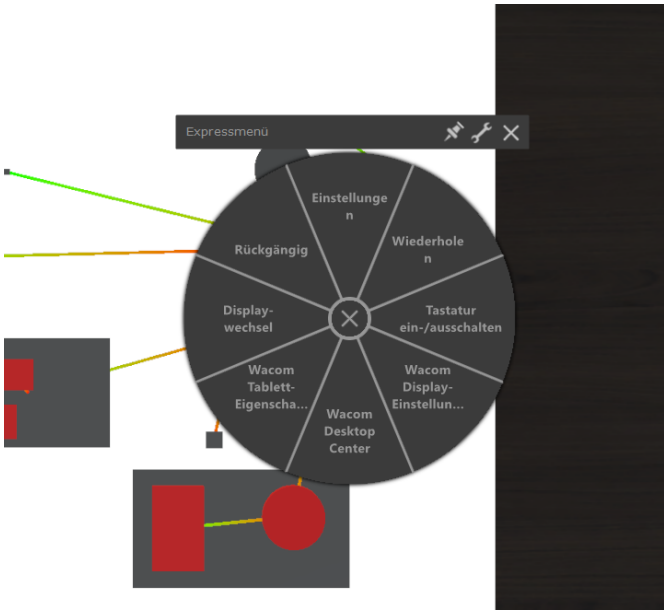


Abbildung 6.1: Wacom Expressmenu

6.2.2 Onscreen Tastatur

Für die Eingabe von Werten in Dialogen ist bisher noch eine Tastatur nötig. Bei jedem Dialog muss der Nutzer somit den Stift aus der Hand legen und auf die Tastatur wechseln. Eine OnScreen Tastatur könnte die Effizienz der Nutzer immens steigern.

Quell und Literaturverzeichnis

[1] OMG. (). “Unified Modelling Langauge (UML),” [Online]. Available: https://www.omg.org/spec/UML/About-UML/ (visited on 06/05/2021).

[2] UMLet. (). “Free UML Tool for Fast UML Diagrams,” [Online]. Available: https://www.umlet.com/ (visited on 08/05/2021).

[3] ArgoUml. (). “ArgoUml,” [Online]. Available: https://github.com/argouml-tigris-org (visited on 08/05/2021).

[4] R. Koschke, “Incremental reflexion analysis,” in *2010 14th European Conference on Software Maintenance and Reengineering*, 2010, pp. 1–10.

[5] L. Bass, P. Clements, and R. Kazman, *Software Architecture in Practice*, 4th ed. Addison-Wesley, 2021.

[6] Unity Technologies. (). “Unity Engine - Real-Time Development Platform,” [Online]. Available: https://unity.com/de (visited on 12/01/2020).

[7] R. Holt, A. Schürr, S. E. Sim, and A. Winter. (Feb. 15, 2008). “GXL - Graph eXchange Language,” [Online]. Available: https://userpages.uni-koblenz.de/~ist/GXL/index.php (visited on 03/07/2021).

[8] E. International. (). “ECMA-404 - The JSON data interchange syntax,” [Online]. Available: https://www.ecma-international.org/publications-and-standards/standards/ecma-404/ (visited on 06/30/2021).

[9] JSON.org. (). “Einführung in JSON,” [Online]. Available: https://www.json.org/json-de.html (visited on 06/30/2021).

[10] Wacom. (). “Wacom Cintiq 16,” [Online]. Available: https://www.wacom.com/de-de/products/pen-displays/wacom-cintiq (visited on 12/01/2020).

[11] ——, (). “Technologien - Elektromagnetische Resonanz,” [Online]. Available: https://www.wacom.com/de-de/for-business/technologies/emr (visited on 05/05/2021).

[12] ——, (). “How our pen work,” [Online]. Available: https://developer.wacom.com/sitecore/content/wacomdotcom/home/enterprise/business-solutions/resources-and-information/emr-benefits (visited on 07/01/2021).

[13] Axivion. (). “Axivion,” [Online]. Available: https://www.axivion.com/produkte/axivion-suite/ (visited on 08/06/2021).

[14] M. Magin and S. Kopf, “A collaborative multi-touch uml design tool,” Feb. 2013.

[15] J. O. Wobbrock, A. D. Wilson, and Y. Li, “Gestures without libraries, toolkits or training: A $1 recognizer for user interface prototypes,” in *Proceedings of the 20th Annual ACM Symposium on User Interface Software and Technology*, ser. UIST ’07, Newport, Rhode Island, USA: Association for Computing Machinery, 2007, pp. 159–168. [Online]. Available: https://doi.org/10.1145/1294211.1294238.

[16] K. Sugiyama, S. Tagawa, and M. Toda, “Methods for visual understanding of hierarchical system structures,” *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 11, no. 2, pp. 109–125, 1981.

[17] H. Eichelberger, “Aesthetics and automatic layout of uml class diagrams,” doctoralthesis, Universität Würzburg, 2005.

[18] R. Oberhauser, “Vr-uml: The unified modeling language in virtual reality – an immersive modeling experience,” in *Business Modeling and Software Design*, B. Shishkov, Ed., Cham: Springer International Publishing, 2021, pp. 40–58.

[19] R. Oberhauser and C. Pogolski, “Vr-ea: Virtual reality visualization of enterprise architecture models with archimate and bpmn,” in *Business Modeling and Software Design*, B. Shishkov, Ed., Cham: Springer International Publishing, 2019, pp. 170–187.

[20] K. Döhl, “Modellierung einer Softwarearchitektur in Virtual Reality mithilfe der Leap Motion,” 2020.

[21] Ultraleap. (). “Leap Motion Controller,” [Online]. Available: https://www.overleaf.com/project/5f86121b2666e80001d327a6 (visited on 07/15/2021).

[22] (Jan. 22, 2021). “Input system | input system | 1.0.2,” [Online]. Available: https://docs.unity3d.com/Packages/com.unity.inputsystem@1.0/manual/index.html (visited on 10/27/2021).

[23] (Oct. 12, 2021). “Impact of $-family,” [Online]. Available: http://depts.washington.edu/acelab/proj/dollar/impact.html (visited on 10/27/2021).

[24] J. O. Wobbrock, A. D. Wilson, and Y. Li, “Gestures without libraries, toolkits or training: A $1 recognizer for user interface prototypes,” in *Proceedings of the 20th Annual ACM Symposium on User Interface Software and Technology*, ser. UIST ’07, Newport, Rhode Island, USA: Association for Computing Machinery, 2007, pp. 159–168. [Online]. Available: https://doi.org/10.1145/1294211.1294238.

[25] L. Anthony and J. O. Wobbrock, “A lightweight multistroke recognizer for user interface prototypes,” in *Proceedings of Graphics Interface 2010*, 2010, pp. 245–252.

[26] R.-D. Vatavu, L. Anthony, and J. O. Wobbrock, “Gestures as point clouds: A $precognizer for user interface prototypes$,” in *Proceedings of the 14th ACM international conference on Multimodal interaction*, 2012, pp. 273–280.

[27] J. Preece, Y. Rogers, and H. Sharp, *Interaction design: beyond human-computer interaction*. Wiley, 2002.

[28] R. Malaka and D. Wenig, *Mensch Technik Interaktion - Kapitel: Evaluation*, 2017.

[29] J. R. Lewis, "The system usability scale: Past, present, and future," *International Journal of Human–Computer Interaction*, vol. 34, no. 7, pp. 577–590, 2018. eprint: https://doi.org/10.1080/10447318.2018.1455307. [Online]. Available: https://doi.org/10.1080/10447318.2018.1455307.

[30] A. Bangor, P. Kortum, and J. Miller, "Determining what individual sus scores mean: Adding an adjective rating scale," *Journal of usability studies*, vol. 4, no. 3, pp. 114–123, 2009.

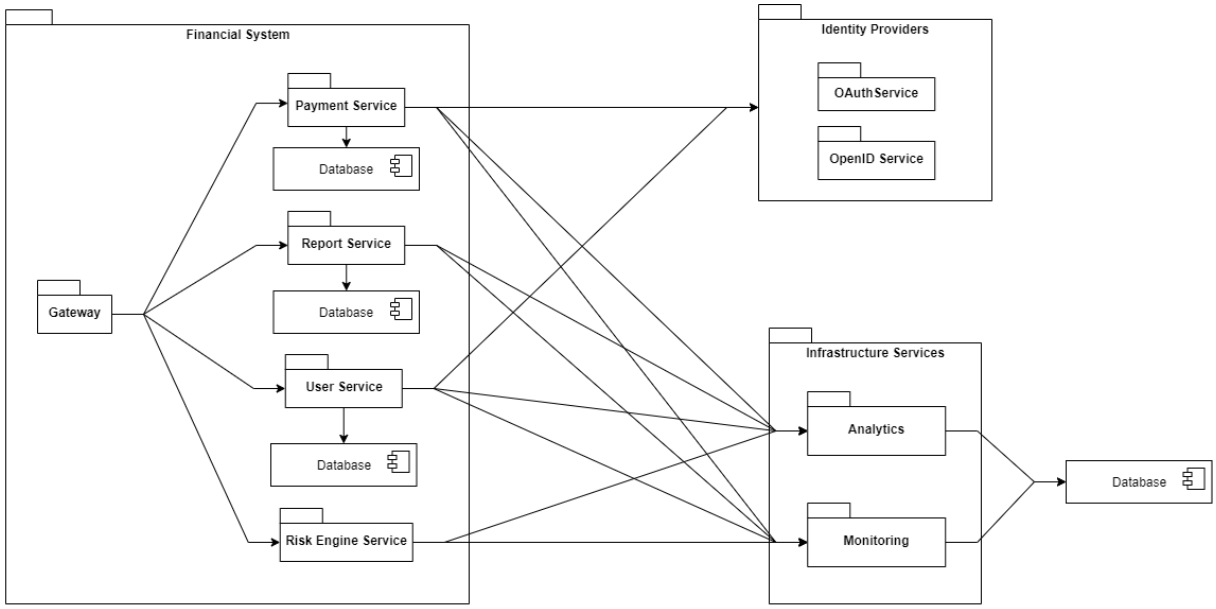
[31] T. O. Group. (). "Universal Unique Identifier," [Online]. Available: https://pubs.opengroup.org/onlinepubs/9629399/apdxa.htm (visited on 08/02/2021).

Anhang

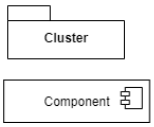
Aufgabe A

Teil 1)

Im Folgenden ist eine fiktive Architektur einer Finanz-Software als UML-Paketdiagramm abgebildet. Das Ziel der Aufgabe ist es, die dargestellte Architektur im bereitgestellten Tool zu modellieren.



Legende



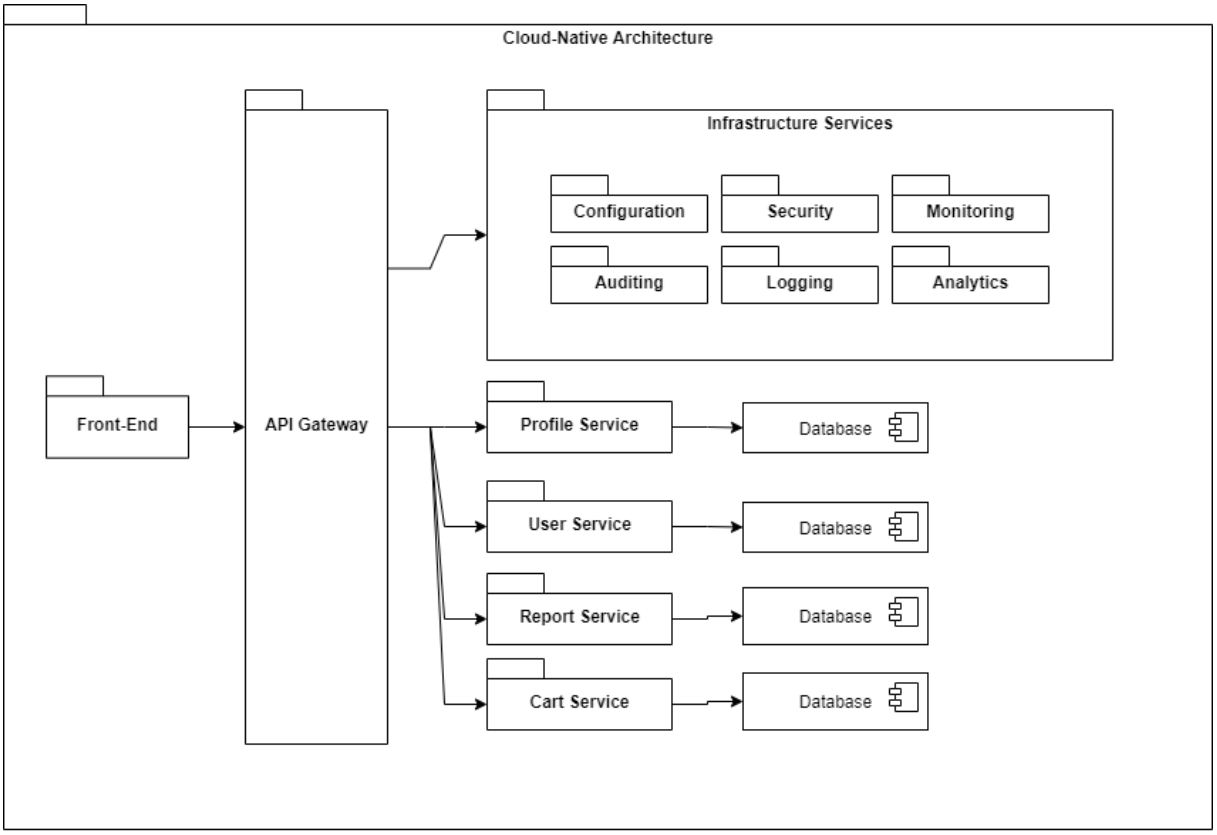
Teil 2)

1. Erzeuge einen Knoten **Database Cluster** vom Typ *Cluster*
2. Verschiebe alle Knoten Database vom Typ **Component** in den Knoten **Database Cluster**
3. Entferne die ausgehende Kante vom Knoten Analytics zum Knoten Database
4. Entferne die ausgehende Kante vom Knoten Monitoring zum Knoten Database
5. Erzeuge eine Kante vom Knoten **Infrastructure Services** zum Knoten **Database Cluster**
6. Erzeuge eine Kante vom Knoten **Identity Providers** zum Knoten **Database Cluster**

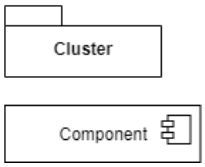
Aufgabe B

Teil 1)

Im Folgenden ist eine fiktive Architektur nach Vorbild einer Cloud-Native Microservice Architektur[1] als UML-Paketdiagramm abgebildet. Das Ziel der Aufgabe ist es, die dargestellte Architektur im bereitgestellten Tool zu modellieren.



Legende



Teil 2)

1. Lösche die ausgehende Kante zwischen dem Knoten P**rofile Service** und dem Knoten **Database**
2. Lösche die ausgehende Kante zwischen dem Knoten **User Service** und dem Knoten **Database**
3. Lösche die ausgehende Kante zwischen dem Knoten **Report Service** und dem Knoten **Database**
4. Lösche die ausgehende Kante zwischen dem Knoten **Cart Service** und dem Knoten **Database**
5. **Erzeuge einen Knoten Database Cluster** vom Typ *Cluster* und verschieben in den Knoten **Cloud Native Architecture**
6. Verschiebe alle Knoten **Database** vom Typ *Component* in den Knoten **Database Cluster**
7. Erzeuge eine Kante zwischen dem Knoten **Profile Service** und dem Knoten **Database Cluster**
8. Erzeuge eine Kante zwischen dem Knoten **User Service** und dem Knoten **Database Cluster**

[1] https://dzone.com/articles/modern-software-architecture-in-real-life-applicat