FACHBEREICH 03: MATHEMATIK/INFORMATIK
ARBEITSGRUPPE SOFTWARETECHNIK

BACHELORARBEIT
zur Erlangung des akademischen Grades
Bachelor of Science

Visualisierung für eine Software-Architekturprüfung in Virtual Reality

---

ERSTER GUTACHTER: PROF. DR. RAINER KOSCHKE
ZWEITER GUTACHTER: DR. ROBERT PORZEL

David Wagner

4354212

11.09.2020

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)

INHALTSVERZEICHNIS

Seite

ABBILDUNGSVERZEICHNIS

TABELLENVERZEICHNIS

LISTINGS

1 Einleitung

Software Engineering nimmt einen immer größeren Teil in der Informatik ein, da Systeme zunehmend komplexer werden. Als Software Engineering wird die Anwendung eines systematischen und disziplinierten Ansatzes für die Entwicklung, Betrieb und Wartung von Software bezeichnet [1].

Eine Software-Architektur beschreibt die Zerteilung des Systems in einzelne Komponenten und dessen Beziehungen zueinander. Dies ist wichtig, da besonders große Projekte abstrakt dargestellt werden müssen, damit sie von den Beteiligten verstanden werden können. Um zu überprüfen, ob die Architektur auch im Code eingehalten wurde, werden zunehmend Architektur-Analysen angewendet. Diese sind häufig wenig intuitiv nutzbar und werden bei größeren Projekten schnell unübersichtlich.

1.1 Motivation

Durch die Implementierung einer Software-Architektur in einer VR-Umgebung soll es Anwendern erleichtert werden, Software-Architekturen zu verstehen und zu bewerten. Mittels der *City Metaphor* könnten Nutzer sich besser in der Implementierung zurecht finden und dementsprechend leichter die passenden Komponenten finden. Sofortiges Feedback der Architekturanalyse könnte dazu führen, dass die Nutzer schneller Diskrepanzen in der Architektur und Implementierung ausfindig machen.

1.2 Ziel der Arbeit

Die Implementierung der Arbeit basiert auf dem Projekt *SEE*, welches innerhalb der Arbeitsgruppe Softwaretechnik unter der Führung von Prof. Dr. Rainer Koschke entwickelt wird. Das *SEE* Projekt hat das Ziel, Implementierungen als Graph zu importieren und als Stadt darstellen zu lassen. Dabei werden Ordnerstrukturen und Abhängigkeiten der Komponenten sinnvoll visualisiert. Parallel zu dieser Arbeit wurde von Kevin Döhl innerhalb seiner Bachelorarbeit eine Erweiterung entwickelt, die es ermöglicht, in Virtual Reality unter Verwendung der *Leap Motion* eine Architektur zu erstellen, zu bearbeiten und zu exportieren. Diese Implementierung wurde in dieser Arbeit verwendet, um die Architektur darzustellen.

Ziel der Arbeit ist es, das Projekt um eine Software-Architekturanalyse zu erweitern. Es soll möglich sein, Teile der Implementierung auf die Architektur abzubilden. Daraufhin soll eine inkrementelle Analyse, welches auf einer Arbeit von Prof. Dr. Rainer Koschke basiert [2], die betroffenen Abhängigkeiten überprüfen. Die Auswirkungen der Analyse sollen direkt visuell sichtbar sein.

1.3 Herangehensweise

In Kapitel 2 werden theoretische Grundlagen, die für ein Verständnis der Arbeit notwendig sind, kurz erläutert. Anschließend wird in Kapitel 3 die Vorgehensweise und die eigentliche Implementierung beschrieben. Dafür werden zunächst Anwendungsfälle deklariert, welche die Software unterstützen soll. Daraufhin wird in Kapitel 4 die Implementierung evaluiert und ausgewertet.

Den Schlussteil nimmt das Fazit und der Ausblick ein, wo die Erkenntnisse, die aus der Arbeit hervorgegangen sind, zusammengefasst und mögliche zukünftige Arbeiten angesprochen werden.

2 GRUNDLAGEN

In diesem Kapitel werden notwendige Grundlagen erklärt, die für ein Verständnis des Projektes notwendig sind. Dafür wird zuerst kurz anhand von gängigen Definitionen erklärt, was eine Software-Architektur ist und wie man diese bewertet. Anschließend wird das *SEE* Projekt und die Engine *Unity Engine* vorgestellt, auf dem diese Arbeit basiert. Zum Schluss werden noch Projekte aufgeführt, die eine ähnliche Funktion erfüllen.

2.1 Software-Architektur

Eine Software-Architektur beschreibt die komplexen Anforderungen als Aufteilung in konkrete, übersichtliche Systeme. Bass et.al. definieren eine Software Architektur wie folgt:

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

Eine Software-Architektur besitzt beliebig viele Komponenten, welche ebenfalls Unterkomponenten beinhalten können. Diese beschreiben Einteilungen des Systems. Innerhalb einer Software-Architektur wird außerdem definiert, welche Abhängigkeiten innerhalb der Komponenten erlaubt sind.

Eine Software-Architektur dient dazu, Software Projekte besser planen und kontrollieren zu können. Zudem kann diese einen ersten Prototypen darstellen, welcher die Struktur der Implementierung den Stakeholdern übermittelt.
Gleichzeitig kann die Software-Architektur als Fundament für die Implementierung angesehen werden [3].

2.2 Validierung einer Software-Architektur

Murphy et al. (2001) beschrieben ein Model, um Software-Architekturen zu evaluieren. Dies war ihrer Meinung nach erforderlich, da bei Softwareprojekten oft die Implementierung von der eigentlich entworfenen Software-Architektur abweicht.

Für die Berechnung der Unterschiede zwischen Implementierung und Architektur sind drei Eingaben nötig. Zum einen die Implementierung und die Architektur. Als dritte Eingabe benötigt der Algorithmus die Information, welche Komponenten der Implementierung auf welche Architekturkomponenten abgebildet worden sind. Das ist das *Mapping*. Diese drei Eingaben können jeweils als einzelne Graphen dargestellt werden. Dabei stellt ein Knoten eine Komponente dar, und eine Kante definiert eine Abhängigkeit. In Abbildung 1 ist der Prozess des *Reflexion-Models* graphisch dargestellt. Im Folgenden wird dieses schrittweise erklärt.

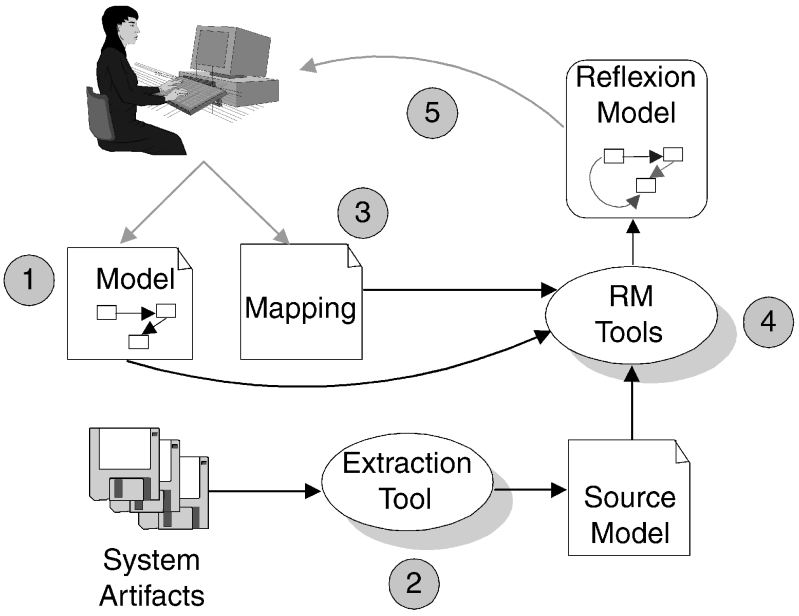


Abbildung 1: Reflexion Model (Murphy, 2001. S.2)

1. **Aufstellung des Architekturmodells:** Im ersten Schritt modelliert der Entwickler eine passende Software-Architektur für das Projekt.
2. **Extrahierung des Implementierungmodells:** Anschließend wird mithilfe eines Tools ein *Source-Model* des Quellcodes erstellt. Dieses enthält verschiedene Informationen des Softwareprojektes wie die einzelnen Komponenten und deren Abhängigkeiten zueinander.
3. **Abbildung der Modelle aufeinander:** Im dritten Schritt werden Komponenten des *Source-Models*auf die Software-Architektur abgebildet. Dies stellt das Mapping dar.
4. **Berechnung des *Reflexion-Models*:** Ein Tool berechnet dann anhand dieser drei Eingaben das *Reflexion-Model* und zeigt mögliche Unterschiede zwischen dem Architektur- und Implementierungsmodell. Die folgenden Fälle können dabei auftreten: Wenn eine Kante in der Architektur spezifiziert wurde und eine entsprechende Abhängigkeit in der Implementierung existiert, ist diese *konvergent*. Existiert solch eine Abhängigkeit in der Implementierung allerdings nicht, so liegt eine *Absenz* vor. Dahingegen werden Abhängigkeiten, die in der Implementierung existieren, aber so nicht in der Architektur spezifiziert wurden, als *Divergenzen* bezeichnet.
5. **Anpassung des Modells:** Im abschließenden Schritt trifft der Entwickler anhand des *Reflexion-Models* weiterführende Schritte, um mögliche Differenzen in der Architektur zu beheben.

Ein Problem dieses Algorithmus ist allerdings, dass das *Reflexion-Model* bei jeder Änderung komplett neu berechnet werden muss, was bei größeren Systemen zu längeren Rechenzeiten führt und somit auch Wartezeit für den Nutzer bedeutet. Auf Basis des *Reflexion-Models* entwickelte Rainer Koschke die *Incremental Reflexion Analysis*. Die *Incremental Reflexion Analysis* berechnet dabei nur die benötigten Komponenten neu, die durch eine Änderung der Graphen entstehen [2]. Dies hat zur Folge, dass die Rechenzeit um bis zu 60% verkürzt werden kann.

2.3 Unity

Unity ist eine Laufzeit- und Entwicklungsumgebung, die primär für die Entwicklung von Spielen genutzt wird. Vorteile von Unity sind unter anderem die unterstützten Zielplattformen, wie beispielsweise PC-Betriebssysteme (Windows, Linux, MacOS), Spielekonsolen (Xbox, Playstation, Nintendo), Mobile Betriebssysteme (iOS, Android) und diverse Webbrowser. Außerdem werden verschiedene Formen der virtuellen und erweiterten Realität (Virtual Reality/Augmented Reality) unterstützt [1]. Die Programmierung von eigenen Skripten wird über die Programmiersprache C# ermöglicht.

Das Beispiel 1 zeigt, wie ein neu erstelltes Script aufgebaut ist. Unity Skripte erben standardmäßig von der Unity Klasse *MonoBehavior*, die viele Funktionalitäten für die Entwicklung bereitstellt. Außerdem werden die Funktionen *Start* und *Update* automatisch generiert. In der *Start* Funktion können Befehle programmiert werden, die beim Start der Anwendung ausgeführt werden. Die *Update* Funktion wird hingegen jeden Frame ausgeführt. Diese eignet sich besonders für Abfragen, ob beispielsweise bestimmte Tasten gedrückt werden oder andere Bedingungen erfüllt sind.

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class NewBehaviourScript : MonoBehaviour
{
    // Start is called before the first frame update
    void Start()
    {

    }

    // Update is called once per frame
    void Update()
    {

    }
}
```

Listing 1: Neues Unity Script

Unity benutzt für die Repräsentation von Objekten sogenannte *GameObjects*. Es gibt dabei vorgefertigte, sogenannte primitive GameObjects, die beispielsweise Würfel, Kugeln oder Quader darstellen. Jedes GameObject kann *Components* beinhalten, die Skripte mit eigenen Funktionen haben. Allerdings existieren auch leere GameObjects, die keine visuelle Präsenz haben, aber dennoch Komponenten beinhalten können.

[1] https://unity.com/de/features/multiplatform

2.4 CodeCity

Um Daten und Code für Menschen verständlich zu strukturieren und visualisieren, entstand Ende der 1980er Jahre die *City Metaphor*. Der Gedanke, Daten als Stadt zu visualisieren beruht auf der These, dass Menschen sich schon früh in Städten zurecht finden müssen und dies auf Quellcode übertragen werden kann [4]. Eine Umsetzung dieser *City Metaphor* erreichte Wettel in 2008 mit dem Projekt *CodeCity* [5]. Dabei handelt es sich um eine interaktive 3D-Visualisierung zur Analyse von objektorientierten Softwaresystemen. Abbildung 2 zeigt die visualisierte Form eines kleinen Softwaretools namens ArgoUML, welches mittels CodeCity visualisiert wurde. Eigenschaften der Elemente werden beispielsweise durch die Höhe, Breite oder Anordnung der Gebäude dargestellt.

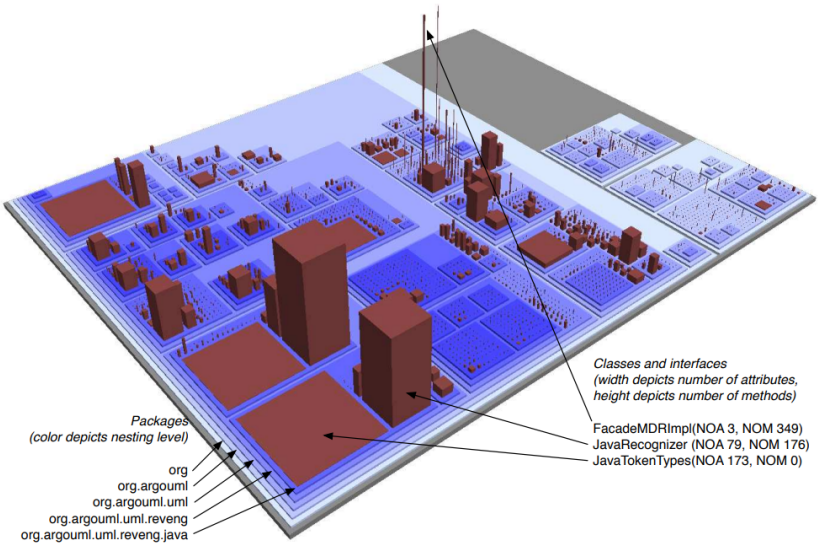


Abbildung 2: ArgoUML mittels CodeCity visualisiert

2.5 SEE

Das *SEE*-Projekt wurde und wird in der Arbeitsgruppe Softwaretechnik unter der Leitung von Prof. Dr. Rainer Koschke entwickelt und steht für Software Engineering Experience. SEE ist eine Anwendung, die in der Unity Engine entwickelt wurde, und visualisiert hierarchische Graphen von Software, basierend auf der *City Metaphor*, welche in Abschnitt 2.4 erläutert wurde.

Importieren kann man Quellcode in Form von *GXL*[2] Dateien, einem standardisierten Format zur Beschreibung von Graphen. Anhand von Graphen kann man die Struktur von Software darstellen. Dateien oder Ordner aus der Implementierung werden dabei als Knoten im Graph beschrieben und Beziehungen zwischen diesen als Kanten.

Derzeit werden drei unterschiedliche Anwendungsszenarien unterstützt, die jeweils über eine eigene Unity Szene gestartet werden können. Das erste Anwendungsszenario

[2] http://www.gupro.de/GXL/Introduction/background.html

visualisiert statische Informationen einer konkreten Software Version. Bei dem zweiten Anwendungsszenario können statische Informationen einer Software im Verlauf der Software Versionen animiert betrachtet werden. Das dritte Anwendungsszenario kann das Laufzeitverhalten einer Software beobachtet werden.

Es werden verschiedenste Eingabegeräte unterstützt, wie eine klassische Desktopsteuerung mit Tastatur und Maus, Gamepads, Touchscreen und verschiedene VR-Geräte. Abbildung 3 zeigt die Komponente, mit welcher man die CodeCity generieren kann.

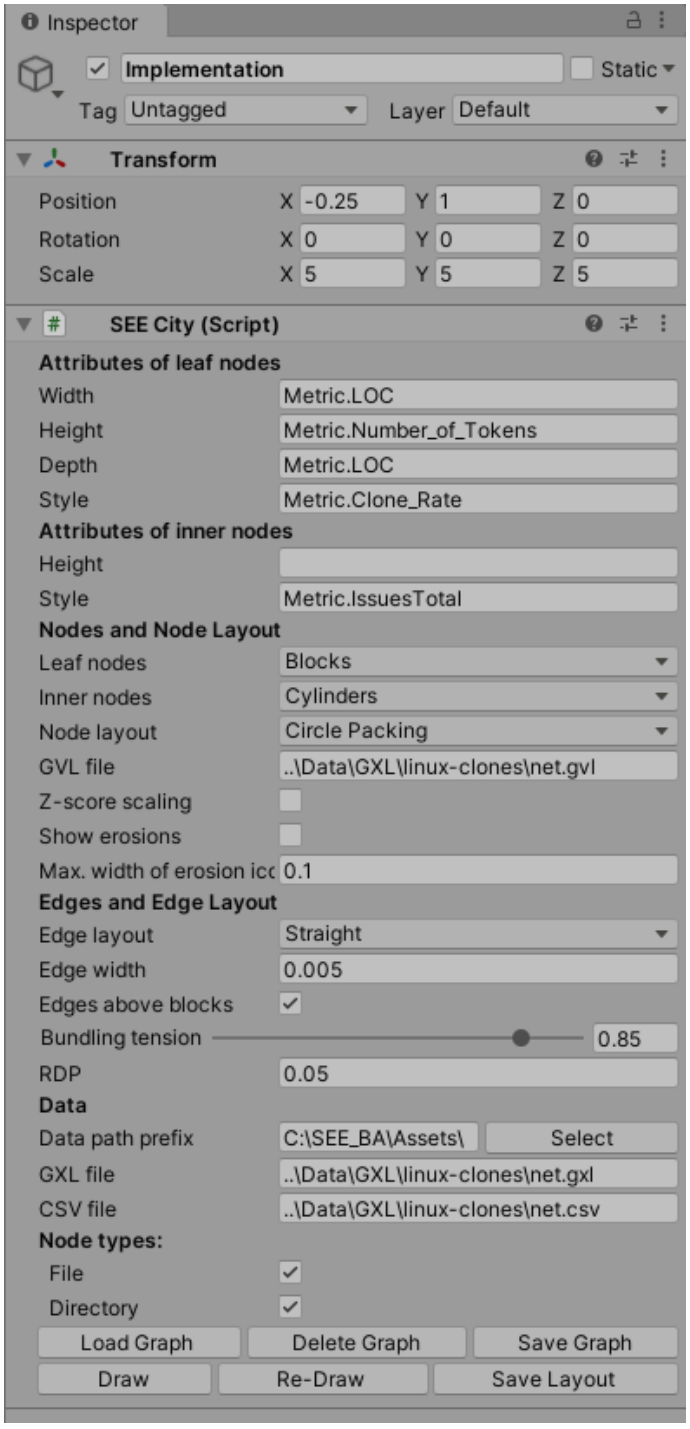


Abbildung 3: SEE Komponente im Unity Editor

Dabei kann man verschiedene Einstellungen vornehmen. Über die Position und Scale Variablen lassen sich die Position und die Skalierung der CodeCity einstellen. Außerdem kann man auf bestehende Metriken zugreifen, falls diese vorhanden sind. Optional kann man die Abhängigkeiten innerhalb der Implementierung mittels Kanten visualisieren. Dabei besitzt die Kante einen farblichen Verlauf von Grün zu Rot, wobei grün den Ursprung und Rot das Ziel der Abhängigkeit signalisiert. Anschließend kann man das Layout der CodeCity festlegen. Blätter können als Blöcke oder Gebäude dargestellt werden. Innere Knoten können als Zylinder, Blöcke, Gebäude, Rechtecke, oder Donuts dargestellt werden. Das allgemeine Layout hat folgende Optionen:

- Evo Streets
- Balloon
- Rectangle Packing
- Treemap
- Circle Packing
- Manhattan
- From File

Man kann eine optionale GVL Datei angeben, die ein bestehendes Layout enthält. Zudem kann man sich Erosionen anzeigen lassen. Abhängigkeiten können als Kanten in den Layouts Straight, Bundling oder Spline generiert werden. Der eigentliche Graph wird aus der GXL Datei importiert. Über den Button *'Load Graph'* wird dieser importiert. Um diesen in der Unity Welt generieren zu lassen, muss man anschließend *'Draw'* betätigen. Änderungen im Graphen kann man direkt über *'Save Graph'* speichern. Das Layout der Positionen kann man über *'Save Layout'* speichern und wird bei einem erneuten generieren berücksichtigt. *'Delete Graph'* löscht die DataCity. Eine generierte Stadt ist in Abbildung 4 sichtbar.

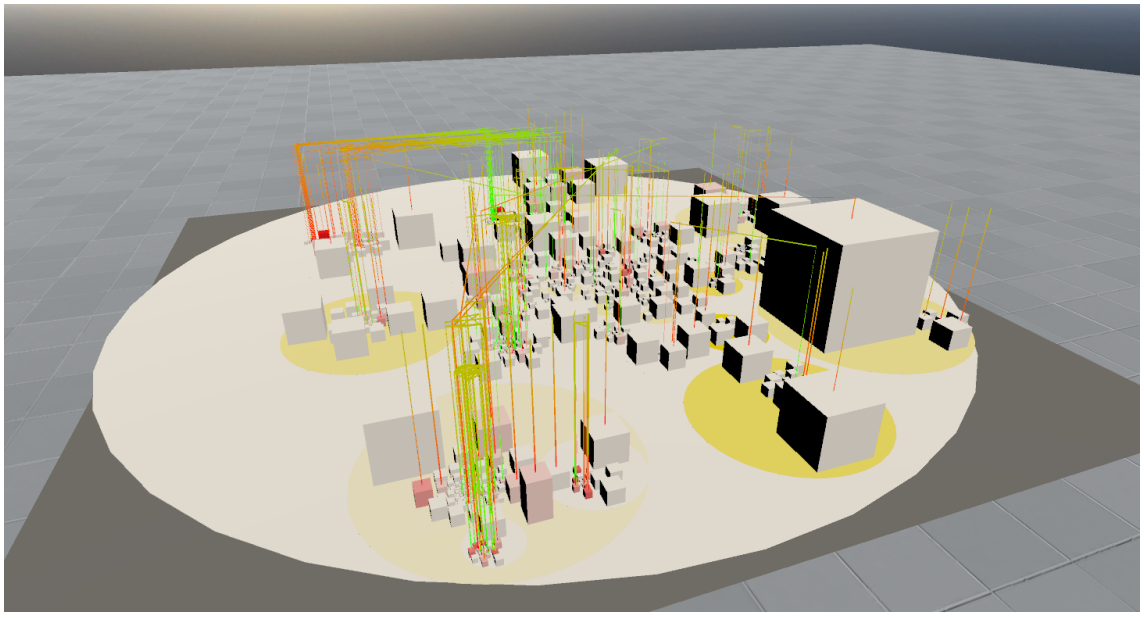


Abbildung 4: Net Graph im Circle Packing Layout

2.6 Modellierung einer Software-Architektur in SEE

Parallel zu dieser Arbeit wurde von Kevin Döhl eine weitere Erweiterung für das SEE-Projekt entwickelt. Bei dieser Erweiterung handelt es sich um die Modellierung einer Software-Architektur in Virtual Reality unter der Verwendung des *Leap Motion* Controllers. Bei der *Leap Motion*[3] handelt es sich um ein Gerät, welches die Hände aufnimmt und die Bewegung ohne Verzögerung wiedergibt. Diese kann mittels verschiedener Gesten als eine Interaktionsform dienen. Die Leap Motion kann beispielsweise an eine VR-Brille befestigt werden. Die Funktionen, die zum Zeitpunkt dieser Arbeit unterstützt werden, werden im Folgenden aufgelistet.

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

Die Erzeugung eines Knotens erfolgt durch das Berühren der Spitze des linken Zeigefingers an die Spitze des rechten Zeigefingers. Abbildung 5 zeigt das Menü, welches beim Umklappen der Hand erscheint. Durch das Antippen eines Knopfes des Menüs wird die entsprechende Funktion ausgeführt. So kann man einzelne Knoten auswählen und benennen, skalieren oder löschen. Außerdem können Kanten zwischen Knoten erstellt werden und der Spieler kann sich in X und Y Position bewegen.

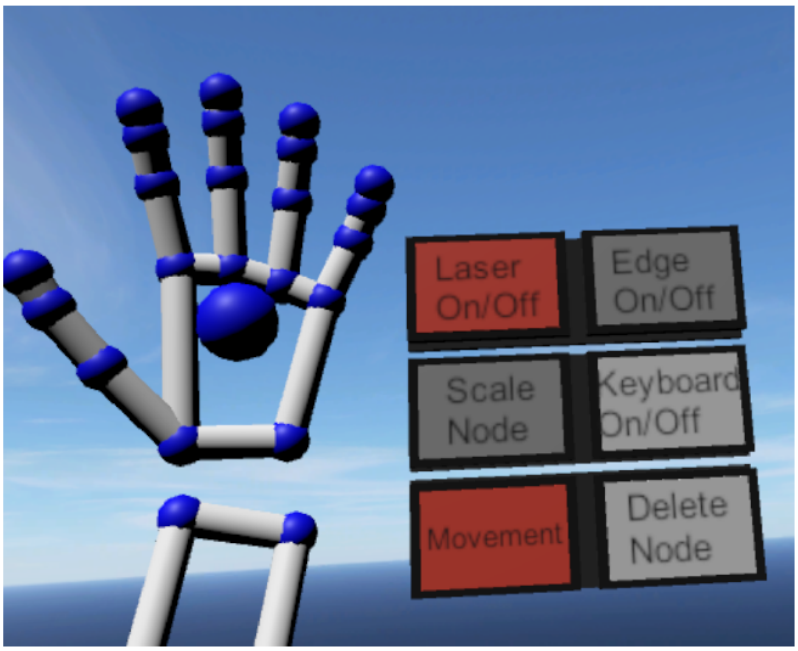


Abbildung 5: Menü für das Auswählen verschiedener Funktionen in der Architekturmodellierung (Döhl 2020)

[3] https://www.ultraleap.com/product/leap-motion-controller/

Abbildung 6 zeigt Teile einer in der Anwendung nachgebauten alten Architektur von Twitter. Jede Architekturkomponente wird als graues Rechteck dargestellt, wobei der Name mittig im oberen Bereich beschriftet ist. Unterkomponenten sind entsprechend kleiner auf einer Oberkomponente platziert, und haben einen dunkleren Grauton, um sich farblich hervorzuheben. Abhängigkeiten innerhalb der Architekturkomponenten werden als Kanten visualisiert, wobei dieser einen farblichen Übergang von Grün zu Rot besitzt. Grün stellt dabei den Ursprung, und Rot das Ziel der Abhängigkeit dar.

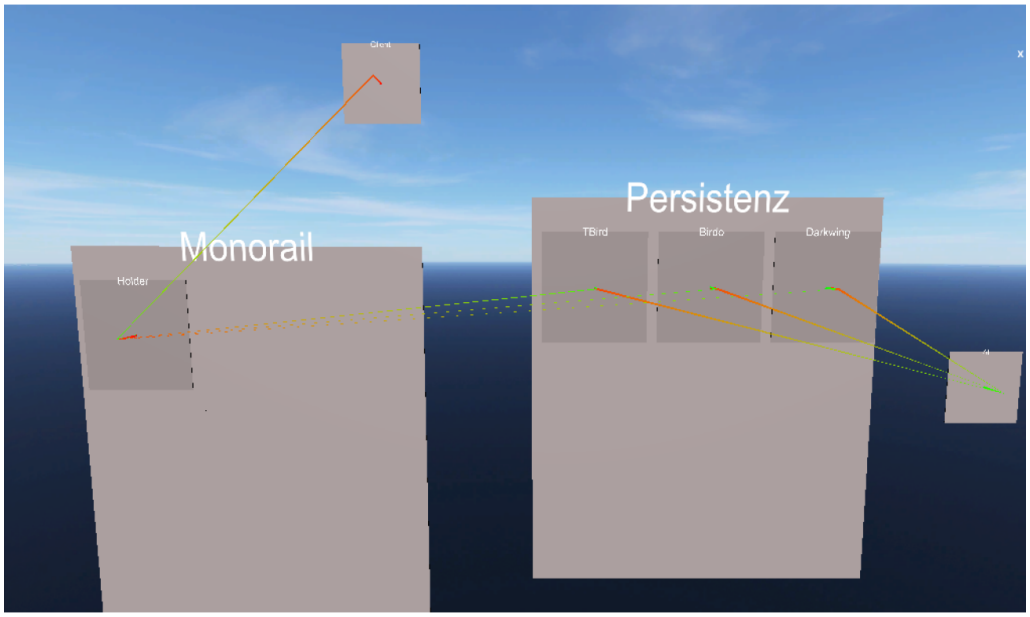


Abbildung 6: Darstellung einer alten Teil-Architektur von Twitter (Döhl 2020)

Die Komponenten kann man beliebig bewegen und skalieren. Zudem kann man die erstellte Architektur im GXL Format sowohl exportieren, als auch importieren. Wenn die Architektur in der Anwendung selbst erstellt wurde, wird das bestehende Layout mitgespeichert und wiederhergestellt, wenn man es importiert. Man kann aber auch andere Software-Architekturen importieren, sofern diese im GXL Format vorliegen. Das Layout für dieses wird dann statisch generiert.

2.7 Ähnliche Arbeiten

Das *Bauhaus Projekt* entstand als Forschungsprojekt zwischen den Universitäten Stuttgart und Bremen. Ziel des Projektes war es Softwareerosion zu verhindern. Um dies zu erreichen, wurden verschiedene Tools erstellt, die im Bereich der Software-Architektur, Software-Analyse und Software Reenginering eingesetzt werden. Aus dem Bauhaus Projekt entstand anschließend das Unternehmen Axivion, welches das Vorhaben im kommerziellen Markt weiterführt[4]. Es wird keine inkrementelle Analyse benutzt, was zur Folge hat, dass bei jeder Änderung die komplette Analyse berechnet werden muss. Zudem wird das Ergebnis der Analyse als neuer Graph dargestellt und das Layout der alten Architektur muss manuell importiert werden.

[4]https://en.wikipedia.org/wiki/Bauhaus_Project_(computing)

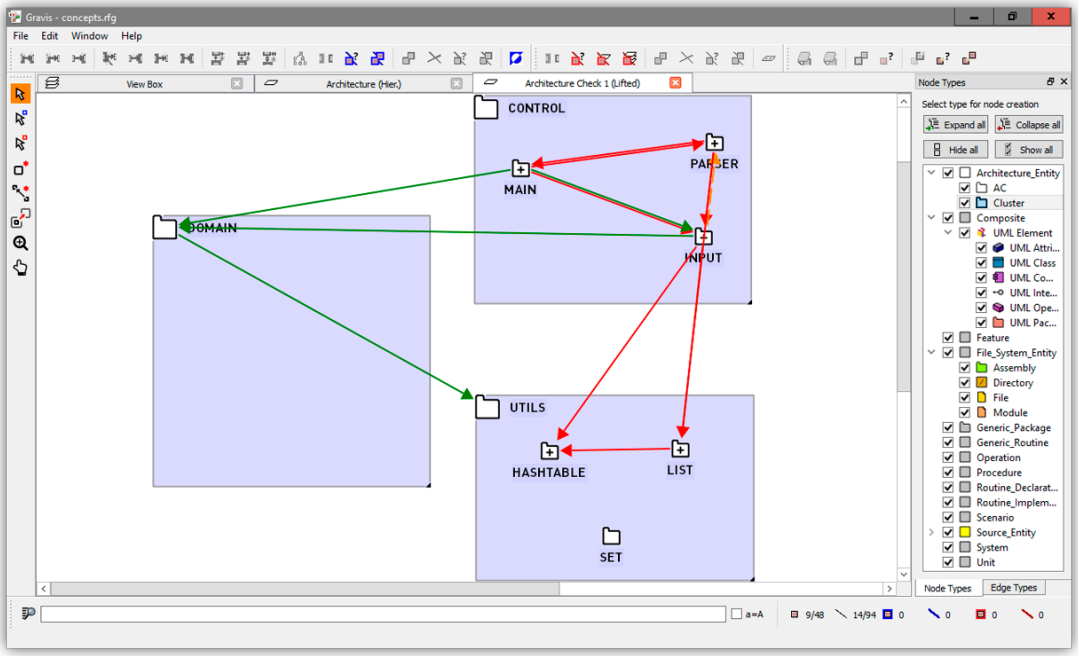


Abbildung 7: Axivion Software-Architekturprüfung

Sotograph[5] ist ein Produkt des Unternehmens *hello2morrow*, welches ein ähnliches Ziel verfolgt. Es ist möglich, eigene Software-Architekturen zu modellieren und diese mit der Implementierung zu vergleichen. Die Resultate dieser Analyse werden in einer zweidimensionalen Ansicht visuell dargestellt. Die Überprüfung erfolgt hierbei nach jeder Änderung. Die Ursache der Architekturabweichung kann auf die entsprechende Stelle im Quellcode zurückverfolgt werden. Außerdem kann das Tool die Überprüfung über verschiedene Versionen der Implementierung durchführen und somit einen Verlauf, beziehungsweise einen Trend aufzeigen. Unterstützt werden die Programmiersprachen Java, C und C/C++.

Abbildung 8: Sotograph Software-Architekturprüfung

[5]https://www.hello2morrow.com/products/sotograph/sotograph

Einen anderen Ansatz verfolgt das Tool *Lattix Architect*[6]. Hier werden anhand einer *Dependency Structure Matrix* die Abhängigkeiten innerhalb eines Software-Systems veranschaulicht. Die Zahlen in den Kästen geben die Stärke der Abhängigkeiten zwischen den verschiedenen Komponenten der Implementierung an. Dadurch kann man feststellen, welche Auswirkung eine Änderung in einer Datei hätte. Die Zahlen in der Diagonale der Matrix geben an, was für einen prozentualen Anteil diese Komponente vom Gesamtsystem ausmacht.

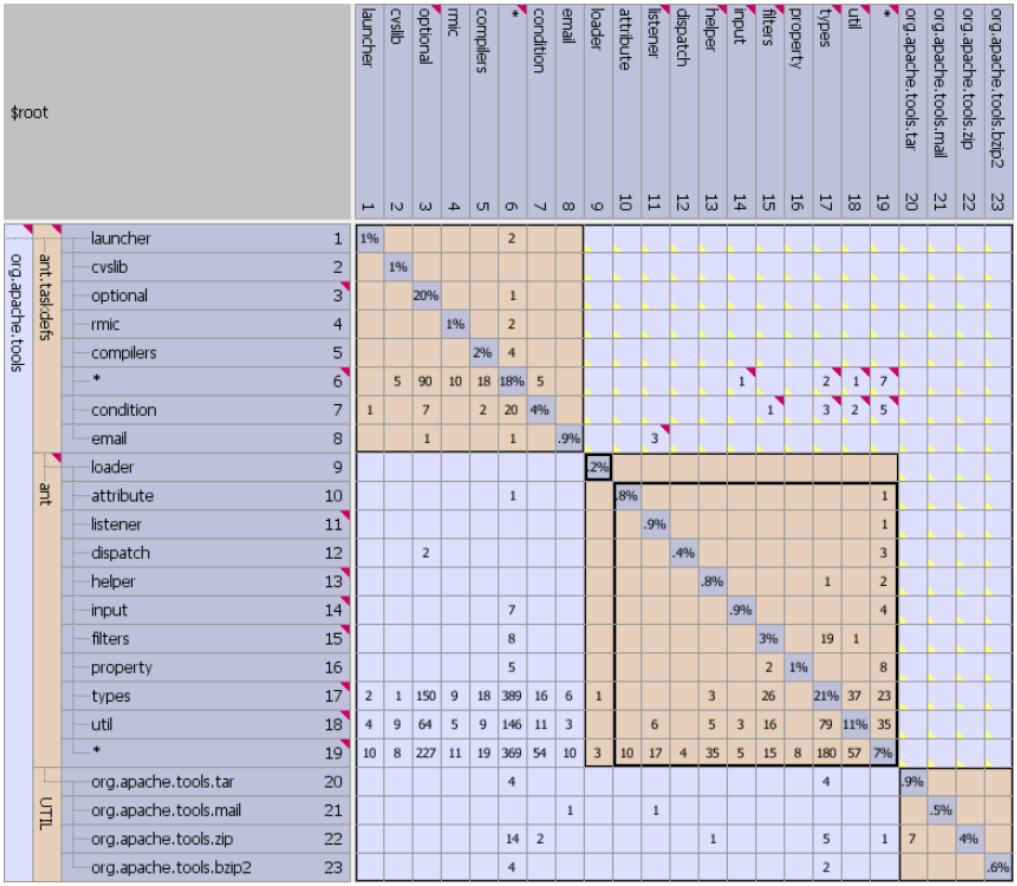


Abbildung 9: Lattix Software-Architekturprüfung

[6]https://www.lattix.com/products-architecture-issues/

3 Konzept und Implementierung

Im Folgenden wird das Konzept und die Implementierung der Arbeit erläutert. Dafür werden zunächst Anwendungsfälle aufgestellt, welche die Anwendung unterstützen soll. Anschließend werden nicht-funktionale Anforderungen aufgestellt. Daraufhin wird das allgemeine Vorgehen erklärt und die Implementierung aufgebaut ist. In Kapitel 3.4 werden die Funktionen vorgestellt, die vor Beginn der Implementierung geplant gewesen sind. Dabei wird zwischen Funktionen, die unerlässlich sind (Basisfunktionen) und optionalen Funktionen unterschieden. Anschließend wird in Kapitel 3.5 ausführlich erklärt, welche Funktionen entwickelt worden sind und wie diese programmiert wurden. Funktionen, die innerhalb dieser Arbeit nicht umgesetzt werden konnten, werden in Kapitel 3.6 erläutert. Abschließend werden Probleme aufgelistet, die die Implementierung zum Zeitpunkt der Abgabe enthält.

Als Basis für die Implementierung fungierte die aktuelle Version des SEE Projektes. Um eine problemlose Funktion und Weiterentwicklung der Arbeit zu ermöglichen, war es nötig, diese in regelmäßigen Abständen auf die aktuellste Version des Projektes zu aktualisieren. Dies hatte zur Folge, dass teilweise bereits funktionierender Code angepasst werden musste, da größere Änderungen im Basisprojekt vorgenommen wurden.

3.1 Anwendungsfälle

Nun folgen Anwendungsfälle, die die Anwendung unterstützen sollte. Die Anwendung richtet sich primär an Experten, wie beispielsweise Software-Architekten. Für die Benutzung werden keine besonderen Kenntnisse benötigt.

3.1.1 Anwendungsfall 1: Bestehende Implementierung auf Architektur mappen

Vorbedingung: Architektur vorhanden, Implementierung vorhanden

Schritte um Anwendungsfall zu lösen:

1. Nutzer importiert Architektur und Implementierung und startet die Anwendung
2. Nutzer mappt die Implementierungskomponenten auf die Software-Architektur
3. Nutzer exportiert das Mapping

Nachbedingung: Nutzer hat das Mapping im GXL Format

3.1.2 Anwendungsfall 2: Probleme in Mapping finden

Vorbedingung: Architektur vorhanden, Implementierung vorhanden, Mapping vorhanden

Schritte um Anwendungsfall zu lösen:

1. Nutzer importiert Architektur und Implementierung und startet die Anwendung
2. Nutzer importiert Mapping
3. Nutzer erkennt Probleme in der Architektur anhand der Prefabs an den Kanten

4. Nutzer erkennt Ursprung der Probleme anhand der Kanten zwischen den Implementierungskomponeten

Nachbedingung: Nutzer kennt Probleme in bestehender Architektur

3.1.3 Anwendungsfall 3: Probleme durch Änderung im Mapping beheben

Vorbedingung: Architektur vorhanden, Implementierung vorhanden, Mapping vorhanden

Schritte um Anwendungsfall zu lösen:

1. Nutzer importiert Architektur und Implementierung und startet die Anwendung
2. Nutzer importiert das Mapping
3. Nutzer mappt Implementierungskomponenten neu, um Probleme zu beheben
4. Nutzer exportiert das Mapping

Nachbedingung: Nutzer hat Probleme durch Änderung des Mappings behoben

3.2 Nicht-funktionale Anforderungen

Nicht-funktionale Anforderungen beschreiben, wie gut die Funktionen vom System umgesetzt werden [6]. Im Folgenden werden die wichtigsten [7] nicht-funktionalen Anforderungen aufgestellt und deren Auswirkungen auf die Arbeit erörtert.

Performance: Die Performance sollte sich nicht stark von der allgemeinen Performance der anderen Anwendungsfälle des SEE Projektes unterscheiden.

Usability : Eine schnelle Nutzung und Erlernbarkeit der Anwendung sollte für die Nutzer möglich sein.

Sicherheit: Die Anwendung sollte jederzeit funktionieren.

Accessibility : Die Anwendung sollte für Personen mit Behinderungen verwendbar sein.

Interoperabilität : Durch den Einsatz der universalen Dateien sollte die Verwendung in anderen Anwendungen gewährleistet werden.

Wartbarkeit: Entwickler sollten auftretende Fehler beheben oder neue Funktionen hinzufügen können. Dafür ist eine klare Dokumentation zwingend erforderlich.

Funktionalität: Die Auswirkung des Mappings soll direkt, korrekt und verständlich angezeigt werden.

Fehlertoleranz: Es sollte eine gewisse Fehlertoleranz bei dem Auswählen der Objekte geben.

Wiederverwendbarkeit: Zwischenergebnisse sollten gespeichert und fortgesetzt werden können.

Skalierbarkeit: Die Anwendung sollte mit beliebig großen Software-Architekturen und Implementierungen funktionieren.

Modularität: Die Anwendung sollte andere Komponenten des *SEE* Projekts nicht beeinflussen.

3.3 Allgemeines Vorgehen

Um eine problemlose Kompatibilität zu gewährleisten, wurde sich an dem Vorgehen des SEE Projektes orientiert. Hervorzuheben ist hierbei, dass die verschiedenen Eingabegeräte und dessen Aktionen abstrakt und modular aufgebaut sind. Nun wird zunächst erklärt, wie die Mapping Szene im SEE Projekt aufgebaut ist, und welche Funktionalitäten die einzelnen Komponenten haben.

Player Settings
Das Player Settings Objekt mit der entsprechenden Komponente hat den Zweck, das gewünschte Eingabegerät zu wählen. Vom SEE Projekt werden zurzeit VR Geräte, Gamepads und eine Desktop Steuerung unterstützt. Außerdem kann man diverse Einstellungen für VR Geräte treffen.

Player
Für jede Eingabevariante muss ein entsprechendes Objekt (VRPlayer, DesktopPlayer, GamepadPlayer) vorhanden sein, welches jeweils eine Actor Komponente besitzen muss. Die Actor Komponente definiert dabei die möglichen Eingaben des jeweiligen Gerätes und die entsprechende Aktion. Für jede komplexere Funktionalität sollte also ein Device Object erstellt werden und anschließend ein Action Object, welches die Eingaben des Device Objects abruft. Primär wurde im Rahmen dieser Arbeit die Anwendung für VR-Systeme konzipiert. Als zusätzliche Steuerungsmethode wurde eine Desktop Steuerung mit Hilfe von Maus und Tastatur entwickelt. Dies war aufgrund der erwähnten Abstraktion ohne größere Aufwände machbar, da man die Implementierung als allgemeine Oberklasse entwickelt und diese in vererbten Klassen für die Steuerungsvarianten anpasst.

Implementation
Das Implementation Object besitzt das in 2.1 beschriebene SeeCity Script und ist dafür verantwortlich, die Implementierung als CodeCity zu generieren.

Whiteboard
Das Whiteboard Object enthält das Skript, welches für die Darstellung der Software-Architektur zuständig ist.

Mapper
Das Mapper Object enthält Einstellungen für die Reflexion Analysis. Hier kann man die Prefabs einfügen, die man für die verschiedenen Status der Analyse nutzen will. Außerdem werden dem GameObject die Klone der gemappten Implementierungskomponenten angehängt.

3.4 Geplante Funktionen

Als Nächstes folgen die Funktionen, die vor der Implementierung geplant wurden. Dabei wurde zwischen Basisfunktionen, die essenziell für die Arbeit sind und optionalen Funktionen unterschieden. Die technische Dokumentation der Implementierung dieser erfolgt in Kapitel 3.5.

3.4.1 Basisfunktionen

Einpflegen der Reflexion-Analysis
Die *Reflexion-Analysis* muss in die Implementierung eingepflegt und initialisiert werden. Da die Analyse dem Observer Pattern[7] folgt, muss ein Observer eingerichtet und beim Start angemeldet werden.

Mapping von Objekten
Um eine Software-Architektur evaluieren zu können, bedarf es neben der Implementierung und der Software-Architektur das Mapping. Das Mapping gibt an, welche Implementierungskomponenten auf welche Architekturkomponenten abgebildet werden. Deshalb muss es möglich sein, gezielt Implementierungskomponenten abbilden, beziehungsweise *mappen* zu können. Es muss zudem ersichtlich sein, welche Objekte bereits gemappt wurden.

Visuelle Darstellung der Architekturprüfung
Das Ergebnis der Architekturprüfung soll visuell dargestellt werden. Es soll also erkennbar sein, welche Abhängigkeiten in der Architektur erlaubt, nicht erlaubt oder ungenutzt sind. Dazu ist es auch nötig, neue Kanten in der Architektur zu erstellen und zu löschen.

Import/Export des Mappings
Es soll möglich sein, ein Mapping im GXL Format zu importieren und daran weiterarbeiten zu können. Auch soll es eine Funktion geben, die das Mapping, welches innerhalb des Projektes vorgenommen wurde, exportiert.

3.4.2 Optionale Funktionen

Weitere Visualisierungen
Es sind noch weitere Visualisierungen denkbar, die die Analyse visuell unterstützen. Beispielsweise wäre es hilfreich, das entsprechende Original des gemappten Objektes farblich hervorzuheben, falls das Original fokussiert ist.

Weitere Eingabegeräte
Die Arbeit wird primär für die HTC Vive, inklusive der Controller entwickelt. Denkbar ist auch eine Steuerung für Desktop, Gamepads oder andere VR Geräte.

Anpassung der Architektur
Es wäre hilfreich, wenn der Nutzer während des Spielens die Architektur nach seinen Vorstellungen positionieren, skalieren und rotieren könnte.

[7] https://sourcemaking.com/design_patterns/observer

3.5 Umsetzung der Funktionen

Nun folgt die technische Erläuterung, wie die Funktionen aus Kapitel 3.4 umgesetzt wurden. Abbildung 10 veranschaulicht die Funktionen der verschiedenen Klassen als UML-Klassendiagramm. Dabei wurde sich nach dem bisherigen Vorgehen des SEE-Projektes orientiert, welches die Eingaben und die Aktionen trennt.

Die Klasse *Mapper* erweitert die abstrakte Basisklasse *Inputdevice* um die Variablen MappingButton, Highlightbutton und Cancelbutton. Diese werden anschließend in den Klassen *MouseMapper* und *XRMapper* implementiert und werden für das jeweilige Anwendungszenario in Desktop bzw. VR-Umgebung verwendet. Im MouseMapper-Skript werden dabei die Buttons als *KeyCodes* implementiert, welche in Unity direkt einstellbar sind. Die Buttons im XRMapper hingegen werden von SteamVR abgerufen, die dort definiert wurden.

Die Klasse *MappingAction* beinhaltet die Logik der Anwendung, in der die Software-Architektur Analyse initialisiert wird und die Aktionen, die auf das Betätigen der Eingaben aus dem *MapperInput* erfolgen. Um zu funktionieren, benötigt das MappingAction Script ein Mapper Objekt, um die Eingaben zu erkennen. Zudem wird ein Observer in der Klasse *MappingObserver* angemeldet, der über Änderungen in der Analyse benachrichtigt. In der Klasse *MapperSettings* werden die Prefabs und die GameObjects der Software-Architektur, sowie der Implementierung hinterlegt.

Die verschiedenen Eingabeoptionen benötigen auch teilweise Änderungen in dem Action Script. Diese Anpassungen werden in den Klassen *Mapping2DAction* und *Mapping3DAction*, die von der Klasse MappingAction erben, implementiert.

Initialisierung der Reflexion Analysis
Beim Start des Projektes wird zunächst die Architektur generiert und angepasst. Die Größe der Architektur richtet sich dabei an der festgelegten Größe der Implementierung. Anschließend wird geprüft, ob die ausgewählten Graphen der Architektur und Implementierung nicht leer sind. Wenn dies nicht der Fall ist, wird mittels dieser Graphen ein neues Reflexion Objekt erstellt, welches für die Berechnung der Software-Architekturanalyse zuständig ist. Bei diesem Reflexion Objekt wird anschließend ein Observer angemeldet, der bei jeder Änderung in der Analyse informiert wird.

Mapping von Objekten
Das Auswählen der Komponenten erfolgt mittels eines *Raycasts*[8]. Innerhalb der Update Funktion wird geprüft, ob und welches Objekt der Raycast getroffen hat. In der VR-Steuerung wird das anvisierte Objekt über einen *Line Renderer* visualisiert, der einen Laser darstellt. Bei der Desktop Variante kann man ein Objekt über die linke Maustaste auswählen. Das aktuell anvisierte Objekt wird dabei grün hervorgehoben. Über einen wählbaren Mapping Button lässt sich das aktuell anvisierte Objekt zwischenspeichern. Um dies farblich zu betonen, wird dieses Objekt blau gefärbt. Anschließend kann man diese Komponente, inklusive ihrer Unterkomponenten, auf die gewünschte Architekturkomponente mappen.

[8] https://docs.unity3d.com/ScriptReference/Physics.Raycast.html

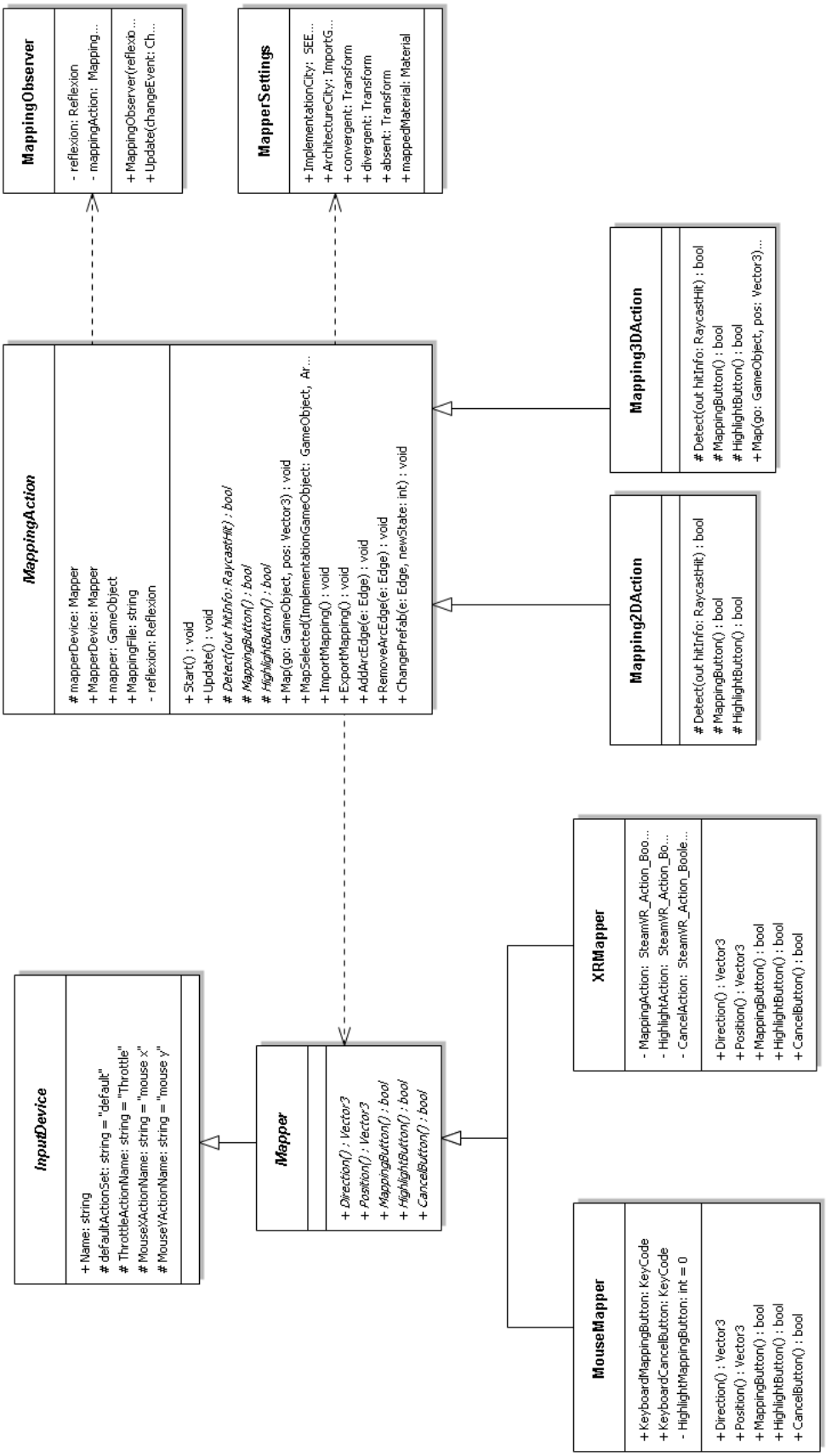


Abbildung 10: UML-Klassendiagramm der implementierten Klassen

Um das zuvor zwischengespeicherte Objekt auf eine Architekturkomponente abzubilden, muss man die Architekturkomponente anvisieren und den Mapping Button betätigen. Die Implementierungskomponente wird anschließend als geklontes GameObject auf die jeweilige Architekturkomponente angefügt. Die originale Implementierungskomponente wird daraufhin in der CodeCity mit einer transparenten Textur versehen, um bemerkbar zu machen, dass es bereits gemappt ist.

Abbildung 11: Eine gemappte Implementierungskomponente

Visuelle Darstellung der Analyse
Bei einer Veränderung des Status einer Kante, ruft der Observer die Funktion *ChangeState* auf. Dafür wurden die in Abbildungen 12, 13 und 14 sichtbaren 3D-Modelle erstellt, die den Status repräsentieren. Bei einer Veränderung eines Status, wird das alte Prefab gelöscht und das Prefab, das den neuen Status repräsentiert, an die Mitte der entsprechenden Kante in der Architektur platziert. Bei einer Divergenz muss zudem eine neue Kante in der Architektur erstellt werden. Falls diese Divergenz im Zuge einer Änderung behoben wird, wird die entsprechende Kante wieder gelöscht.

Abbildung 12: Prefab Absenz

Abbildung 13: Prefab Konvergenz

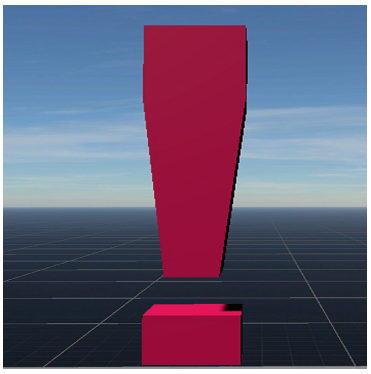


Abbildung 14: Prefab Divergenz

Weitere Visualisierungen
Um die Auswirkungen der Architekturanalyse visuell zu verstärken, wurden weitere Funktionen entwickelt. Falls eine Implementierungskomponente bereits gemappt ist und diese anvisiert wird, wird das entsprechende Gegenstück in der CodeCity bzw. die Kopie auf der Architektur farblich hervorgehoben. Außerdem werden alle Kanten, die zwischen Implementierungskomponenten existieren, welche auf die Architektur abgebildet wurden, auch in der Architektur angezeigt. Dadurch ist es nachvollziehbar, weshalb eine Konvergenz, beziehungsweise Divergenz, in der Architektur vorliegt.

Import/Export
Um ein Weiterarbeiten an einem bestehenden Mapping zu ermöglichen, war es nötig, eine Funktion zum Exportieren und Importieren von Mapping bereitzustellen.
Da das Mapping intern als Graph Objekt behandelt wird und SEE eine Export Funktion für Graph Objekte bereitstellt, war der Export mit wenig Aufwand verbunden. Der Mapping Graph enthält alle Abbildungen der Implementierung auf die Architektur, und wird als GXL Datei exportiert.

Der Import ist so aufgebaut, dass die Mappings, die innerhalb der GXL Datei vorhanden sind, importiert werden und die Mapping Funktion für jede aufgerufen wird. Abschließend wurden die Funktionen noch in der GUI integriert.

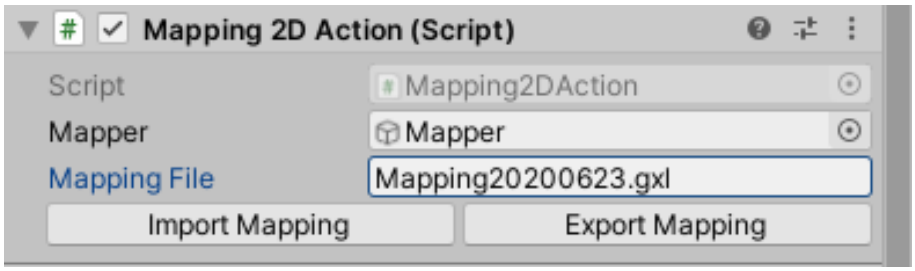


Abbildung 15: Import und Export Oberfläche

Eingabeoptionen
Zusätzlich zur VR-Steuerung, die zur Beginn der Arbeit als einzige Steuerungsvariante geplant gewesen war, wurde eine Steuerungsvariante für den Desktop mit Tastatur und Maus entwickelt. Dies war aufgrund der modularen Steuerungskomponente des SEE-Projektes problemlos möglich. Die Belegung der VR-Steuerung erfolgt über SteamVR. Bei dem SteamVR-Plugin für Unity handelt es sich um eine Erweiterung, die zum einen die 3D-Modelle der Controller bereitstellt, die man während der Benutzung sieht. Außerdem werden die Hände modelliert, sowie Position und Bewegung simuliert. Zuletzt wird die Tastenbelegung der VR-Controller in SteamVR [9] vorgenommen.

Bei der Desktop-Steuerung kann die Belegung direkt in Unity vorgenommen werden. Standardmäßig erfolgt die Bewegung über die übliche *WASD*-Belegung. Für die Bewegung der Kamera wird die rechte Maustaste benutzt. Die linke Maustaste benutzt man zum Anvisieren von Komponenten. Über die Taste *M* kann man Implementierungs-

[9] https://valvesoftware.github.io/steamvr_unity_plugin/

komponenten für das Mapping zwischenspeichern und auf Architekturkomponenten abbilden.

3.6 Fehlende Funktionen

Die Basisfunktionen, wie in Kapitel 3.4 beschrieben, konnten alle plangemäß umgesetzt werden. Bei den optionalen Funktionen konnte zusätzlich zur VR-Steuerung eine Desktop Steuerung umgesetzt werden. Eine Gamepad-Steurung und andere VR-Geräte konnten mangels Zeit nicht implementiert werden. Die optionale Funktion, Kanten anhand ihres Status mit einer Art Aura zu ummanteln, wurde ebenfalls nicht in dieser Arbeit umgesetzt. Zudem fehlt die dynamische Anpassung der Architektur in Form von Position, Größe und Rotation. Diese Funktionen konnten ebenso aufgrund fehlender Zeit nicht implementiert werden.

3.7 Probleme

Während des Testens sind Probleme aufgefallen, die nicht innerhalb dieser Arbeit gelöst werden konnten. Im Folgenden werden diese aufgeführt und mögliche Lösungsansätze geliefert.

Performance
Die Implementierung zeigt noch teilweise Schwächen in der Performance, sodass es bei großen Graphen und vielen abgebildeten Komponenten zu Wartezeiten kommen kann. Dies kann beispielsweise bei der Fokussierung einer Implementierungskomponente auftreten, um das entsprechende Gegenstück (Original oder Klon) zu finden. Dabei werden die kompletten Objekte der Implementierung beziehungsweise die Objekte der gemappten Komponenten in einer Schleife durchsucht. Eine weitere Verzögerung kann auftreten, wenn eine Komponente mit vielen Unterkomponenten gemappt wird, da zum einen jede Unterkomponente durchgegangen wird um die Textur zu ändern und zum anderen bestimmte Attribute beim Klonen verloren gehen. Diese Attribute müssen dem geklonten Objekt wieder manuell zugewiesen werden.

Fehlende Kollision
Wenn man eine neue Implementierungskomponente mappt, so prüft die Software nicht, ob diese Implementierungskomponente an eine bereits gemappte Komponente grenzt. Dies hat zur Folge, dass man diese nicht mehr unterscheiden kann, falls sich diese überlappen. Auch können Implementierungskomponenten über den Architekturkomponenten hinausragen. Insbesondere problematisch ist das beim Import eines Mappings, da dort immer die Mitte der Architekturkomponente als Position angegeben wird. Dies könnte man beheben, indem man die Position der gemappten Objekte beim Exportieren mitspeichert.

Darstellung der Architektur
Die Lesbarkeit der Namen der Architekturkomponenten verschlechtert sich bei steigender Tiefe des Graphen, da die Unterkomponenten bei jeder Schicht kleiner werden.

Aus diesem Grund muss man teilweise sehr nah an die Komponente kommen, um sie zu erkennen. Ab Schicht sechs ist es unmöglich die Architekturkomponente anhand des Titels zu identifizieren.

4 Evaluation

Die Evaluation nimmt einen wichtigen Stellenwert in der Arbeit ein, da hiermit festgestellt werden soll, wie sich die Implementierung in der Praxis unter realen Bedingungen eignet. Dies hat den Grund, da Menschen Individuen sind und dementsprechend verschiedene Fehler machen [8]. Mittels dieser Evaluationen sollen diese Fehler und Probleme erkannt und analysiert werden. Primär soll hierbei getestet werden, ob sich die Usability zwischen den Eingabevarianten mit der VR-Steuerung, beziehungsweise mit einer Maus und Tastatur Variante unterscheidet. Aus diesem Grund wird folgende *Forschungsfrage* aufgestellt:

> Haben verschiedene Interaktionsformen bei der Abbildung von Implementierungskomponenten auf Architekturkomponenten einen Einfluss auf die empfundene Usability, sowie gemessene Usability in Bezug auf Effizienz, Effektivität und Sicherheit?

Dabei wird die empfundene Usability mit dem SUS-Fragebogen gemessen, welcher in Kapitel 4.2.1 vorgestellt wird. Die Effizienz wird anhand der benötigten Dauer des Durchgangs gemessen. Die Effektivität bezeichnet die Korrektheit des angestrebten Ziels. Die Sicherheit hingegen gibt an, ob die Nutzer die Anwendung ohne externe Hilfe verwenden können. Die Unterschiede in den Interaktionsformen betrifft dabei die Hardware. In diesem Experiment wird eine handelsübliche Maus und Tastatur als Eingabegerät für die Desktopsteuerung und eine HTC Vive mit Controllern für die VR-Steuerung genutzt. Durch diesen Unterschied ergeben sich Unterschiede in der eigentlichen Interaktion der Anwendung. So wählt man Komponenten in der Desktopsteuerung durch das Anklicken mittels des Mauszeigers an und in der VR-Steuerung zielt man mittels eines Lasers auf die Komponente.

4.1 Hypothesen

Um diese Forschungsfrage beantworten zu können, werden folgende Hypothesen aufgestellt.

4.1.1 Haupthypothese

- **H1:** Es gibt einen Unterschied bei der empfundenen Gebrauchstauglichkeit zwischen verschiedenen Interaktionsformen.
- **H0:** Es gibt keinen Unterschied bei der empfundenen Gebrauchstauglichkeit zwischen verschiedenen Interaktionsformen.

4.1.2 Unterhypothese 1

- **H1:** Es gibt einen Unterschied bei der Effizienz zwischen verschiedenen Interaktionsformen.
- **H0:** Es gibt keinen Unterschied bei der Effizienz zwischen verschiedenen Interaktionsformen.

4.1.3 Unterhypothese 2

- **H1:** Es gibt einen Unterschied bei der Effektivität zwischen verschiedenen Interaktionsformen.
- **H0:** Es gibt keinen Unterschied bei der Effektivität zwischen verschiedenen Interaktionsformen.

4.1.4 Unterhypothese 3

- **H1:** Es gibt einen Unterschied bei der Sicherheit zwischen verschiedenen Interaktionsformen.
- **H0:** Es gibt keinen Unterschied bei der Sicherheit zwischen verschiedenen Interaktionsformen.

4.2 Studiendesign

Um eine Evaluation erfolgreich durchführen zu können war es davor nötig, davor ein entsprechendes Studiendesign zu erstellen. Bei dieser Evaluation handelt es sich um eine Feldstudie mit echten Benutzern. Da es sich hierbei um Software handelt, die primär für Experten (beispielsweise Software-Architekten) gedacht ist, musste sichergestellt werden, dass die Probanden grundlegende Kenntnisse von Software Architekturen haben. Dies wurde mit einer Einleitung (siehe Anhang) gewährleistet. Zudem wurde die Evaluation nach dem *within-subject Design* durchgeführt. Das heißt, dass jeder Nutzer sowohl die VR-Steuerung als auch die Tastatursteuerung getestet hat. Dies hat den Vorteil, dass man im Vergleich zum *between-subject Design* doppelt so viele Testergebnisse hat. Um mögliche Lerneffekte auszugleichen, startet die Hälfte der Probanden mit System A und die andere Hälfte mit System B.

4.2.1 Fragebogen

Es standen verschiedene Fragebögen zur Auswahl, um die empfundene Usability zu bewerten. Näher betrachtet wurden die folgenden drei Fragebögen.

Der System Usability Score (SUS) Fragebogen bewertet die Usability eines Systems anhand von zehn Fragen, die der Proband im Anschluss beantwortet. Daraus resultiert ein Wert zwischen 0-100 als Ergebnis, der die Gebrauchstauglichkeit des Systems bestimmen soll [9]. Vorteile dieses Fragebogens sind, dass dieser sehr gängig und leicht durchzuführen ist. Außerdem lässt sich durch das Ergebnis die Usability zwischen verschiedenen Systemen bewerten.

Ein weiterer, gängiger Fragebogen ist der NASA Task Load Index, auch NASA-TLX genannt. Im Unterschied zu dem SUS fokussieren sich die Fragen des NASA-TLX eher auf die mentale Belastung während der Durchführung, als auf die Usability. Zudem besteht dieser lediglich aus sechs Fragen, die auf einer Skala zwischen sehr niedrig und sehr hoch beantworten werden können [10].

Abschließend wurde der AttrakDIff betrachtet. Hierbei muss der Proband das Produkt anhand von 28 Adjektiven bewerten. Das Ergebnis ermittelt die pragmatische Qualität, hedonische Qualität und die Attraktivität [11]. Die Umfrage kann man direkt auf der Webseite www.attrakdiff.de durchführen und sich die Ergebnisse auswerten lassen. Die Ergebnisse werden hierbei grafisch veranschaulicht.

Da der SUS als einziger von den Fragebögen die allgemeine Usability bewertet, wurde entschieden, diesen Fragebogen in der Studie einzusetzen. Mithilfe des Resultats in Form des Score, kann man einen Durchschnittswert aller Probanden für jeweils beide Systeme berechnen, und diese dann vergleichen. Der Fragebogen ist im Anhang zu finden.

4.2.2 Aufbau

Die Evaluation wurde auf einem Rechner mit einem Intel Core i5-8400 Prozessor, einer GeForce GTX 1060 6GB Grafikkarte, 16GB Arbeitsspeicher und Windows 10 als Betriebssystem durchgeführt. Als VR-Gerät wurde die HTC Vive benutzt. Dabei ist zu erwähnen, dass nur ein Raumsensor zur Verfügung stand, was zu Verbindungsabbrüchen während der Verwendung führte.

4.2.3 Durchführung

Nachdem der Proband begrüßt wurde, wurde dieser darum gebeten, die Einverständniserklärung zu unterschreiben. Den Probanden wurde vor Beginn der Evaluation ein Text (siehe Anhang) gegeben, der das System beschreibt. Damit soll gewährleistet werden, dass die Probanden formell das gleiche Wissen über das Projekt und die gleichen Grundlagen von der Validierung von Software-Architekturen verfügen.

Daraufhin erhält der Proband eine Einweisung in die jeweilige Steuerungsmethode und beginnt anschließend einen Probedurchgang mit System A, um sich mit der Steuerung vertraut zu machen. Nach dem Probedurchgang absolvierte der Proband die Aufgaben mit dem System A. Nachdem der Durchgang mit System A abgeschlossen wurde, wurde dem Probanden der Fragebogen übergeben, um das System A zu bewerten. Der gleiche Prozess wurde mit dem System B wiederholt.

Während des Durchgangs wird zum einen die Zeit gemessen, die für das Erledigen aller Aufgaben benötigt wird. Aus dieser Zeit wird die Effektivität des Programms ermittelt. Außerdem wird nach jedem Durchgang überprüft, ob die gewünschten Ergebnisse erreicht wurden. Jede Komponente, die falsch gemappt wurde, wird als Fehler notiert. Außerdem wird notiert, ob ein Nutzer während der Durchführung Hilfe benötigt hat, um die gewünschten Ergebnisse zu erreichen. Daraus wird die Sicherheit des Programms abgeleitet.

4.2.4 Verwendete Systeme

Für die Studie wurde die Implementierung und die Architektur des Minilax-Compilers verwendet. Der Minilax-Compiler stellt einen Compiler für eine einfache Sprache dar, der Code für eine virtuelle Maschine generiert. Die Implementierungssprache ist C.

Die Implementierung besitzt 748 Knoten und 3273 Kanten. Die Architektur besitzt 27 Knoten und 41 Kanten. Als Layout für die Knoten wurde in SEE das Balloon Layout gewählt.

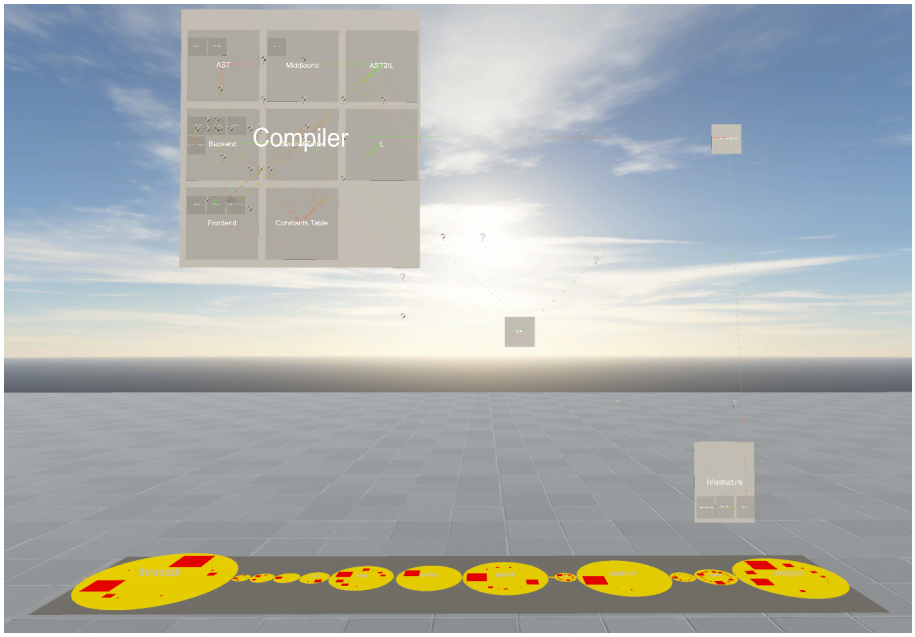


Abbildung 16: Ausgangslage der in der Evaluation verwendeten Systeme

4.2.5 Aufgaben

Jeder Proband hat eine vordefinierte Anzahl von Aufgaben zu erledigen. Dabei ist die Reihenfolge der Aufgaben innerhalb der zwei Teile irrelevant. Teil eins muss aber vollständig abgeschlossen werden, bevor Teil zwei begonnen werden kann. Die Aufgaben sind wie folgt definiert:

Erster Teil

- Mappe Implementierungskomponente *M argument* auf Architekturkomponente *Scanner*.
- Mappe Implementierungskomponente *M threaadr* auf Architekturkomponente *Main*.
- Mappe Implementierungskomponente *M symtable* auf Architekturkomponente *Global Controls*.
- Mappe Implementierungskomponente *M Codegen* auf Architekturkomponente *AST2Code*.
- Mappe Implementierungskomponente *M code list* auf Architekturkomponente *Target Program*.
- Mappe Implementierungskomponente *M scanner* auf Architekturkomponente *Scanner*.
- Mappe Implementierungskomponente *M parser* auf Architekturkomponente *Parser*.
- Mappe Implementierungskomponente *M constab* auf Architekturkomponente *Constants table*.

Zweiter Teil

- Remappe Implementierungskomponente *M argument* auf Architekturkomponente *Global Controls*.
- Remappe Implementierungskomponente *M threaadr* auf Architekturkomponente *IL*.
- Remappe Implementierungskomponente *M symtable* auf Architekturkomponente *Symbol Table*.

4.3 Experimentelle Variablen

Um die Forschungsfrage beantworten zu können, müssen experimentelle Variablen aufgestellt werden. Unabhängige Variablen stellen Faktoren dar, welche geändert werden können, um die abhängigen Variablen zu manipulieren. Zudem werden Umstände, welche für jeden Probanden möglichst gleich sein sollen, als kontrollierte Variablen bezeichnet [8]. Im Folgenden wird beschrieben, welche Variablen für das Experiment aufgestellt wurden.

4.3.1 Unabhängige Variablen

Die unabhängige Variable bei diesem Experiment ist die Eingabemethode. Diese ist ausgeprägt als VR-Steuerung oder Desktop-Steuerung.

4.3.2 Abhängige Variablen

Der SUS-Score stellt eine abhängige Variable dar, da mit dieser eine der Null Hypothesen bestätigt oder widerlegt werden soll. Da der SUS-Score überwiegend die empfundene Wahrnehmung bewertet [12], werden Metriken herangezogen, die weitere abhängige Variablen darstellen. Es wird die Fehlerrate gemessen, wobei als Fehler jegliche Eingabe gewertet wird, die nicht zum gewünschten Ziel der Aufgaben führt. Aus der Fehlerrate lässt sich dann die Effektivität bestimmen. Des Weiteren wird die Dauer des Durchgangs gemessen. Anhand der Dauer kann die Effizienz bestimmt werden. Um die Sicherheit zu ermitteln, wird abschließend noch erfasst, wie viele der Probanden Hilfe während des Durchgangs benötigt haben.

4.3.3 Kontrollierte Variablen

Die Studie wurde von allen Probanden unter möglichst gleichen Voraussetzungen durchgeführt. Es wurde während der kompletten Zeit sichergestellt, dass der Raum mit genügend Sauerstoff versorgt ist, kein Lärm zu hören ist und keine anderen äußeren Einflüsse den Probanden beeinflussen. Außerdem hat jeder Nutzer die gleiche Hardware mit den gleichen Einstellungen verwendet.

4.4 Ergebnisse

Nun folgen die Daten, die im Rahmen der Evaluation gesammelt werden konnten. Dafür werden zunächst die Probanden vorgestellt, die an der Studie teilnahmen. Anschließend werden die Messdaten vorgestellt und abschließend das qualitative Feedback zusammengefasst.

4.4.1 Teilnehmer

Insgesamt haben sechs Personen an der Evaluation teilgenommen, welche willkürlich aus dem privaten Freundeskreis ausgewählt wurden sind und allesamt keine Erfahrung mit der Anwendung hatten. Lediglich eine Person besaß keine VR Erfahrung. Vier Personen davon waren männlich und zwei weiblich. Das Durchschnittsalter beträgt 23 Jahre mit einer Standardabweichung von 1,27 Jahren. Drei Personen sind ehemalige Informatik Studenten. Die restlichen drei Personen haben keine Bezüge zur Informatik.

4.4.2 Quantitative Messergebnisse

Der durchschnittliche SUS-Score bei der Tastatur Steuerung liegt bei 80 Punkten, was einer hohen Benutzbarkeit entspricht [13]. Dabei benötigten die Probanden im Durchschnitt 09:16 Minuten für die Bearbeitung der Aufgaben. Fünf Probanden erledigten die Aufgaben dabei fehlerfrei. Ein Proband beging drei Fehler. Vier Probanden benötigten keine Hilfe zur Lösung der Aufgaben. Eine Person benötigte ein Mal Hilfe und eine Person benötigte zwei Mal Hilfe.

Im Vergleich dazu ergab der durchschnittliche SUS-Score bei der VR Steuerung 65,83 Punkte, was einer akzeptablen Benutzbarkeit entspricht [13]. Dabei benötigten die Probanden im Durchschnitt 10:15 Minuten für die Bearbeitung der Aufgaben. Vier Probanden erledigten die Aufgaben dabei fehlerfrei. Die restlichen zwei Probanden haben jeweils zwei Fehler begangen. Drei Probanden benötigten keine Hilfe zur Lösung der Aufgaben. Zwei Personen benötigten jeweils zwei mal Hilfe zur Lösung, und eine Person einmal.

	Mittelwert VR	**Standardabweichung VR**	**Mittelwert Tastatur**	**Standardabweichung Tastatur**
Frage 1	2,83	0,75	3	0,89
Frage 2	2,17	0,41	1,5	0,55
Frage 3	3,33	0,82	4,33	0,52
Frage 4	2,17	0,75	2	0,63
Frage 5	4,17	0,41	4	0,63
Frage 6	1,67	0,52	1,67	0,52
Frage 7	3,83	0,41	4,67	0,52
Frage 8	3,33	0,52	1,67	0,82
Frage 9	3,33	0,82	4,67	0,52
Frage 10	1,83	0,98	1,83	0,99
SUS-Score	65,83	0,17	80	0,2
Zeit	10:35	3:06	9:16	1:51
Fehler	0,94	0,94	0,6	1,1
Hilfe	0,83	0,9	0,5	0,76

Tabelle 1: Quantitative Messergebnisse der Evaluation

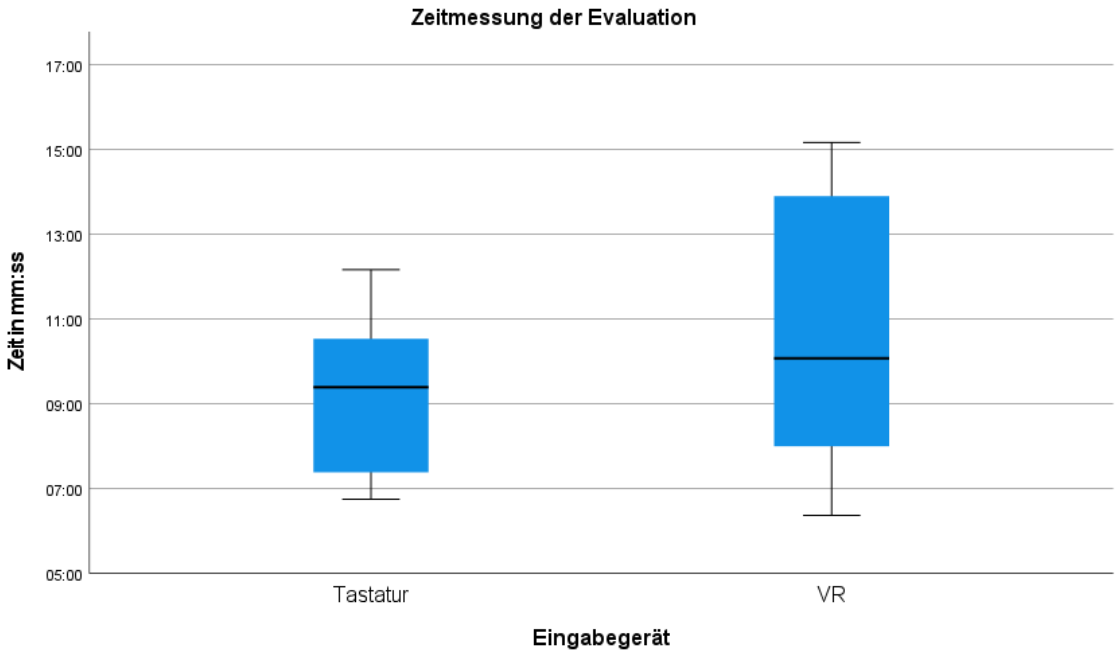


Abbildung 17: Boxplot der Zeitmessung während der Evaluation

Abbildung 18 zeigt einen Vergleich der Antworten. Für eine bessere Vergleichbarkeit wurde jede zweite Frage invertiert, da diese negativ formuliert sind. Dabei ist zu

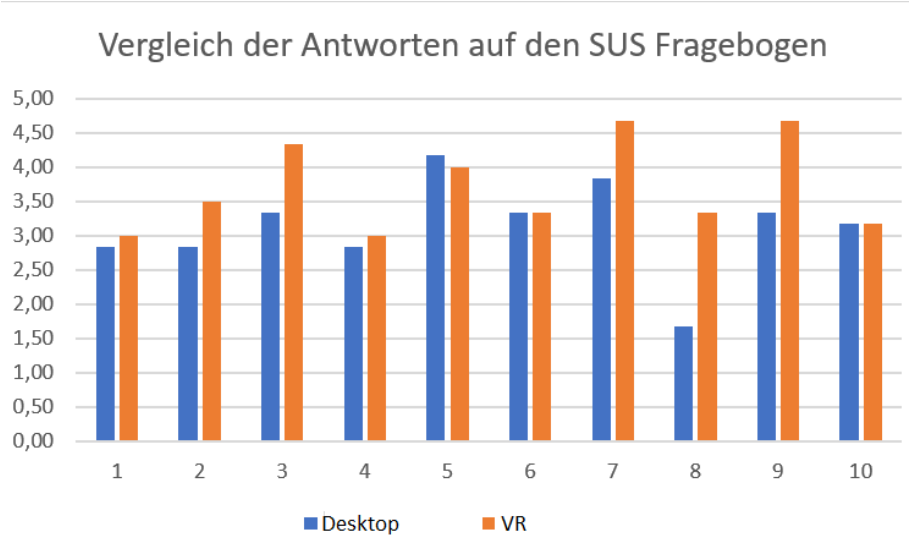


Abbildung 18: Vergleich der Antworten auf den SUS Fragebogen

erwähnen, dass der SUS-Fragebogen für eine bessere Robustheit bewusst redundante Fragen enthält [14]. Dabei kann man erkennen, dass die Antworten auf die Fragen meistens konsistent sind. Einen Ausreißer stellt dabei die Frage 8 (Ich empfinde die Bedienung nicht als sehr umständlich) bei der Desktop-Steuerung dar. Dort war der Mittelwert von 1,67 deutlich geringer als der allgemeine Mittelwert (3,13). Einen weiteren Ausreißer nach oben hin stellt Frage 5 (Ich finde, dass die verschiedenen Funktionen des Systems gut integriert sind) dar, bei welcher der Mittelwert 4,17 deutlich höher liegt.

Bei der VR-Steuerung sind die Antworten auf die Frage 7 (Ich kann mir vorstellen, dass die meisten Leute das System schnell zu beherrschen lernen) und Frage 9 (Ich habe mich bei der Nutzung des Systems sehr sicher gefühlt) mit einem durchschnittlichen Wert von 4,67 deutlich höher als der allgemeine Mittelwert von 3,7.

4.4.3 Qualitatives Feedback

Zusätzlich zum SUS-Fragebogen hatten die Probanden die Möglichkeit, eigene Anmerkungen und Verbesserungsvorschläge am Ende des Durchgangs anzugeben. Dabei wurden einerseits Anmerkungen bezüglich der Steuerung, als auch an der Visualisierung der Analyse abgegeben. Diese Kommentare wurden anhand ihrer Themenzugehörigkeit sortiert und im Folgenden aufgeführt.

1. Allgemeine Anmerkungen.
    - Gemappte Implementierungsklone können sich gegenseitig verdecken und sind schwer wiederzufinden.
    - Die Abstände zwischen Architekturkomponenten sind zu groß.
    - Eine Suchfunktion wäre hilfreich, um Komponenten schneller finden zu können.
    - Die 3D Welt stellt eine Verkomplizierung dar.
2. Anmerkungen zu der VR-Steuerung.
    - Die Verbindung ist oft abgebrochen.
    - Die Bewegung mit den Controllern ist schwergängig.
    - Die Namen der Komponenten (insbesondere die kleinen Architekturkomponenten) sind sehr schwer lesbar.
3. Anmerkungen zu der Desktop-Steuerung.
    - Die Maus Sensitivität beim Bewegen der Kamera ist sehr hoch.

4.5 Auswertung

Für die Überprüfung, ob die Daten normalverteilt sind, wird der Shapiro-Wilk-Test verwendet. Zur Prüfung der Unterschiedshypothesen wurde bei normalverteilten Daten der t-Test für abhängige Stichproben, und bei nicht-normalverteilten Daten der gepaarte Wilcoxon-Test, verwendet.

4.5.1 Prüfung der Hypothesen

Ausgehend aus den gesammelten Daten werden nun Schlussfolgerungen auf die aufgestellten Hypothesen gezogen.

Haupthypothese

- **H1:** Es gibt einen Unterschied bei der empfundenen Gebrauchstauglichkeit zwischen verschiedenen Interaktionsformen.
- **H0:** Es gibt keinen Unterschied bei der empfundenen Gebrauchstauglichkeit zwischen verschiedenen Interaktionsformen.

Test der empfundenen Gebrauchstauglichkeit anhand des SUS Scores

	Gepaarte Differenzen							
	Mittelwert	Standardabweichung	Standardfehler des Mittelwertes	95% Konfidenzintervall der Differenz		T	df	Sig. (2-seitig)
				Unterer Wert	Oberer Wert			
Tastatur - VR	11.67	3,41565	1,39443	8.0822	15.251	8.37	5	,000

Abbildung 19: Ergebnis des t-Tests für die Bewertung des SUS-Scores

Da die Normalverteilung gegeben ist, wird der t-Test verwendet. Es ergibt sich ein signifikanter Unterschied (p = 0,0004 < 0,05). Damit kann die H0-Hypothese verworfen und die H1 Hypothese angenommen werden.

Unterhypothese 1

- **H1:** Es gibt einen Unterschied bei der Effizienz zwischen verschiedenen Interaktionsformen.
- **H0:** Es gibt keinen Unterschied bei der Effizienz zwischen verschiedenen Interaktionsformen.

Test der Effizienz anhand der Durchführungsdauer

	Gepaarte Differenzen							
	Mittelwert	Standardabweichung	Standardfehler des Mittelwertes	95% Konfidenzintervall der Differenz		T	df	Sig. (2-seitig)
				Unterer Wert	Oberer Wert			
Tastatur - VR	-79.7	162.463	66,32529	-250.1613	90,82793	-1.2	5	,283

Abbildung 20: Ergebnis des t-Tests für die Bewertung der Effizienz

Da die Normalverteilung gegeben ist, wird der t-Test verwendet. Es konnte kein signifikanter Unterschied, in den Mittelwerten der Dauer je Testdurchgang, festgestellt werden. Aus diesem Grund wird die Nullhypothese beibehalten.

Unterhypothese 2

- **H1:** Es gibt einen Unterschied bei der Effektivität verschiedenen Interaktionsformen.
- **H0:** Es gibt keinen Unterschied bei der Effektivität zwischen verschiedenen Interaktionsformen.

Zusammenfassung des Wilcoxon-Tests bei verbundenen Stichproben

Gesamtzahl	6
Teststatistik	3,000
Standardfehler	1,837
Standardisierte Teststatistik	,000
Asymptotische Sig. (zweiseitiger Test)	1,000

Abbildung 21: Ergebnis des gepaarten Wilcoxon-Tests für die Bewertung der Effektivität

Da die Normalverteilung nicht gegeben ist, wird der gepaarte Wilcoxon-Test verwendet. Es konnte kein signifikanter Unterschied festgestellt werden. Aus diesem Grund wird die Nullhypothese beibehalten.

Unterhypothese 3

- **H1:** Es gibt einen Unterschied bei der Sicherheit zwischen verschiedenen Interaktionsformen.
- **H0:** Es gibt keinen Unterschied bei der Sicherheit zwischen verschiedenen Interaktionsformen.

Zusammenfassung des Wilcoxon-Tests bei verbundenen Stichproben

Gesamtzahl	6
Teststatistik	9,500
Standardfehler	3,623
Standardisierte Teststatistik	,552
Asymptotische Sig. (zweiseitiger Test)	,581

Abbildung 22: Ergebnis des gepaarten Wilcoxon-Tests für die Bewertung der Sicherheit

Da die Normalverteilung nicht gegeben ist, wird der gepaarte Mann-Whitney-U-Test verwendet. Es konnte kein signifikanter Unterschied festgestellt werden. Aus diesem Grund wird die Nullhypothese beibehalten.

4.5.2 Auswertung des qualitativen Feedbacks

Allgemein wurde häufig kritisiert, dass die Abstände der Architektur Komponenten zu groß waren und teilweise aus dem Sichtfeld verschwanden. Bezüglich der Architektur war noch ein häufig genannter Kritikpunkt, dass die Namen der Unterkomponenten teilweise so klein waren, dass man diese nicht lesen konnte.

Dies war besonders bei der Verwendung der Vive der Fall, da die Vive eine relativ niedrige Auflösung besitzt. Negativ angemerkt wurde zudem, dass die Klone der Implementierungskomponenten, die auf demselben Architekturknoten gemappt wurden sind, sich überlappen können. Außerdem wurde häufig der Wunsch genannt, eine Suche zu integrieren, um Implementierungs- oder Architekturkomponenten schneller finden zu können. Ein Proband empfand die Darstellung der Architektur und Implementierung in einer dreidimensionalen Welt als unnötig komplex.

Der häufigste Kritikpunkt bei der VR-Steuerung waren die häufigen Verbindungsabbrüche der Vive, was teilweise zu Schwindelgefühlen führte. Dies ist darauf zurückzuführen, dass nur ein Sensor zur Verfügung stand, statt den empfohlenen zwei. Außerdem hatten die Probanden teilweise Probleme mit der Steuerung.

Bei der Desktop Steuerung wurde einzig kritisiert, dass die Sensitivität zu hoch war.

Die Kritikpunkte waren fast alle nachvollziehbar und können in Zukunft verbessert werden. Da die Darstellung der Architektur für einen anderen Anwendungsfall konzipiert wurde, ist es nötig, diese speziell für diesen Fall anzupassen. Es wird im Moment an einer Spracheingabe gearbeitet, welche auch als Suchfunktion fungieren könnte. Bezüglich der VR-Steuerung kann nur wiederholt werden, dass es zur Zeit der Evaluation keine Möglichkeit gab, einen weiteren Vive Sensor zu organisieren und die Nutzung mit einem nicht empfohlen wird.
Besonders positiv wurde die leichte Erlernbarkeit und die visuelle Darstellung der Analyse hervorgehoben.

4.5.3 Schlussfolgerung

Es konnte bewiesen werden, dass die empfundene Gebrauchstauglichkeit bei der Verwendung der Tastatur Steuerung besser ist. Kein signifikanter Unterschied konnte bei der Effektivität, Effizienz und Sicherheit festgestellt werden. Insbesondere bei der Effizienz kann es jedoch sein, dass dies aufgrund der geringen Anzahl der Versuchspersonen der Fall ist, da sich die Mittelwerte an sich stark unterscheiden. Da jede Person die gleiche Aufgabe zwei Mal macht, ist ein Lerneffekt vorhanden. Die Daten, die im qualitativen Teil der Evaluation gesammelt worden, konnten die quantitativen Ergebnisse teilweise bestätigen. Auch hier konnte eine Priorisierung der Desktop Steuerung im Vergleich zur VR-Steuerung festgestellt werden. Außerdem enthielt das qualitative Feedback eine Vielzahl von Verbesserungsvorschlägen, die die allgemeine Gebrauchstauglichkeit der Software verbessern könnten.

4.6 Limitierungen

Aufgrund der Covid-19 Pandemie war keine Studie an der Universität möglich. Dementsprechend war es nötig, sich bei den Teilnehmern auf private Kontakte zu beschränken. Außerdem verfälschte die Tatsache, dass nur ein Vive Sensor zur Verfügung stand, das Ergebnis.

5 Fazit und Ausblick

Das SEE Projekt wurde in dieser Arbeit erfolgreich um eine visualisierte Software-Architekturprüfung erweitert. Zeitweise gab es Komplikationen bei der Implementierung. Dies hatte unter anderem den Hintergrund, dass die Arbeit auf zwei Projekten basiert, die aktiv in der Entwicklung sind und in denen laufende Änderungen gab. Dementsprechend musste die Implementierung daran angepasst werden. So wurde beispielsweise während der Implementierung das Konzept der Eingabegeräte geändert, was zusätzliche Arbeit bedeutete. Glücklicherweise wurde dies aufgrund der guten Kommunikation erleichtert, sodass Fragen und Probleme schnell geklärt werden konnten.

Darüber hinaus wurde durch die durchgeführte Evaluation belegt, dass ein Unterschied im subjektiven Wohlbefinden im Vergleich von verschiedenen Interaktionsformen besteht. Daraus kann geschlussfolgert werden, dass Nutzer eine Desktop-Steuerung der VR-Steuerung vorziehen. Hingegen konnte nicht belegt werden, dass es einen Unterschied in der Effektivität, Effizienz und Sicherheit im Hinblick auf die quantitativen Messzahlen gibt. Hierbei ist noch zu erwähnen, dass aufgrund der Covid-19 Pandemie keine Studie an der Universität durchgeführt wurde und sie deshalb auf den Bekanntenkreis eingegrenzt werden musste. Auch waren die Bedingungen nicht ideal, da die VR-Steuerung mit nur einem Sensor durchgeführt wurde.

SEE ist ein aktives Projekt, an dem auch nach Abgabe dieser Arbeit weitergearbeitet wird. Zu diesem Zeitpunkt werden parallel noch weitere Funktionen entwickelt, wie beispielsweise ein Multiplayer, weitere Eingabevarianten und Layouts. Eine *augmented reality* Variante ist geplant, welche mittels der Microsoft HoloLens 2 umgesetzt werden soll. Zudem wird an einer Sprachunterstützung gearbeitet, welche sich für das Mapping eignen könnte, um Komponenten schneller finden zu können. Diese zusätzlichen Funktionen müssten gegebenenfalls an diese Arbeit angepasst werden.

Zu diesem Zeitpunkt fehlt noch die Unterstützung von Gamepads, die in dem normalen SEE Projekt gegeben ist. Denkbar wäre auch das Mapping der Komponenten nicht mehr per Auswahl zu tätigen, sondern die Implementierungskomponenten direkt zu verschieben. Eine mögliche Erweiterung besteht zudem in der Integration der Architekturmodellierung, sodass man während des Mappings auch Teile der Architektur ändern kann. Verbesserungswürdig ist außerdem die Performance bei sehr großen Graphen und vielen abgebildeten Komponenten.

LITERATUR

[1] IEEE. IEEE standard glossary of software engineering terminology. *IEEE Std 610.12-1990*, 1990.

[2] Rainer Koschke. Incremental reflexion analysis. In *2010 14th European Conference on Software Maintenance and Reengineering*, pages 1–10, 2010.

[3] Len Bass, Paul Clements, and Rick Kazman. *Software Architecture in Practice*. Addison-Wesley Professional, 3rd edition, 2012.

[4] Jeffery Clinton. The city metaphor in software visualization. 2019.

[5] Richard Wettel and Michele Lanza. Codecity: 3d visualization of large-scale software. 2008.

[6] Suzanne Robertson and James Robertson. Mastering the requirements process: Getting requirements right, 2012.

[7] David Ameller, Claudia Ayala, Jordi Cabot, and Xavier Franch. How do software architects consider non-functional requirements: An exploratory study. *2012 20th IEEE International Requirements Engineering Conference (RE)*, 2012.

[8] I. Scott MacKenzie. *Human-Computer Interaction - An Empirical Research Perspective*. Newnes, London, 2012.

[9] John Brooke. *SUS - A quick and dirty usability scale*. 1990.

[10] NASA. *Nasa Task Load Index (TLX) v. 1.0 Manual*. 1986.

[11] Marc Hassenzahl, Franz Koller, and Michael Burmester. *Der User Experience (UX) auf der Spur: Zum Einsatz von www.attrakdiff.de*. 2008.

[12] Mandy R. Drew, Brooke Falcone, and Wendy L. Baccus. *What Does the System Usability Scale (SUS) Measure?* 2018.

[13] Sami Binyamin, Malcolm Rutter, and Sally Smith. *The Utilization of System Usability Scale in Learning Management Systems: A Case Study of Jeddah Community College*. 2016.

[14] Attila Vertesi, Huseyin Dogan, Angelos Stefanidis, Giles Ashton, and Wendy Drake. Usability evaluation of a virtual learning environment: A university case study. 2018.

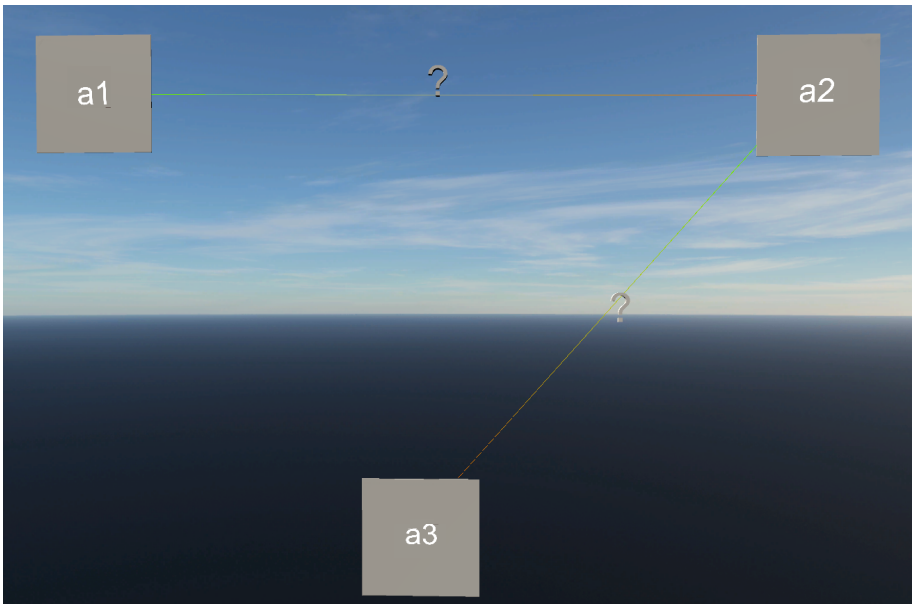
ANHANG

Text Evaluation

Bei diesem Projekt handelt es sich um SEE. SEE bedeutet *Software Engineering Experience* und hilft dabei, die Strukturen von Programmen anhand ihrer Komponenten und deren Beziehungen zu visualisieren. Dieser Quellcode wird mithilfe einer Stadtmetapher veranschaulicht, wobei jedes Gebäude eine Datei darstellt. Dabei nutzen Dateien teilweise Inhalte von anderen Dateien. Sie sind also abhängig voneinander. Diese Abhängigkeiten werden mit Kanten dargestellt, die jeweils über den Gebäuden beginnen und enden.

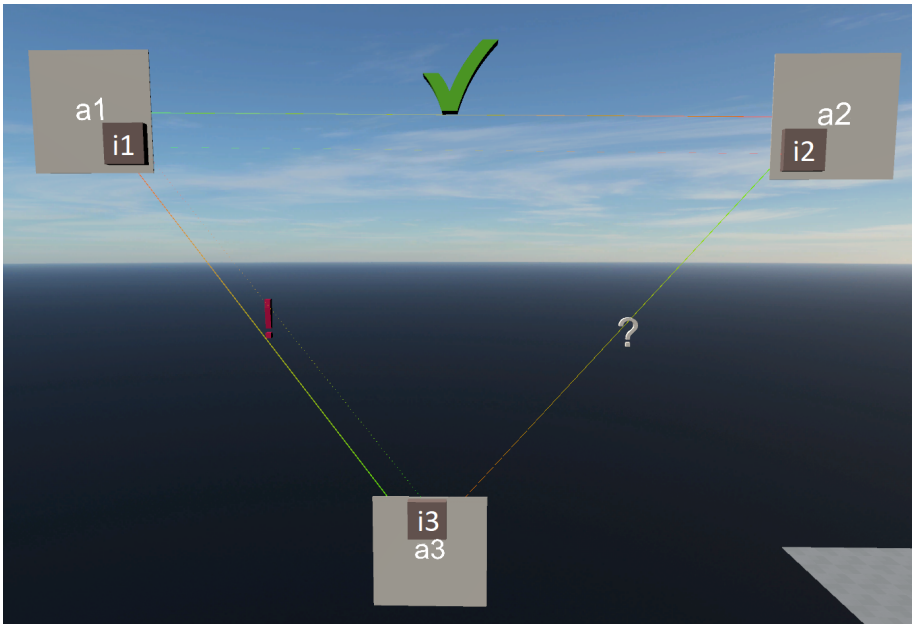
Um komplexe Software Systeme planen zu können, ist es nötig, Architekturen zu erstellen. Architekturkomponenten haben meist logische und verständliche Namen, da diese passende Dateien gruppieren. Dabei ist es aber trotzdem möglich, dass Dateien, die nicht aus einer Architekturkomponente stammen, voneinander abhängig sind. Der Software Architekt gibt für jede Architekturkomponente an, welche Abhängigkeiten erlaubt sind.

Die Software erlaubt es, die Implementierungskomponenten auf die Architektur abzubilden und die Auswirkung dieser Abbildung darzustellen. Jede Kante der Architektur hat einen Zustand, der entweder als Fragezeichen, Ausrufezeichen oder Haken dargestellt wird. Wenn eine Abhängigkeit in der Architektur spezifiziert, aber von der Implementierung ungenutzt ist, ist diese *absent* und es erscheint ein Fragezeichen über der Kante.



Ein Haken über der Kante erscheint, wenn diese *convergent* ist. Convergent bedeutet, dass eine Abhängigkeit in der Architektur spezifiziert wurde und diese auch von der Implementierung genutzt wird. Nutzt die Implementierung allerdings eine Abhängigkeit, die in der Architektur nicht spezifiziert

wurde, ist diese divergent. Es wird eine neue Kante in der Architektur erstellt, welche allerdings mit einem Ausrufezeichen markiert ist. In Abbildung 5 kann man die drei Status sehen.



Nun folgen deine Aufgaben: (hier kommen die Aufgaben aus 4.2.5)

Einverständniserklärung

Name, Vorname: _____

Hiermit erkläre ich mich einverstanden, dass im Rahmen der Studie Daten über mich gesammelt und anonymisiert aufgezeichnet werden. Außerdem stimme ich zu, dass die Ergebnisse zu Studienzwecken weiterverwendet werden dürfen. Es wird gewährleistet, dass meine personenbezogenen Daten nicht an Dritte weitergegeben werden.

_____

Datum, Unterschrift

	STIMME GAR NICHT ZU				STIMME VOLL ZU
Ich kann mir sehr gut vorstellen, das System regelmäßig zu nutzen.					
Ich empfinde das System als unnötig komplex.					
Ich empfinde das System als einfach zu nutzen.					
Ich denke, dass ich technischen Support brauchen würde, um das System zu nutzen.					
Ich finde, dass die verschiedenen Funktionen des Systems gut integriert sind.					
Ich finde, dass es im System zu viele Inkonsistenzen gibt.					
Ich kann mir vorstellen, dass die meisten Leute das System schnell zu beherrschen lernen.					
Ich empfinde die Bedienung als sehr umständlich.					
Ich habe mich bei der Nutzung des Systems sehr sicher gefühlt.					
Ich musste eine Menge Dinge lernen, bevor ich mit dem System arbeiten konnte.					