Grafische Darstellung der Metriken des SEE City Projekts

Bachelorarbeit

Robert Bohnsack

Matrikelnummer: 4355279

06.08.2020

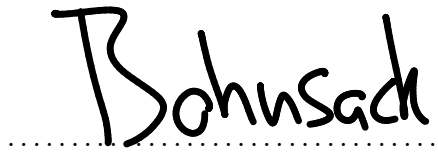
Fachbereich Mathematik / Informatik
Studiengang Informatik

1. Gutachter: Prof. Dr. rer. nat. Rainer Koschke
2. Gutachter: Prof. Dr. Gabriel Zachmann

Erklärung

Ich versichere, die Bachelorarbeit 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.

Bremen, den 06.08.2020



(Robert Bohnsack)

Danksagung

Hiermit möchte ich mich herzlich bei allen bedanken, die mir beim Implementieren und Ausarbeiten dieser Arbeit unter die Arme gegriffen haben.

Zu aller erst möchte ich mich bei Aytac bedanken, der mich vor allem in den ersten Wochen viel über neue Ideen nachdenken lassen hat und immer bereit war sich meine Prototypen anzuschauen.
Dank geht auch an alle, die mir trotz der Krise ihre Zeit für eine Evaluation geschenkt haben. Das hat mir sehr geholfen!
Da die Rechtschreibung mir nicht so sehr liegt wie manch anderen, muss ich mich natürlich auch ausdrücklich bei meinen Korrekturlesern bedanken!
Zu guter Letzt möchte ich mich natürlich bei meinen Prüfern Prof. Dr. Rainer Koschke und Prof. Dr. Gabriel Zachmann für den Aufwand der Bewertung einer Bachelorarbeit, aber auch die Begleitung und Hilfestellung beim erreichen dieses Ergebnisses bedanken.

Ohne euch wäre diese Arbeit wohl nicht so zustande gekommen.
Vielen Dank!

INHALTSVERZEICHNIS

GLOSSAR

Dictionary Eine Liste von Keys (Schlüssel), das sind einzigartig identifizierbare Objekte, zu welchen Values (Werte) gespeichert werden. So kann schnell der Wert eines Schlüssels abgefragt werden. 1, 25

Framework Ein Framework ist eine Plattform zur Entwicklung von Software Applikationen. Es bietet eine Grundlage auf welcher Entwickler Programme für bestimmte Plattformen bauen können. Zum Beispiel kann ein Framework Klassen und Funktionen enthalten, welche zur Verarbeitung von Eingaben, Management von Hardware Geräten oder der Interaktion mit System Software genutzt werden können. Das vereinfacht den Entwicklungsprozess, da Programmierer nicht mit jeder neuen Anwendung grundlegenden Code neu schreiben müssen[1] (vgl. (Christensson, 2020, Web: /definition/framework)). 1, 9

GameObject Unity Technologies (2019) beschreibt GameObjects als fundamentale Objekte, welche Charaktere, Requisiten und die Kulisse darstellen. Selbst haben sie wenige Funktionen und dienen eher als Kontainer für Komponenten, welche die richtige Funktionalität implementieren[1]. 1, xi, 23, 27, 29

Knoten Ein Knoten in einer Software-Stadt wird als Gebäude oder Straße visualisiert und stellt zum Beispiel eine Datei oder ein Verzeichnis aus der Software dar. 1, 27

Metrik (Software-)Metriken werden vom IEEE (1993) als eine Funktion, die Software Daten als Eingabe zu einem einzelnen numerischen Wert als Ausgabe umwandelt, welcher genutzt werden kann, um die Qualität der Software in einem Bereich zu bestimmen[1], beschrieben. Das kann zum Beispiel die Anzahl der Zeilen an Code sein. 1, 3, 4, 5, 11, 12, 13, 15, 17, 20, 21, 30

Prefab Prefabs werden in der Unity Engine als eine Art Bausteine genutzt. Anstatt ein GameObject für jede Szene neu zu erstellen, kann dies ein mal als Prefab angefertigt werden um es mehrmals wiederzuverwenden. 1, 10, 26, 29, 47

Skript Ein Computer Skript ist eine Liste von Befehlen, welche von einem bestimmten Programm oder Engine ausgeführt werden können. Skripte sind normalerweise Textdateien,

[1] Durch den Verfasser aus dem Englischen übersetzt

welche in einer bestimmten Programmiersprache geschrieben sind. Sie können mit einem Texteditor geöffnet und bearbeitet werden. Werden sie von einer fähigen Engine geöffnet, so können die Befehle ausgeführt[1] (vgl. (Christensson, 2020, Web: /definition/script)). 1, 27, 29, 30

Szene Szenen stellen die Umgebungen und Menüs in Unity dar. Jede Szene kann als ein eigenes Level mit Landschaften, Hindernissen, Dekorationen oder, im Falle dieser Arbeit, Städten gesehen werden. 1, xi, 23

ABKÜRZUNGSVERZEICHNIS

AGST Arbeitsgruppe Softwaretechnik. 1, 3

CSV Comma-separated values. 1, 3

GXL Graph eXchange Language. 1, 3

HMD Head Mounted Display. 1, 6, 20, 43, 51

SEE Software Engineering Experience. 1, ix, 1, 2, 3, 4, 5, 6, 10, 11, 12, 13, 22, 51, 52

SUS System Usability Scale. 1, 36, 37

UI User Interface. 1, 12, 14

UML Unified Modeling Language. 1, 4, 5, 22, 24, 25, 26, 59

VR Virtual Reality. 1, viii, ix, 6, 12, 18, 20, 21, 22, 26, 30, 32, 34, 35, 36, 37, 39, 40, 41, 42, 43, 46, 47, 48, 49, 51, 59

KAPITEL 1

Einleitung

Die Welt befindet sich im Wandel. Mehr und mehr Berufe werden durch Maschinen oder Software ersetzt und aus dem Alltag ist das Smartphone für die meisten nicht mehr wegzudenken. Mit etlichen Bibliotheken und Entwicklungsumgebungen welche es auf dem Markt gibt, ist es heute einfacher denn je auch im Alleingang Software zu entwickeln. Trotzdem ist es üblich, dass seriöse Software von riesigen Teams geschrieben wird, welche untereinander eine ausgesprochene Koordination brauchen um effizient und erfolgreich zu entwickeln. Nicht selten nimmt die Planung und Umsetzung der Produkte Ausmaße der Planung und Konstruktion von Städten an.
Genau darauf baut das Projekt Software Engineering Experience (SEE) der Arbeitsgruppe Softwaretechnik (AGST) auf. Mit diesem ist es möglich, bestehende Software als virtuelle Städte darzustellen. Etliche Informationen welche aus der Ordnerstruktur und dem darin enthaltenen Quellcode ausgelesen werden, können so in den Aufbau der Stadt einfließen. Je nach Wert einer bestimmten Metrik verändert sich die Höhe, Breite oder eine andere Variable eines zu einer Datei gehörigen Gebäudes. Gebäude stehen an Straßen, welche den Ordner, in dem die zum Gebäude gehörige Datei gespeichert ist, widerspiegelt. Eine solche Darstellung soll es Entwicklern unter anderem ermöglichen, einen schnelleren Überblick über die Projekte an denen sie arbeiten zu erlangen, oder gar neue Software anhand der Stadt-Metapher als Team zu planen.

1.1 Ziel der Arbeit

Die Art wie die Städte des SEE Projektes bei der schnellen Analyse von komplizierter Software helfen, war bereits effizient. Auch war sie aber nicht die finale Vision dessen, was möglich sein kann. Um dieser Vision einen Schritt näherzukommen sollte das Projekt um eine weitere Darstellungsebene ergänzt werden, welche in der klassischen Softwareanalyse Anwendung findet. Die Stärken und Schwächen dieses Ansatzes sollten sich mit denen der Darstellung als Stadt ergänzen. Unter anderem, weil viele Menschen bereits mit einem Umgang damit vertraut sind, wurde eine Visualisierung über Graphen gewählt. Die Hauptfunktion der Graphen sollte sein, die Knoten oder Gebäude der Stadt nach den Werten ihrer Metriken auf zwei Ach-

sen aufzulisten. Vorher waren in der Stadt nur die Metriken erkennbar, welche durch Höhe, Breite, Tiefe oder Farbe der Gebäude oder Straßen dargestellt wurden. Mit den Graphen wäre es außerdem möglich alle anderen numerischen Metriken, welche aus dem Quellcode ausgelesen werden konnten, auf den Achsen der Graphen darzustellen. Auch sollten die Ergebnisse im Graphen interaktiv sein. Zum Beispiel um das zugehörige Gebäude zu einem Eintrag im Graphen anzuzeigen oder mehrere Ergebnisse auszuwählen um die genauen zugehörigen Werte der dargestellten Metriken anzuzeigen. Während dieser Arbeit sollte eine solche Art von Graphen implementiert und evaluiert werden.

1.2 Aufbau der Arbeit

In den nächsten Abschnitten dieser Arbeit wird der Ablauf zum Erreichen der Ziele[1] näher erläutert. Um jedem Leser eine solide Grundlage für das Verständnis dieser Arbeit zu bieten, wird im Kapitel „Grundlagen“ eine grobe Übersicht über vorherige Arbeiten und existierende Techniken auf dem Gebiet gegeben. Wie die Funktionen der Software vor der Implementierung geplant wurden, wird in Kapitel „Planung“ erläutert. In Kapitel „Implementierung“ soll erörtert werden, wie mit der Unity Engine[2] vom Benutzer bedienbare Graphen zum SEE Projekt hinzugefügt wurden. Im darauf folgenden Kapitel „Evaluation“ soll die Effizienz der neuen Funktionen anhand von Nutzertests analysiert werden. Im Kapitel „Diskussion“ wird noch einmal über die Arbeit im Ganzen reflektiert, um die Ergebnisse dieser genauer zu Interpretieren.

[1] siehe Kapitel 1.1
[2] siehe Kapitel 2.2.3

KAPITEL 2

Grundlagen

Für diese Arbeit wurden einige existierende Systeme verwendet, um die Implementierung zu erleichtern. Außerdem gab es zur Softwarevisualisierung als Stadt bereits Arbeiten. Um allen Lesern einen groben Überblick über die verwendeten Techniken und vorherigen Arbeiten zu geben, werden die wichtigsten Aspekte in diesem Kapitel kurz erklärt und zusammengefasst.

2.1 Was ist SEE City

Mit dem SEE Projekt versucht die AGST der Universität Bremen die bisherigen Möglichkeiten der Software Visualisierung[1] zu erweitern. Das Projekt basierte ehemals auf der Unreal Engine und wurde inzwischen zur Unity Engine[2] portiert.

2.1.1 Übersicht der Funktionen

Mit SEE wird Nutzern inzwischen einige Möglichkeiten geboten, Software als Städte-Metaphern darzustellen und selber zu bauen. Einige dieser Funktionen hatten während dieser Arbeit jedoch noch Entwicklungsstatus.

Darstellung von Software als Stadt

Die Hauptfunktion von SEE ist es, sämtliche Daten zu einem bestehenden Softwareprojekt aus vom Quellcode generierten Graph eXchange Language (GXL) und Comma-separated values (CSV) Dateien auszulesen und daraus eine Stadt in verschiedenen Stilen zu generieren. Den Gebäuden, welche Blattknoten im Dateisystem (also die wirklichen Dateien, in denen der Code steht) darstellen, können unterschiedliche Metriken für deren Höhe, Breite, Tiefe und Farbe zugewiesen werden. Sie skalieren also entsprechend der gewählten Metrik. Alle anderen Knoten des Dateisystems (also die Ordner oder Pakete, in denen die Dateien gespeichert sind) werden nach einem gewählten Layout angeordnet. Sie stellen in der Stadt die Straßen,

[1] siehe Kapitel 2.2.1
[2] siehe Abschnitt 2.2.3

an denen die Gebäude gebaut sind, dar. Als Layouts stehen zum Beispiel das „Evo Streets“ Layout von Steinbrückner (2013) zur Auswahl. Bei diesem befindet sich in der Mitte der Stadt eine „Hauptstraße“ welche den Hauptordner eines Projektes darstellen. Von dieser gehen kleinere „Nebenstraßen“ ab, welche die Unterordner darstellen. Andere Layouts welche zur Auswahl stehen sind unter anderem das „Rectangle Packing“ Layout, bei welchem Straßen ineinander verschachtelt angeordnet werden, und viele weitere.

Dynamische Darstellung der Software als Stadt

In dieser Darstellung wird eine Software während ihrer Laufzeit dargestellt. Das funktioniert ähnlich wie die normale Darstellung als Stadt. Es werden jedoch zusätzlich zwischen den Gebäuden Linien gezogen, welche Aufrufe einer in der zu dem Gebäude gehörigen Datei gespeicherten Methode visualisieren. Je nach Häufigkeit der Aufrufe verändert sich die Farbe der Gebäude, um einen guten Überblick über potenzielle Problemzonen und ähnlichem zu erlauben.

Animation der Evolution von Software

Hat man ein Softwareprojekt über einen längeren Zeitraum entwickelt, so ist es oft interessant diese Entwicklung zu visualisieren. Mit SEE lässt sich solch eine Evolution darstellen, indem die Änderungen zwischen mehreren Versionen animiert werden.

2.2 Stand der Forschung

2.2.1 Software Visualisierung

Die Software Visualisierung beschreibt den Prozess Informationen eines bestehenden Softwaresystems grafisch darzustellen. Dabei wird auf Informationen der Softwarearchitektur des Quellcodes oder auf Metriken des Laufzeitverhaltens zugegriffen um statische, interaktive oder dynamische Repräsentationen in 2D oder 3D zu erschaffen. Nach Trümper (2014) vereinfacht bietet die Softwareanalyse und Visualisierung ihren Nutzern leistungsstarke, interaktive Mittel. Diese ermöglichen es Aufgaben zu automatisieren und insbesondere wertvolle und belastbare Einsichten aus den Rohdaten zu erlangen.

Aufbau von Software

Den Grundstein dieser Arbeit bildet die Software Visualisierung. Bereits im vorherigen Projekt in der Unreal Engine und auch im aktuellen Projekt in Unity[3] ist diese der Hauptnutzen

[3] siehe Abschnitt 2.2.3

des SEE Projektes. Die Software Visualisierung soll es unter Anderem neuen Entwicklern in einem Team erleichtern einen besseren Überblick über den Quellcode zu erhalten. Ein anderer beispielhafter Nutzen wäre es, Schwachstellen zu finden, an welchen die Software optimiert werden kann, da Software heutzutage schnell enorme Ausmaße annimmt.
Die vorherigen Arbeiten, welche zu diesem Forschungsthema gefunden wurden, haben zwar alle schon einige Jahre seit der Fertigstellung gesehen, jedoch scheint das Gebiet noch relativ unerforscht zu sein. Nichtsdestotrotz kann diesen Arbeiten einiges entnommen werden. Das Offensichtlichste was viele Arbeiten zur Software Visualisierung gemeinsam haben ist, dass in den meisten fällen eine hierarchische Struktur dargestellt wird. In der Objektorientierten Programmierung sind viele Sprachen mit Paketen und Vererbungen hierarchisch aufgebaut, und Entwicklungsumgebungen bauen deshalb oft ein hierarchisches Dateisystem auf.
Es gibt bereits einige Darstellungsarten, welche unter Entwicklern gängige Begriffe sind. Dazu zählt zum Beispiel die Unified Modeling Language (UML), welche anhand von verschiedenen Diagrammarten statische und dynamische Aspekte einer Software abbildet. Allerdings sind diese Darstellungsarten oft ungewohnte Metaphern, welche das schnelle Verständnis erschweren. Deshalb wurde in dieser, wie auch einigen der bereits bestehenden Arbeiten eine Darstellung als Stadt gewählt. Eine jedem bekannte Umgebung, welche tagtäglich navigiert wird. Die bestehenden Arbeiten gleichen sich meistens darin, dass Pakete als Bezirke oder Straßen dargestellt werden und Klassen (meist Dateien) als Gebäude. Der Unterschied liegt dann meist in den Informationen, welche dargestellt werden.
Da dies eine gute Einführung zum Verständnis der Arbeit ist, wird in folgenden auf Alleinstellungsmerkmale der vorangehenden Arbeiten eingegangen.

Bisherige Darstellungsarten

Unter den bekannten Darstellungsarten von Software, gibt es zwei die genauer erläutert werden sollten.

Unified Modeling Language (UML) Die Object Management Group (2017) beschreibt ihre UML als „graphical language for visualizing, specifying, constructing, and documenting the artifacts of distributed object systems“.
Mit der UML wurde ein standardisierter Satz an Zeichen, Symbolen und Notationen entworfen, mit welchem es Entwicklern möglich ist den Aufbau oder Abläufe ihrer Software zu visualisieren. Dafür bestehen verschiedene Diagrammtypen, welche später[4] noch einmal genauer erläutert werden. So kann Software vor der Implementierung geplant, oder nach der Implementierung dokumentiert werden.

[4] siehe Abschnitt 4.3.2

Tree Maps Sehr viel näher ist SEE an den Tree Maps von Johnson und Shneiderman (1991). Diese stellen auf einer gegebenen Fläche hierarchische Daten dar, indem sie den vorhandenen Platz unter den anzuzeigenden Elementen verteilen. Ein Element erhält dabei so viel Platz, wie der Wert der darzustellende Metrik für dieses Element proportional zur Summe der Werte aller Elemente ausfällt. Ein Beispiel wäre, Dateien nach ihrer Größe auf der Festplatte darzustellen. Ist die Festplatte 100 Gigabyte groß, so nimmt eine zehn Gigabyte große Datei genau zehn Prozent der Anzeigefläche ein. Oft werden die dargestellten Daten eingefärbt, um eine weitere Metrik darzustellen. Aus diesem Prinzip entwickelten sich später andere Darstellungen, wie die Darstellung als Stadt.

Evo Spaces (Alam und Dugerdil, 2007)

Wo SEE bisher nur so weit geht, einzelne Klassen als Gebäude anzuzeigen, sind die Autoren von Evo Spaces einen weiteren Schritt gegangen und haben den Blick in die Gebäude ermöglicht. Damit war es ihnen möglich, einzelne Methoden oder Funktionen anhand der Etagen als Arbeiter in diesen darzustellen. Außerdem wurde eine „Nachtsicht“ implementiert, welche die dynamischen Informationen einer Software anhand von Lichtstrahlen von einem Gebäude zum anderen visualisiert. Oft aufgerufene Gebäude werden dabei heller dargestellt. Ein ähnliches Feature ist in SEE in Entwicklung.

SynchroVis (Waller u. a., 2013)

Die Autoren von SynchroVis haben sich auf das Problem konzentriert, dass gleichzeitig auftretende Informationen in vorhandenen Darstellungsarten kompliziert zu analysieren sind. Diese Informationen können zum Beispiel beim Multithreading auftreten. Waller u. a. (2013) gehen hier so vor, dass Gebäude mehrere Etagen haben. Die erste Etage stellt eine Klasse an sich dar. Alle weiteren Etagen stellen Instanzen dieser Klasse dar. Wird ein Aufruf getätigt, so wird dieser als eine Straße dargestellt, wobei alle Straßen einer Farbe zu einem Thread gehören, welcher für den Aufruf verantwortlich ist. Führt eine Straße zum Erdgeschoss eines Gebäudes, so handelt es sich um einen statischen oder Konstruktor-Aufruf. Über Verbindungen von Dach zu Dach werden statische Relationen wie Vererbungen dargestellt. Zur besseren Veranschaulichung von einzelnen Threads wird für diese ein eigenes Gebäude erzeugt, welches pro Thread eine Etage hat, von welcher dieser ausgeht. So können auch bereits gestoppte Threads weiter dargestellt werden. Warteoperationen zur Synchronisierung zwischen Threads werden mit gestrichelten Linien angezeigt, welche auf separate wait/notify beziehungsweise Semaphoren/Monitor Gebäude zeigen.

2.2.2 Virtual Reality

Von Burdea und Coiffet (2003) wird die Virtual Reality (VR) als realistisch aussehende Welt, welche durch Computer Grafik erzeugt wird und mit der in Echtzeit interagiert werden kann, beschrieben. Das bedeutet, dass der Computer Eingaben des Nutzers erkennen kann, um die Simulation sofort an diese anzupassen. Auch erklären Burdea und Coiffet (2003), dass VR oft mit den häufig dafür genutzten Geräten in Verbindung gebracht wird, was damals Head Mounted Displays (HMDs) und Sensorhandschuhe waren.
Wenn heutzutage die Rede von VR ist, so wird fast ausschließlich über den Einsatz von HMDs geredet. Dazu gibt es einiges an Eingabelösungen, wobei der Trend eher in Richtung Controller welche in jeder Hand gehalten werden verläuft.
In SEE wird genau dieser Aufbau eingesetzt. Das HMD wird im Raum getracked, um die Position des Spielers in der Stadt festzustellen. Zusätzlich können die Controller genutzt werden, um in die gezeigte Richtung zu „fliegen“. So können größere Distanzen, welche über den getrackten Bereich hinaus laufen würden, zurückgelegt werden.

2.2.3 Game Engines

Die Spiele Entwicklung für sich ist ein Gebiet mit vielen Facetten und Richtungen in denen Entwickler sich Spezialisieren können. Von der Spiellogik über korrekte physikalische Darstellung bis hin zum Rendern um das Spiel auf dem Bildschirm darzustellen. Auch auf Musikalischer und Audio-technischer Ebene ist die Entwicklung mit Soundtracks, Waffensounds oder Umgebungsgeräuschen ausgeprägt. Obwohl sich alle Spiele unterscheiden laufen im Hintergrund immer wieder die gleichen Prozesse ab. Vor ein paar Jahren mussten diese für neue Spiele jedes mal mitentwickelt werden. Das wird heute von Game Engines übernommen. Diese können als eine Art Framework für Spiele bezeichnet werden. Sie bieten also einfachen zugriff auf häufig genutzte aber komplexe Funktionen. Möchte man zum Beispiel einen Schuss links vom Spieler erklingen lassen, so muss nur der Engine die Position des Spielers und die der Audioquelle mitgeteilt werden, und diese übernimmt den Rest.
In der Welt der Game Engines gibt es eine große Auswahl. GameMaker: Studio von YoYo Games bietet einen leichten Einstieg in die Entwicklung, sodass Nutzer ohne Code zu schreiben Spiele erschaffen können, verzichtet aber auf einiges an Funktionalität. Die Unreal Engine bietet dafür einen viel größeren Umfang, fordert aber einiges an Einarbeitung und das Schreiben von Code. Ein Mittelstück dazu, welches durch leichte Bedienung und eine große Community schneller zu erlernen ist, bietet die Unity Engine.
Aus dem Grund, dass komplexere Engines mit ihren Untersystemen ein großes Spektrum an Einsatzmöglichkeiten in der Spieleindustrie anbieten, haben sich inzwischen auch Architekten und Filmstudios diese zunutze gemacht. Natürlich entstanden viele Animationsfilme in Unreal und Unity, aber schaut man sich Projekte wie „Star Wars: Mandalorian“[5] an, so erfährt man

[5] siehe https://youtu.be/gUnxzVOs3rk

die ersten Schritte in eine Zukunft, in der Filmtechnisch alles bis auf die Schauspieler virtuell erzeugt werden kann um trotz übermäßigen Einsatz von Computer generierten Umgebungen und Effekten ein realistisches Bild zu erzeugen.

Die Unity Engine

Ohne eine Engine wie Unity wäre diese Arbeit nicht möglich. Die Engine wird seit 2005 von Unity Technologies entwickelt und stets verbessert. Unity wird, gleich den anderen bekannten Game Engines wie zum Beispiel Unreal, hauptsächlich als Entwicklungsumgebung für Spiele sämtlicher Plattformen genutzt. Darunter fallen Windows, Linux und Mac Build Support, aber auch Mobile Plattformen und Virtual Reality werden unterstützt. Einfache Spiele und Programme können in der Engine, ohne Code zu schreiben, mithilfe von Integrationen und Assets erstellt werden. Für mehr Kontrolle kann die Skriptsprache C# verwendet werden. In dieser Arbeit wird Unity genutzt, um kein Spiel, sondern eine Spiel-ähnliche 3D-Anwendung zur Analyse von Software zu entwickeln.

2.3 Methodisches Vorgehen

Normalerweise wird Software in Teams entwickelt, in welchen gute Koordination herrschen muss, sodass jeder mit seinen Teilaufgaben zum Gesamtprodukt beitragen kann. Damit bei einer solchen Arbeit strukturiert und übersichtlich vorgegangen werden kann, gibt es einige standardisierte Entwicklungsprozesse, welche dabei helfen sollen. Grob unterscheidet man zwischen linearer und agiler Softwareentwicklung. Grob ausgedrückt wird bei linearen Verfahren, zum Beispiel nach dem Wasserfallmodell[6], auf eine ausführliche Planungsphase[7] gesetzt. Da im darauf folgenden die Phasen Implementierung, Überprüfung und Wartung nur nacheinander abgearbeitet werden, hat man zwar weniger Aufwand in der Koordination, jedoch kann ein Fehler aus einer früheren Phase später schwer ausgebessert werden. Auch auf Änderungswünsche den Kunden kann nach Abschluss der Planungsphase schwer eingegangen werden.
Die agile Entwicklung leidet nicht unter solchen schwächen. Ein Verfahren, welches oft mit dieser in Verbindung gebracht wird, ist Scrum[8]. Hier werden üblicherweise, während sogenannten Sprints was maximal 30 tägige Phasen sind, kleinere Meilensteine erreicht. Diese können direkt getestet werden. Das passiert iterativ, sodass in den Tests gefundene Fehler frühzeitig in einem der nächsten Sprints behoben werden können. Außerdem kann dem Kunden regelmäßig ein funktionierendes Zwischenprodukt präsentiert werden. Damit wird weniger Gefahr gelaufen an den Wünschen des Kunden „vorbei zu Entwickeln“ und dessen Erwartungen nicht zu erfüllen. Dieser Prozess ist komplexer als lineare Verfahren und

[6] siehe https://www.dev-insider.de/was-ist-das-wasserfallmodell-a-680501/
[7] siehe Kapitel 3
[8] siehe https://www.scrum.org/resources/what-is-scrum

benötigt mehr Aufwand und Koordination um ihn erfolgreich umzusetzen.
Vergleicht man die beiden Verfahren miteinander merkt man, dass lineare Verfahren am besten in kleinen Teams und kleinen Projekten einzusetzen sind. Bei diesen sind die Ziele der Software oft von vorneherein klar und der größere Aufwand zur Umsetzung eines agilen Prozesses lohnt sich selten. Sobald die Teams oder das Projekt jedoch größere Ausmaße annehmen und mit Kunden gearbeitet wird, die noch keine klare Vision vom Endprodukt vor Augen haben, kann der zusätzliche Aufwand aber klein im Gegensatz zum geringeren Anpassungsaufwand und höherer Flexibilität ausfallen.

2.3.1 Angewandte Arbeitsweise

In diesem Projekt war die Wahl eines Entwicklungsprozesses komplizierter. Es sollte möglichst linear gearbeitet werden, um ein gutes Verhältnis zwischen Planung und Implementierung zu schließen und Entwicklungshasen nacheinander abschließen zu können. Es musste jedoch flexibel mit Änderungswünschen umgegangen werden können, da wie bei kommerzieller Software eine Art Kunde in Form der Prüfer bestand. Auch musste aufgrund von geringer Projekterfahrung davon ausgegangen werden, dass nicht das gesamte Produkt in einer Iteration geplant werden konnte.

Abbildung 2.1: Beispielhafter Aufbau eines Kanban Boards

Die Entscheidung fiel darauf, das Kanban-Framework[9] (Abbildung 2.1) zu benutzen. Bei dessen Verwendung werden möglichst kleine Teilaufgaben erstellt, welche dann auf Karten (zum Beispiel auf einem Whiteboard) in mehrere Spalten angeordnet werden. Jede Spalte stellt den aktuellen Status der in dieser befindlichen Aufgabe dar. So genügt ein schneller Blick, um eine Übersicht über Aufgaben in Planung, in Entwicklung und abgeschlossene Aufgaben zu erhalten. Karten können farbliche Markierungen zugeordnet werden, um deren Signifikanz erkennbar zu machen. Auch können die Aufgaben anhand der Markierungen in Kategorien (Organisation, Bug, Feature, ...) eingeteilt werden. So kann jederzeit auf Änderungswünsche reagiert werden, indem die Signifikanz der Karten angepasst wird. Auch bleibt bei der Entwicklung anhand von abgeschlossenen Karten ein guter Überblick über den Fortschritt.
Um den Stand des Boards immer und überall einsehbar zu haben wurde mit der Onlinelösung

[9] siehe https://www.atlassian.com/agile/kanban

Trello[10] gearbeitet. Diese bietet alle Grundfunktionen eines Kanban Boards und lässt sich mit Plugins Erweitern.

2.3.2 Versionsverwaltung

Zwar wurde alleine an den Graphen des SEE Projektes gearbeitet, jedoch wurde parallel dazu von anderen Leuten an anderen Funktionen der Software gearbeitet. Viele der Features sind dabei auf denselben Grundlagen, seien es Klassen von denen geerbt wird oder verwendete Prefabs, aufgebaut. Deshalb kann es dazu kommen, dass ein Entwickler eine Funktion komplett umbaut, während ein anderer eine neue Funktion implementiert, welche auf dieser basiert. Werden nun beide Änderungen übernommen, so funktioniert die neue Funktion des zweiten Entwicklers höchstwahrscheinlich nicht mehr. Um das zu vermeiden wurde ein Versionsverwaltungssystem namens Git verwendet. Chacon und Straub (2014) beschreiben diese wie folgt:

> „Versionsverwaltung ist ein System, welches die Änderungen an einer oder einer Reihe von Dateien über die Zeit hinweg protokolliert, sodass man später auf eine bestimmte Version zurückgreifen kann."

Git speichert jede Änderung an einem Projekt als „Commit", welcher eine Beschreibung und die Änderungen pro Datei enthält. Damit hat jeder einen Überblick an was gearbeitet wird und Änderungen können verglichen und zusammengeführt, oder fachlich „gemerged" werden, falls es zu Konflikten kommt. Ist dies im Notfall nicht mehr möglich, so kann ein älterer Commit gewählt werden, um alle darauf folgenden Änderungen zu verwerfen.
Eine weitere Funktion ist das Branchen. Damit kann eine Kopie der aktuellen Version des Projektes erstellt werden, welche individuell bearbeitet werden kann. Ist dann die Arbeit an dem Feature, für welches der Branch erstellt wurde, abgeschlossen, so können die Änderungen in einem Commit auf den Hauptbranch übertragen werden. Für diese Arbeit wurde also auch ein Branch des SEE Projektes erstellt. Während der Implementierungsphase wurde regelmäßig der aktuelle Stand des Hauptbranches auf den Branch dieser Arbeit kopiert.

[10] siehe https://trello.com/

KAPITEL 3

Planung

In dieser Arbeit war das Ziel, das bestehende SEE Projekt um einige Funktionen zu erweitern. Eine gemeinsame Anforderungsanalyse stellte die Grundlage für die zu implementierenden Funktionen, jedoch blieb bei den Details ein gewisser Entscheidungsspielraum. Um nicht während der Implementierungsphase auf Design-spezifische Fehler zu stoßen, welche im späteren Projektverlauf schwer anzupassen wären, wurde in einer kurzen Planungsphase eine Vorstellung für die finale Software entworfen. Außerdem wurde auf eine agile Weise[1] gearbeitet, die es ermöglichte während der Implementierung die Modellierung anzupassen.

3.1 Anforderungen

In vielen Firmen und Unternehmen wird Software für einen Kunden entwickelt. Auch zu dem Endprodukt dieser Arbeit wurde durch die Wünsche eines „Kunden“ in Form des ersten Gutachters, Prof. Dr. rer. nat. Rainer Koschke, geleitet. Die Wünsche des Kunden wurden als Anforderungen aufgenommen und zusammengefasst um sie hier aufzulisten.
Außerdem wurde der Fortschritt während der Implementierung dem „Kunden“ und Testern vorgestellt, wodurch sich einige Anforderungen änderten oder neue hinzukamen.

3.1.1 Von Anfang an bekannte Anforderungen

Das finale Produkt dieser Arbeit sollte zwei hauptsächliche Ziele erfüllen, welche sich einfach zusammenfassen lassen.

Darstellung aller Metriken

Eine Stadt des SEE Projekts bot bereits viele Möglichkeiten zur Inspektion einer Software. Das Problem dabei war jedoch, dass nur eine limitierte Anzahl an Metriken in einer Stadt dargestellt werden konnte. Um weitere Metriken zu inspizieren, mussten die Metriken neu zugewiesen werden, um dann die Stadt neu zu generieren. Während dieser Arbeit musste also

[1] siehe Abschnitt 2.3.1

eine Lösung entwickelt werden, mit welcher alle Metriken zur Visualisierung einer Software angezeigt werden konnten.

Bessere Übersicht

Das Zweite Ziel sollte eine bessere Übersicht in den Städten sein. Zwar bietet SEE einen schnellen, intuitiven Überblick, jedoch war dieser in großen Städten ungenau. Der Nutzer musste beispielsweise, um das größte Gebäude einer Stadt zu finden, schätzen. Den Nutzern sollte also auf Wunsch mehr Genauigkeit geboten werden.
Auf Vorschlag des Kunden, aber auch weil nach einiger Überlegung keine passendere Umsetzung gefunden wurde, wurde eine Darstellung über Graphen gewählt. Also eine Möglichkeit eine bis drei Metriken in einem Koordinatensystem abzubilden um sie miteinander zu vergleichen. Für die Implementierung dieser gab es folgende Ideen.

Darstellung der Daten

Vorerst sollten die Daten in einem zweidimensionalen Koordinatensystem dargestellt werden. Auf jeder der beiden Achsen sollte dabei eine Metrik abgebildet werden können. Eine dritte Achse könnte im Nachhinein durch Farbverläufe der Einträge ergänzt werden. Zum Erstellen der Graphen sollte der Nutzer vorerst alle Knoten einer Stadt auswählen, welche er in diesem angezeigt haben wollte. In diesem Schritt wird außerdem eine Metrik für jede Achse ausgewählt. Jeder Knoten welcher diese beiden Metriken enthielt, sollte dann anhand der Werte dieser Metriken in dem Koordinatensystem eingetragen werden. Nun sollte der Nutzer die Möglichkeit haben mit diesen Eintragungen interagieren zu können, um genaue Werte oder den zugehörigen Knoten anzuzeigen. Dies sollte über Kamerafahrten und Popups beim Hovern geschehen. Auch sollte es möglich sein mehrere Einträge auf einmal auszuwählen und die Metriken, welche auf den Achsen dargestellt werden, nachträglich zu ändern.

User Interface (UI)

Die Graphen sollten in der Desktop-Variante wie ein klassisches UI in Spielen und anderer 3D-Software, also über das Geschehen gelegt, präsentiert werden. Damit sollten sie nicht von der Bewegung des Nutzers in der dreidimensionalen Welt beeinflusst werden. Dem Nutzer sollten die Möglichkeiten gegeben sein, einen Graphen zu verschieben, seine Größe anzupassen, ihn zu minimieren oder zu schließen. Für die Ansicht in VR wäre diese Darstellung jedoch von Nachteil gewesen, da durch die Mitbewegung der Graphen bei der Rotation des Kopfes schnell ein Schwindelgefühl oder anderes Unwohl hervorgerufen wird. Deshalb sollten in VR die Graphen als eigene Objekte in der Stadt dargestellt werden. Damit ist gemeint, dass diese ihre Position bei einer Rotation des Kopfes oder kleineren Translationen, also Bewegungen im Raum, beibehalten. Für schnellere Erreichbarkeit sollten sie aber den größeren Translationen

des Nutzers verzögert folgen. Dabei sollten die Graphen selbst immer so rotiert sein, dass sie mit der Vorderseite zum Nutzer zeigten.

Erweiterungsideen

Um nach einer schnellen Implementierung die Software noch zu verbessern gab es einige Möglichkeiten.

Anpassung der Start- und Endwerte Für eine bessere Übersicht könnten Start- und Endwerte auf den Achsen anpassbar gemacht werden, sodass nur ein bestimmter Bereich der Metriken dargestellt wird.

Zusammenfassung von Einträgen Falls mehrere Einträge in einem Graphen nahe beieinander liegen sollten, könnten diese zusammengefasst oder farblich markiert werden. Dadurch wären Hotspots schnell ersichtlich.

Zoom Funktion und Änderung der Einträge Eine Zoom Funktion in einen Graphen, so wie die nachträgliche Änderung der im Graphen dargestellten Knoten wären auch Optionen gewesen.

Sollten diese Anforderungen erfüllt werden, so würde ein Nutzer des SEE Projektes diese ohne Störungen bei Nutzung der bisherigen Funktionen nutzen können und bei Bedarf zusätzlich auf eine genauere Darstellung der Daten zugreifen können, ohne dafür eine externe Software aufzurufen.

3.1.2 Neue und geänderte Anforderungen während der Implementierung

Während einiger Gespräche mit Betreuern oder hilfsbereiten Kommilitonen haben sich in der Implementierungsphase noch neue Anforderungen ergeben, oder alte geändert.

Selbe Metrik auf beiden Achsen

Der anspruchsvollste aber wahrscheinlich sinnvollste Änderungswunsch, war eine andere Darstellung von Werten in Graphen, welche dieselbe Metrik auf beiden Achsen abbildeten. Vor der Änderung wurden auf beiden Achsen dieselben Werte abgebildet. Enthielt eine Metrik also Werte von 0 bis 15, so war der kleinste Wert auf der x- und der y-Achse 0 und der größte 15. Dadurch wurden Einträge immer auf der Diagonalen, welche in einem 45° Winkel durch den Nullpunkt verlief, dargestellt. So war es schwer die Einträge untereinander zu Vergleichen.

Die Lösung war es, den Wert der doppelten Metrik nur auf der y-Achse darzustellen. Damit blieb die x-Achse, um auf dieser eine Nummerierung der Einträge von 1 bis zu der Anzahl an dargestellten Knoten einzuführen. Auf dieser Achse würden die Einträge einen regelmäßigen Abstand voneinander haben. Die Einträge wurden nach ihrer Größe bezogen auf die gewählte Metrik sortiert, sodass der kleinste ganz links im Graphen und der größte ganz rechts im Graphen angezeigt werden würde. So wurden Einträge mit demselben Wert auf einer Horizontalen Linie nebeneinander dargestellt, ohne sich wie vorher zu überlappen.

Auswahlmodus

Ach neu war ein Auswahlmodus, wie man ihn beispielsweise aus dem Windows File Explorer kennt. Wird eine bestimmte Taste gedrückt gehalten, so bleiben alle zuvor markierten Einträge bei der Auswahl eines neuen markiert. In den Graphen sollte es möglich sein, die automatische Abschaltung der Hervorhebungen, also der aktuellen Auswahl, zu umgehen, um beliebig viele Einträge zu markieren.

Baum Darstellung der Seitenleiste

Durch die Verlängerung der Arbeit blieb außerdem Zeit für ein weiteres Feature. Anfangs wurden in einer Seitenleiste[2] neben einem Graphen alle Einträge in diesem als Liste dargestellt. In dieser sollte der Nutzer einzelne Einträge deaktivieren können, damit diese nicht mehr im Graphen dargestellt werden. Hier wurde in der Aufteilung in Unterkategorien nur zwischen „Buildings“ (Gebäuden) und „Nodes“ (Knoten) unterschieden. Auf Wunsch des Nutzers sollte es aber auch möglich sein diese in ihrer ursprünglichen Anordnung des Dateibaumes darzustellen, aus welchem sie eingelesen wurden.

Alles in allem ließen sich die neuen Anforderungen gut in den Arbeitsfluss einbringen und haben die Software noch bereichert.

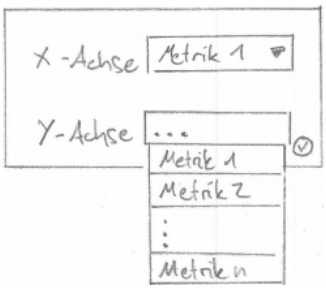
3.2 Mockups

Um bereits vor der Implementierungsphase einen Plan über die Umsetzung der Funktionen der Software zu haben, wurden diese im Voraus modelliert. Das kostet zwar zusätzliche Zeit, kann sich jedoch auszahlen, da hier Änderungen deutlich leichter umzusetzen sind als wären diese bereits Implementiert. Als Mittel wurden Mockups gewählt. Mockups sind eine „quick and dirty“ (Rivero u. a., 2010), also schnelle und einfache, Art Prototypen für vor allem das User Interface (UI) der Software zu entwerfen.

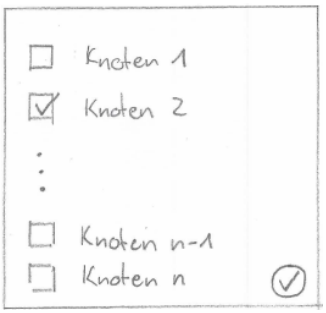
[2] siehe Abschnitt 4.1.4

„A mockup is a sketch of a possible user interface (UI) of the application that helps to agree on broad aspects of the UI and can be easily created by any stakeholder.“ (Rivero u. a., 2010)

Für diese Arbeit entworfene Mockups



(a) 1. : Auswahl der Metriken



(b) 2. : Auswahl der Knoten

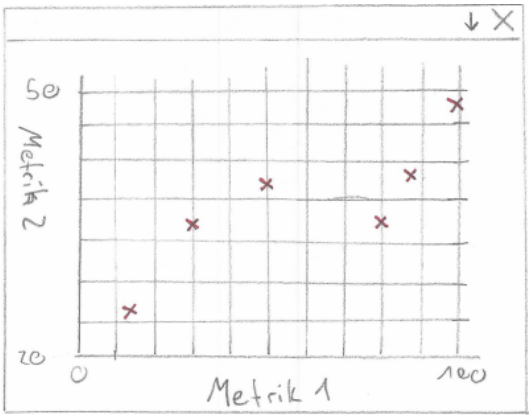


Abbildung 3.1: 3. : Der erstellte Graph

Mit diesen Mockups wird der geplante Ablauf zur Erstellung eines Graphen beschrieben. Auf Abbildung 3.1(a) ist die Auswahl der Metriken für jede Achse des neuen Graphen abgebildet. Hat der Nutzer zwei Metriken gewählt, so wird das Fenster aus Abbildung 3.1(b) geöffnet. In diesem kann der Nutzer alle Knoten wählen, welche im Graphen angezeigt werden sollen. Resultierend aus der Auswahl des Nutzers wird der Graph aus Abbildung 3.1 generiert. Dieser verhält sich wie ein Windows 10 Fenster, kann also über die Icons oben rechts geschlossen und minimiert werden und über den Rand vergrößert und verkleinert. Über die obere Leiste kann der Graph verschoben werden. Für jede der gewählten Metriken werden der maximale und minimale Wert gefunden um den Bereich der Darstellung festzulegen. Die gewählten Knoten werden als Markierungen in den Graphen eingetragen.

KAPITEL 4

Implementierung

Mit einer ungefähren Vorstellung vom Endprodukt konnte mit der Implementierung begonnen werden. Die Ergebnisse dieser werden in diesem Kapitel erläutert.

4.1 Funktionen des fertigen Produktes

Wie zu erwarten, fielen viele der Ergebnisse dieser Arbeit am Ende anders aus als geplant. Was von den Anforderungen implementiert wurde, was abgewandelt wurde und welche Ideen verworfen wurden, wird in diesem Abschnitt erläutert. Das Ergebnis wird mit den in Abschnitt 3.1 beschriebenen Anforderungen verglichen. Ist dies nicht der Fall, so ist davon auszugehen, dass die Anforderung wie beschrieben implementiert wurde.

4.1.1 Erstellung von Graphen

Anfangs war der Plan, den Nutzer erst bestimmen zu lassen, welche Gebäude im Graphen anzuzeigen sind und welche Metriken auf den Achsen dargestellt werden sollen, um dann den Graphen dazu zu bilden. Dieser Ansatz war unausgereift und benötigte viele Schritte vom Nutzer um einen Graphen zu erstellen. In der finalen Version wird, sobald der Nutzer den erstellen Knopf betätigt, ein neuer Graph mit allen Knoten und der ersten ausgelesenen Metriken auf beiden Achsen erstellt. Möchte der Nutzer andere Metriken anzeigen lassen, so können diese aus einer ausklappbaren Liste ausgewählt werden. Der Graph wird automatisch aktualisiert. Sollen die angezeigten Knoten geändert werden wollen, gibt es dafür eine Liste in der Seitenleiste[1] rechts am Graphen, wo jeder Knoten einzeln aktiviert oder deaktiviert werden kann.

[1] siehe Abschnitt 4.1.4

4.1.2 Darstellung von Daten

Die Idee die Darstellung von zwei auf drei Achsen zu erweitern, wurde leider nicht mehr mit aufgegriffen[2]. Eine Lösung mehr als zwei, theoretisch endlos viele, Metriken miteinander zu vergleichen wurde dennoch gefunden. Der Vergleich wurde ermöglicht, indem die Einträge über mehrere Graphen hinweg anhand der Hervorhebungen verbunden wurden. Wird ein Eintrag angeklickt, so wird nicht nur der verbundene Knoten in der Stadt hervorgehoben, sondern es wird ein Befehl an alle aktiven Graphen gesendet. Falls ein Graph einen Eintrag enthält, welcher mit dem selben Knoten verbunden ist, so wird dieser hervorgehoben[3]. Das ermöglicht beispielsweise, dass der Nutzer zwei Metriken im ersten und zwei weitere im zweiten Graphen anzeigen lässt, dann einen oder mehrere Einträge im ersten Graphen anklickt um diese in beiden Graphen hervorzuheben. Somit können die Werte dieser Einträge im zweiten Graphen ausgelesen werden. Dabei hilft eine weitere neue Funktion, welche es ermöglicht, Hervorhebungen so lange aktiviert zu lassen, wie eine bestimmte Taste gehalten wird[4].

4.1.3 Interaktion mit Einträgen

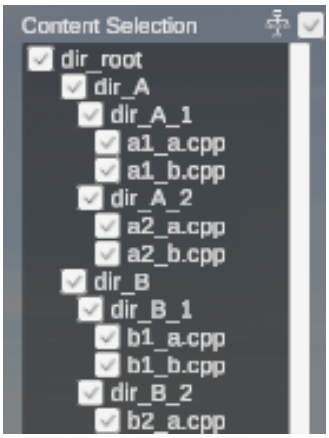
Die Anforderungen, wie mit den Einträgen in einem Graphen interagiert werden kann, wurden fast genau so wie geplant implementiert. Leichter umzusetzen waren die Aktionen, welche durch einen einfachen Klick und einen Doppelklick auf einen Eintrag ausgelöst werden können. Durch ersteren werden Hervorhebungen aktiviert, und durch letzteren wird in der Desktop-Variante eine Kamerafahrt zum verbundenen Knoten ausgelöst. Entweder mit oder ohne Drehung der Kamera.
Komplexer war die Implementierung eines Rechtecks, was zur Auswahl von mehreren Einträgen um diese herum gezogen werden kann. Dies funktioniert ähnlich wie die Auswahl mehrerer Dateien auf dem Windows 10 Desktop. Bei der Auswahl müssen verschiedene Fälle geprüft werden, um herauszufinden ob ein Eintrag in einem Graphen zwischen zwei diagonal gegenüberliegenden Ecken des Rechtecks liegt oder nicht. Der Aufwand welcher in diese Funktion geflossen ist war es aber wert. Gerade in VR ist es hiermit deutlich leichter Einträge, auch einzelne, auszuwählen. Die Interaktion mit überlappenden Einträgen gestaltet sich leider als nicht möglich. Der einzige Umweg welcher diese ermöglicht ist, überlappende Einträge anhand einer Einfärbung, welche je nach Anzahl der Überlappungen roter ausfällt zu erkennen. Diese können dann in der Seitenleiste[1] deaktiviert werden, um auf darunter liegende Einträge zuzugreifen.

[2] siehe Abschnitt 4.1.7
[3] siehe Abschnitt 4.3.2
[4] siehe Abschnitt 3.1.2

4.1.4 Seitenleiste

In Abschnitt 4.1.1 wurde beschrieben, wie die Auswahl der in einem Graphen angezeigten Knoten umgebaut wurde, um nach der Erstellung des Graphen zu geschehen, anstatt davor. Für diese Umsetzung wurde eine Möglichkeit benötigt die Knoten, welche nicht angezeigt werden wieder einzublenden. Natürlich hätte dies über die Auswahl von Gebäuden direkt in der Stadt passieren können, die hätte jedoch den Nachteil einer schlechten Übersicht. Deshalb wurde eine Darstellung als pro Graphen ausklappbare Liste gewählt, in welcher alle Knoten der Stadt, ob im Graphen aktiviert oder nicht, anhand deren Namen angezeigt werden. Diese werden standardmäßig wie in Abbildung 4.1(a) in Gebäude und Straßen unterteilt. Zu jedem Eintrag kann ein Haken gesetzt werden, um diesen aktiv oder inaktiv für den aktuellen Graphen zu schalten.
Um schnell Einträge eines Graphen in der Seitenleiste zu finden, werden auch in dieser die Hervorhebungen von Knoten synchronisiert. Wird also ein Eintrag ausgewählt, so wird das zugehörige Gebäude, alle Einträge zu dem selben Knoten aus anderen Graphen und der Eintrag in der Seitenleiste hervorgehoben.

(a) Darstellung als Liste



(b) Darstellung als Baum

Abbildung 4.1: Auswahl der anzuzeigenden Knoten über die Seitenleiste

Die Darstellung der Stadt basiert auf Daten, welche vorher aus einer hierarchischen Datenstruktur ausgelesen wurden. Um diese Struktur nicht nur anhand der Stadt auslesen zu können, ist es möglich diese wie in Abbildung 4.1(b) in der Seitenleiste anzuzeigen. Dabei wird die Anordnung nach Gebäuden und Straßen deaktiviert, um die Knoten der Stadt mit den Wurzelknoten beginnend nach der gegebenen Datenstruktur anzuordnen.

4.1.5 Überlappungen

Die im Folgenden beschriebene Darstellung von überlappenden Einträgen ersetzt zwei Funktionen aus den Erweiterungsideen[5]. Die Zusammenfassung von Einträgen und die Zoom Funktion.
In der finalen Version der Software werden Verfärbungen der Einträge genutzt, um die Anzahl der auf einer Stelle überlappenden Einträge abschätzbar zu machen. Dadurch, dass die Eintragungen nacheinander geschehen wird jedes mal ein definierter Bereich um den genauen Punkt des Eintrags geprüft. Für jeden in diesen Bereich ragenden anderen Eintrag wird der neue dann in einem dunkleren Rot eingefärbt als der vorherige und über alle anderen gelegt. Das Zusammenfassen der Einträge geschieht somit durch die Überlagerung der unterliegenden Einträge.
Die Zoom Funktion sollte zum einem dazu dienen Ansammlungen zu entdecken, was hiermit gelöst wurde, aber zum anderen sollte sie diese Ansammlungen genauer untersuchbar machen. Da die unterliegenden Einträge momentan nur durch das deaktivieren der oberen auswählbar sind, wäre das Zoomen weiterhin eine Überlegung für die zukünftige Entwicklung.

4.1.6 Virtual Reality (VR)

Alle Funktionen, welche in der Desktop-Variante der Software benutzbar sind, funktionieren auch über SteamVR, also über ein HMD in VR. Dafür wurden die Vorstellungen aus 3.1.1 umgesetzt. Jeder Graph ist ein eigenes im Raum bewegbares Objekt, welches den Translationen des Nutzers folgt. Zur Bedienung wirft der rechte Controller einen Strahl, welcher bei Interaktionen ähnlich wie der Mauszeiger funktioniert. Zeigt der Nutzer auf das verschieben Symbol und hält die selbst belegte Interaktionstaste gedrückt, kann er den Graphen um sich herum bewegen und über das Touchpad weiter weg oder näher heran schieben. Damit Graphen nicht bei der automatischen Bewegung verloren gehen, gibt es eine Taste um alle Graphen hintereinander angeordnet vor dem Spieler zu platzieren.

4.1.7 Nicht implementierte Anforderungen

Wie erwartet kam es vor, dass einige Anforderungen aufgrund von Zeitmangel oder besseren Lösungen nicht implementiert wurden. Diese werden im Folgenden beschrieben.

Boolesche Werte

Ausgelassen wurde die Darstellung von booleschen Werten. Man hätte diese zwar anhand einer Einfärbung der Einträge in zwei Farben, wahr oder falsch, darstellen können jedoch sollten die Graphen dazu dienen Werte in Relation zueinander Darzustellen. Ein besserer

[5] siehe Abschnitt 3.1.1

Nutzen der farblichen Markierung wurde in Abschnitt 4.1.5 beschrieben. In der Zukunft könnten boolesche Werte über Graphen, welche alle Metriken von nur einem Knoten anzeigen, implementiert werden.

Dritte Achse

Ähnlich wie die booleschen Werte wurde die in Abschnitt 3.1.1 erwähnte dritte Achse ausgelassen. Auch diese hätte über die Farbe der Einträge dargestellt werden können. Eine dreidimensionale Darstellung der Graphen hätte zwar in VR gut funktionieren können, wäre aber in der Desktopversion schwer zu Nutzen. In Abschnitt 4.1.2 wurde jedoch eine Lösung für das Problem gefunden.

4.2 Design

Wenn es darum geht Informationen effizient und übersichtlich darzustellen, dann ist einerseits viel Optimierung im Backend (was der Nutzer nicht sieht) der Software nötig, damit diese nicht zu lange für Anfragen benötigt und diese korrekt auswertet. Andererseits ist das Frontend (was der Nutzer sieht) aber genauso- wenn nicht sogar wichtiger, weil hier die Effizienz des Nutzers beeinflusst wird. Braucht der Nutzer mehrere Interaktionen mit der Software um eine Anfrage zu starten, wenn es auch mit einer ginge, so bietet ein noch so hochperformantes Backend keinen großen Vorteil. Der Nutzer sollte immer wissen wie er mit der Software umzugehen hat, um seine Ziele schnell zu erreichen. Im Falle von Fehlern oder falsche Benutzung soll bei der Entwicklung auch dafür ein entsprechendes verhalten programmiert werden.

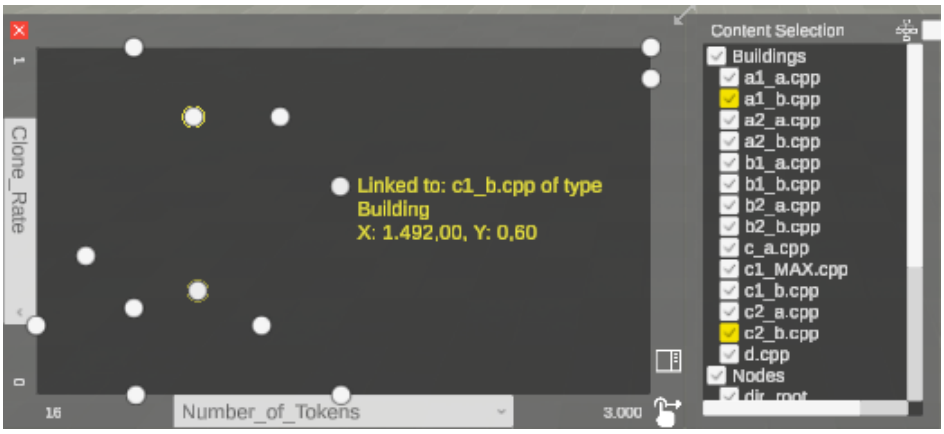


Abbildung 4.2: Finales Design der Graphen

Abbildung 4.2 zeigt einen Graphen wie er in der finalen Version der Software dargestellt wird. Bis auf die Benutzeroberfläche zum Erstellen neuer Graphen ist dies die einzige Nutzerschnittstelle. Diese wurde versucht möglichst simpel und übersichtlich zu halten. Nicht in dieser Abbildung sichtbar ist, dass der Nutzer den Graphen auch verschieben kann, indem er

die hellgraue Oberfläche gedrückt hält. Dies haben zuvor viele Tester vergebens versucht. Der gelbe Text wird angezeigt, wenn der Cursor über einen Eintrag gehalten wird. Er zeigt die genauen Werte zu diesem an, da kein wie im Mockup[6] gezeigtes Gitter implementiert wurde. Dafür lassen sich die beiden Leisten, welche anzeigen was auf den Achsen dargestellt wird, ausklappen um die Metrik zu wechseln. Der Inhalt aktualisiert sich dann automatisch. Über das verschieben Symbol unten rechts kann der Graph außerdem minimiert werden.

4.3 (Code/Software) Architektur

In diesem Abschnitt wird der Aufbau der Software weiter erläutert, indem auf wichtige Skripte, Klassen, Abläufe und Abhängigkeiten eingegangen wird.

4.3.1 Essenzielle Skripte und Klassen

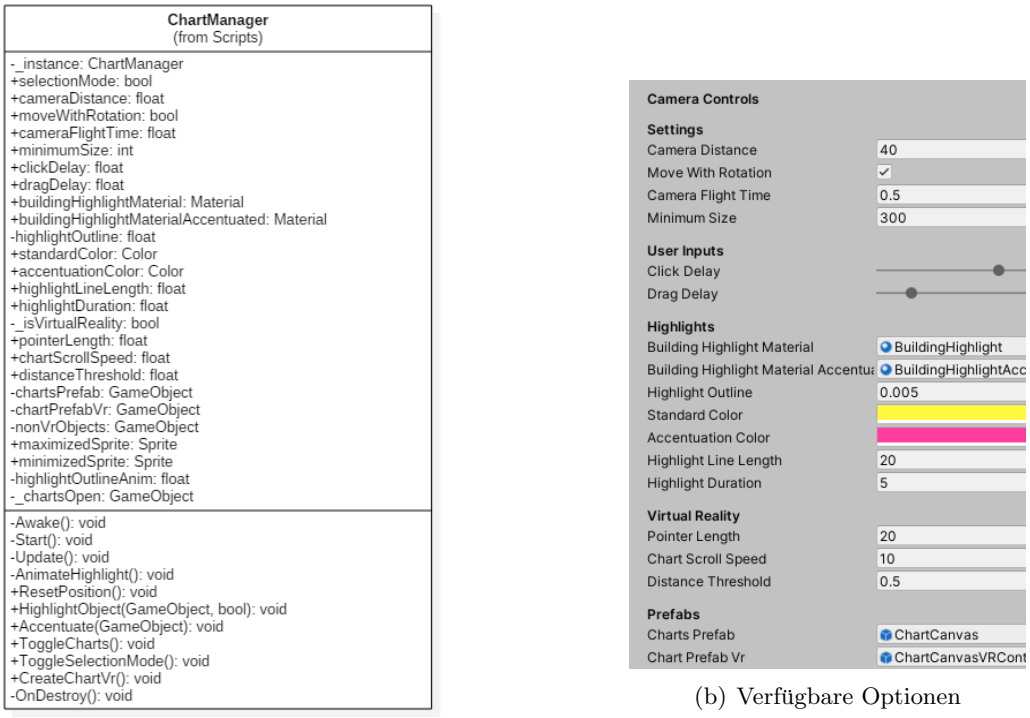
Die Darstellung der Graphen in SEE kann in kleinere Funktionalitäten herunter gebrochen werden. Das ist zum Beispiel das Erstellen von Graphen, das Anpassen der Werte über die Seitenleiste in diesen oder die Interaktion mit Einträgen. Um den Code strukturierter und modularer zu halten werden einzelne Funktionalitäten in eigene Skripte ausgelagert. Einige dieser Skripte finden häufigere Verwendung bei der Benutzung der Software und werden im Folgenden erläutert.

ChartManager

In der Entwicklung in Unity ist es üblich, einen GameManager zu schreiben. Dies ist eine Klasse, in welcher die Initialisierung der Szene, Transitionen ins nächste Level oder die verstrichene Zeit seit Spielbeginn gehandhabt werden. Der `ChartManager` übernimmt eine ähnliche Rolle für die Graphen in SEE.

In diesem werden alle wichtigen Einstellungen, welche die Graphen betreffen als öffentliche Attribute eingetragen. So können alle anderen Skripte, welche auf diese zugreifen müssen dies tun, während Entwickler alle Variablen an einem leicht auffindbaren Platz anpassen können. Teilen sich zwei Skripte ein Attribut, so muss dieses nicht doppelt angelegt werden.
Zudem wird im `ChartManager`, wie auch beim Ansatz des GameManagers, grundlegende Logik implementiert, welche die Graphen betrifft. Das Pulsieren der hervorgehobenen Gebäude und Einträge wird hier animiert. Der `ChartManager` ist während eine Szene in welcher Graphen unterstützt werden gespielt wird immer aktiv. Skripte sind meist nur so lange aktiv, wie das entsprechende Objekt zu welchem sie gehören in der Szene ist. Ein solches Verhalten wäre für die Animation eines durchgehenden Pulsierens ungeeignet.

[6] siehe Abschnitt 3.2



(a) UML Ansicht

(b) Verfügbare Optionen

Abbildung 4.3: Aufbau des `ChartManagers`

Auch werden aufrufe welche alle aktiven Graphen betreffen, wie das Hervorheben aller Einträge zu einem bestimmten Knoten oder das Zurücksetzen der Position der Graphen in VR hier ausgeführt. Dazu werden die Graphen mit einem „Tag“ markiert, mit welchem sie in der Szene gefunden werden. An jeden der gefundenen Graphen wird dann das hervorzuhebende Objekt weitergegeben.

Listing 4.1: Methode zum Hervorheben aller zu einem Knoten gehörigen Einträge

```
public static void HighlightObject(GameObject highlight, bool
    scrollView)
 {
    var charts = GameObject.FindGameObjectsWithTag("Chart");
    foreach (var chart in charts)
    {
       chart.TryGetComponent<ChartContent>(out var content);
       content.HighlightCorrespondingMarker(highlight, scrollView)
           ;
    }
 }
```

ChartContent

Jeder erstellte Graph hat eigene Einträge, eine eigene Inhaltsauswahl in der Seitenleiste am Rand und muss vergrößert, verschoben und geschlossen werden können. Das `ChartContent` Skript wird pro erstelltem Graphen instantiiert, und realisiert die grundlegenden Funktionen eines Graphen. Die Erstellung eines neuen Graphen erfordert zum Beispiel das Finden aller Knoten, welche abgebildet werden sollen, um dann für jeden dieser Knoten einen Eintrag zum Graphen hinzuzufügen. Dies wird genauer in Abschnitt 4.3.2 erklärt. Der Inhalt des Graphen muss regelmäßig aktualisiert werden, um auf Größenänderung oder eine Änderung der Knoten in der Szene zu reagieren. Wird der Button zum Schließen eines Graphen betätigt, so muss der Graph und die zugehörigen Objekte zerstört (das GameObject wird aus der Szene entfernt) werden.

ChartMarker

Abbildung 4.4: UML Ansicht der Klasse `ChartMarker`

Die in einem Graphen angezeigten Daten werden über diese Klasse als kreisförmige Einträge dargestellt. Sie werden in der Software als „Marker“ bezeichnet. Das `ChartMarker` Skript verwaltet alle Interaktionen, welche der Nutzer mit „Markern“ tätigt. Dazu gehören unter anderem die Hervorhebung von Einträgen und dem mit ihnen verbundenen Knoten in der Szene, die Kameraführung zu dem verbundenen Knoten oder die Aktivierung des Infotextes, welcher genauere Informationen zu dem jeweiligen Eintrag anzeigt.

NodeHighlights

Dieses Skript existiert pro Knoten in der Szene und aktiviert oder deaktiviert die Hervorhebungen. Diese können durch einen Klick auf den Knoten ausgelöst werden und werden wie in Abschnitt 4.1.2 beschrieben über alle Graphen synchronisiert. Das geschieht über ein Dictionary, welches jeden Graphen und einen zugehörigen booleschen Wert speichert. Dieser Wert gibt an ob der Knoten in dem Graphen aktiv ist.

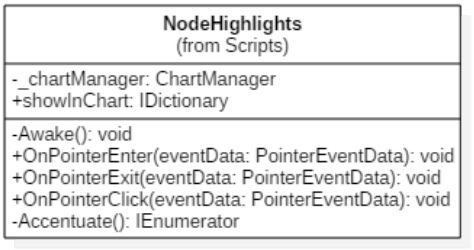


Abbildung 4.5: UML Ansicht der Klasse `NodeHighlights`

Listing 4.2: Das Dictionary der Knoten (aus `NodeHighlights`)

```
public IDictionary showInChart = new Dictionary<ChartContent, bool
    >();
```

Listing 4.3: Eintragung eines Graphen in die Dictionaries der Knoten (aus `ChartContent`)

```
_dataObjects = GameObject.FindGameObjectsWithTag("Node");
      foreach (var entry in _dataObjects)
      {
        entry.TryGetComponent<NodeHighlights>(out var highlights);
        if (!highlights.showInChart.Contains(this)) highlights.
            showInChart.Add(this, true);
      }
```

Virtual Reality (VR)

Für den VR Modus wurde von allen Skripten welche ein verhalten beinhalten, welches in VR anders funktioniert ein von diesem Skript erbendes Skript erstellt. In diesem werden die entsprechenden Methoden dann wie in Abbildung 4.6 überschrieben. Damit diese Skripte geladen werden, wurde für die meisten Prefabs noch einmal eine neue Variante für den VR Modus erstellt.

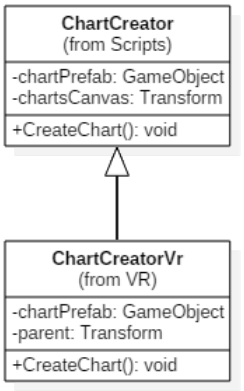


Abbildung 4.6: Vererbung der VR Skripte

4.3.2 Prozessabläufe

In diesem Abschnitt soll die finale Architektur der Software anhand von UML[7] Sequenzdiagrammen näher erläutert werden. Dazu kann nicht auf die gesamte Software eingegangen werden. Es werden die wichtigsten Prozesse erläutert.

Was sind Sequenzdiagramme?

Diese Art von UML[8] Diagrammen stellt die Interaktionen zwischen Objekten in der Reihenfolge dar, in welcher sie bei der Nutzung stattfinden. Das bedeutet, ein solches Diagramm ist von oben nach unten zu lesen. Die Interaktionen werden anhand von Nachrichten in Form von Pfeilen zwischen den Objekten dargestellt. Ausgefüllte Pfeilspitzen stehen für synchrone Nachrichten, auf welche eine Antwortnachricht erwartet wird, bevor fortgefahren werden kann. Die Antwort wird durch einen gestrichelten Pfeil dargestellt. Asynchrone Nachrichten haben keine ausgefüllte Pfeilspitze. Eine gestrichelte, vertikale Linie unter den Objekten gibt deren Lebensdauer an, wobei Kästen ihre Aktivität anzeigen. Es werden Rahmen um Nachrichtenaustausche genutzt, um Schleifen oder if-Abfragen darzustellen.
Anzumerken ist hier, dass für einer klarere Darstellung einige Symbole anders genutzt wurden, als durch die Spezifikation (Object Management Group, 2017) vorgesehen. Wurde Beispielsweise die Methode `GameObject.FindGameObjectWithTag(tag)` genutzt, um bereits existierende Objekte in der Szene zu finden, so wurde diese als erstellende Nachricht dargestellt. Diese werden sonst genutzt, um Objekte zu Instantiieren. Da sie jedoch dem aufrufenden Objekt erst nach dem Aufruf bekannt sind, wurde diese Anpassung als sinnvoll empfunden.

Hervorhebung von Einträgen

Das Sequenzdiagramm 4.7 zeigt den Prozess welcher im Hintergrund abläuft, wenn der Nutzer einen Eintrag in einem Graphen anklickt, um diesen hervorzuheben. Dieser findet hierbei in

[7] siehe Abschnitt 2.2.1
[8] genauere Informationen sind auf der offiziellen Seite der Object Management Group (2017) zu finden

der Desktop-Variante statt.

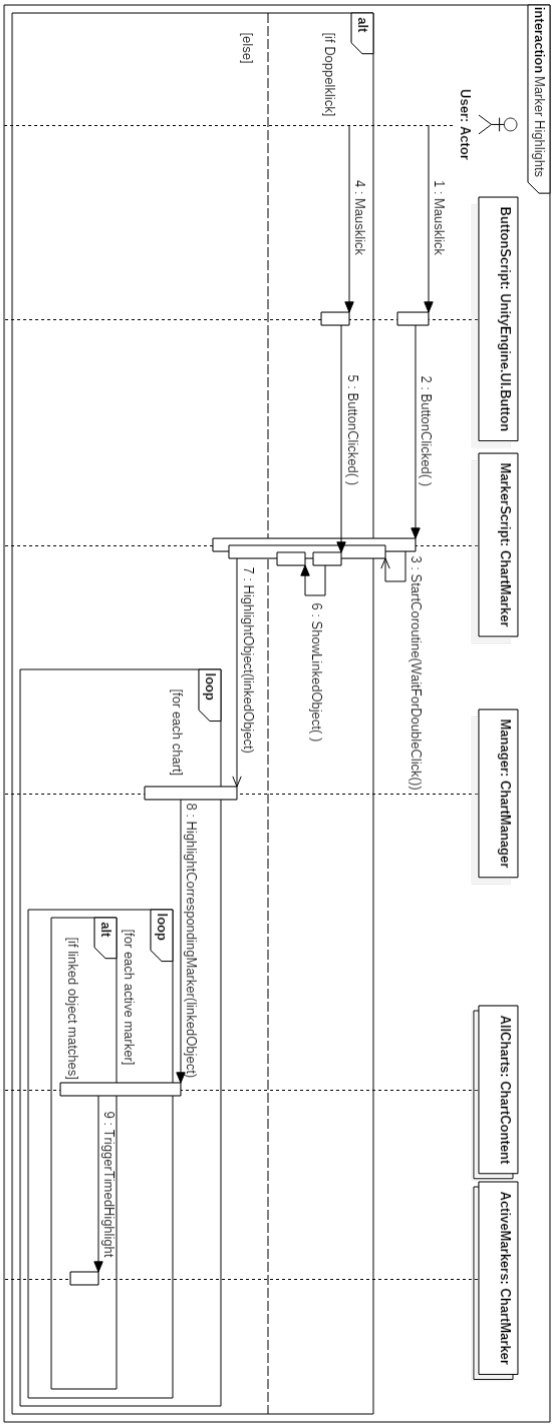


Abbildung 4.7: Sequenzdiagramm zur Hervorhebung von Einträgen.

Im Diagramm 4.7 wird der Ablauf des Prozesses Schritt für Schritt genauer erläutert. Der Prozess wird ausgelöst, indem der Nutzer auf einen Eintrag in einem Graphen klickt. Der Ablauf

wie ein Mausklick von Unity verarbeitet wird wurde hierbei ein wenig vereinfacht. Es läuft jedoch darauf hinaus, dass das Skript welches Buttons realisiert auf das auf dem selben GameObject liegende Skript `ChartMarker` zugreift, indem es die Methode `ButtonClicked()`[9] aufruft. Diese startet eine Coroutine. Das sind Funktionen, welche nicht in einem Frame von Unity abgearbeitet werden müssen, sondern über mehrere Frames hinweg parallel zu den Hauptprozessen laufen können um diese nicht zu unterbrechen. Diese Coroutine läuft für eine bestimmte Zeit. Wird in dieser Zeit noch einmal die Methode `ButtonClicked()`[10] aufgerufen, so meldet diese der laufenden Coroutine, dass ein Doppelklick durchgeführt wurde. Das führt dazu, dass die Kamera auf den mit dem Eintrag verbundenen Knoten Zoomt, worauf in diesem Diagramm nicht weiter eingegangen wird. In dem Fall, dass die Zeit ohne einen zweiten Klick ausläuft, wird das dem `ChartManager` durch den Aufruf von `HighlightObject(linkedObject)`[11] gemeldet. Dabei ist `linkedObject` der Knoten welcher mit dem Eintrag verbunden ist. Im `ChartManager` werden jetzt alle aktiven Graphen gesucht, um anhand des Aufrufes `HighlightCorrespondingMarker(linkedObject)`[12] den hervorzuhebenden Knoten weiterzugeben. In jedem Graphen werden dann alle aktiven Einträge aufgerufen, um deren verbundenen Knoten mit dem weitergegebenen Knoten zu vergleichen. Handelt es sich um das selbe Objekt, so wird `ToggleTimedHighlight()`[13] auf den Eintrag aufgerufen.

Hinzufügen eines Graphen

Im folgenden Sequenzdiagramm (Abbildung 4.8) wird der Ablauf zum Erstellen von neuen Graphen in der Desktop-Variante dargestellt.

[9] siehe Nachricht 1 in Abbildung 4.7
[10] siehe Nachricht 4 in Abbildung 4.7
[11] siehe Nachricht 7 in Abbildung 4.7
[12] siehe Nachricht 8 in Abbildung 4.7
[13] siehe Nachricht 9 in Abbildung 4.7

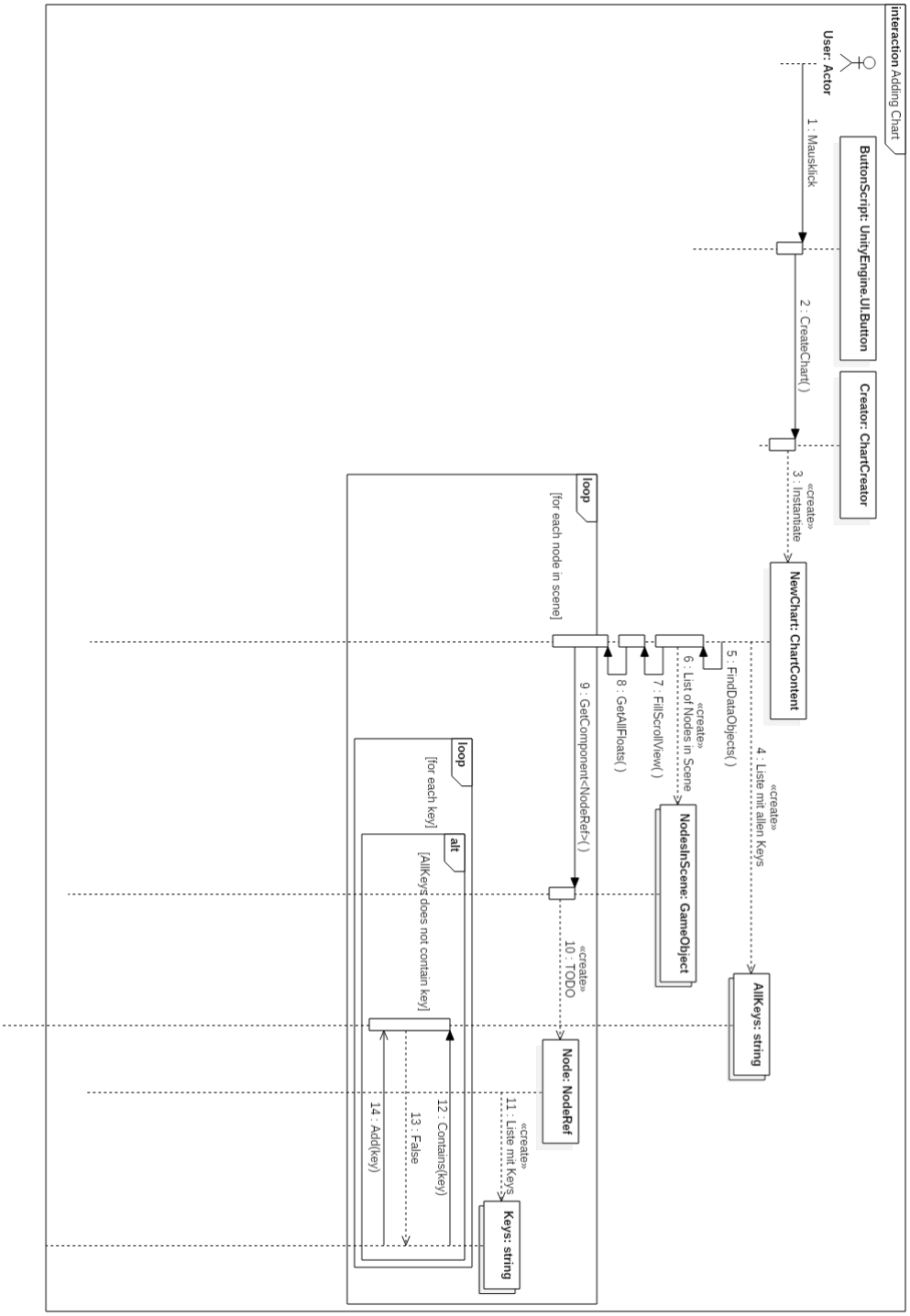


Abbildung 4.8: Sequenzdiagramm zum Hinzufügen von neuen Graphen

Wie bei der Hervorhebung von Einträgen aus 4.3.2, wird auch die Erstellung eines neuen Graphen durch den Mausklick eines Nutzers ausgelöst. Dieser führt durch das `Button` Skript

zu dem Aufruf `CreateChart()`[14] umgewandelt. Darauf folgt eine richtige Instantiierung des Graphen Prefabs und nicht zu einer wie in 4.3.2 beschriebenen Suche von bestehenden Objekten, welche auf die selbe Art dargestellt wird. Das Prefab enthält das Skript `ChartContent`. Dieses Skript Sucht nach der Erstellung alle Knoten aus der Szene und fügt diese in eine Liste ein. Diese Knoten werden dann auch mit `FillScrollView()`[15] in die Seitenleiste des Graphen eingetragen, von welcher aus sie in diesem aktiviert oder deaktiviert werden könne. Die Nachricht acht in Abbildung 4.7 steht nun für zwei separate Aufrufe der Methoden `GetAllFloats()` und `GetAllIntegers()`. Da sich diese jedoch bis auf den Datentyp der abgefragt wird gleichen, wurden sie zu einer Nachricht zusammengefasst. Um alle Keys welche von Knoten enthalten werden zu finden wird hier wie folgt vorgegangen. Zuerst wird mit `GetComponent<NodeRef>()` das an ein GameObject in der Szene angehängte `NodeRef` Skript abgefragt. Dieses enthält ein weiteres Skript, welches Schlüssel und zu den Schlüsseln gehörige Werte enthält. Diese sind dabei nach Datentypen gruppiert. Es werden also alle Schlüssel eines Datentyps abgerufen, um dann für jeden dieser Schlüssel zu überprüfen, ob dieser bereits in einer Liste mit allen Schlüsseln aufzufinden ist. Falls nicht so wird er dieser hinzugefügt. Das geschieht für alle GameObjects, welche zuvor in Nachricht 5 in Abbildung 4.8 als Liste gespeichert wurden.

4.4 Komplexität der Hervorhebungen

Eine Funktion, welche in den Graphen automatisch bei verschiedenen Interaktionen aktiviert wird, sind die Hervorhebungen von Einträgen. Diese Funktion trägt zu der Orientierung des Nutzers bei, sollte also nicht ausgelassen oder um Unterfunktionen reduziert werden. Diese, wie auch andere Funktionen, im Voraus zu Planen hätte durch wenig Vorerfahrungen viel Zeit gekostet. Anfangs fiel der Code der die Hervorhebungen realisierte nicht negativ auf, da die Funktionalität der Graphen noch nicht ausgeprägt war, doch gegen Ende wurde musste dieser zu viele Funktionen gleichzeitig erfüllen und wurde unübersichtlich und fehlerhaft. Die Komplikationen begannen mit der Synchronisation von Hervorhebungen[16] und der Möglichkeit durch einen Klick auf den Knoten in der Szene zu aktivieren. Um diese Funktionen zu ermöglichen durfte die Aktivierung nicht mehr im Skript ChartMarkers stattfinden. Sie musste über den ChartManager aufgerufen werden, um alle aktiven Graphen zu finden und den zum Knoten gehörigen Eintrag, wenn enthalten, hervorzuheben. Außerdem mussten durch einen Klick auf ein hervorgehobenes Objekt alle zugehörigen Markierungen auch wieder deaktiviert werden und es musste zwischen Hervorhebungen welche durch hovern und denen welche durch richtige Klicks ausgelöst wurden unterschieden werden. Im Nachhinein hätten einige dieser Funktionen ausgelagert werden sollen, um eine bessere Wartbarkeit zu ermöglichen.

[14] siehe Nachricht 2 in Abbildung 4.8
[15] siehe Nachricht 7 in Abbildung 4.8
[16] siehe Abschnitt 4.3.2

4.5 Integration mit vorhandenem Code

Während der Implementierung musste natürlich auf dem vorhandenen Code des SEE City Projektes aufgebaut werden. Anders als bei der Generierung von Städten, welche über ein vorhandenes Skript in die Szene geladen werden konnten, musste der vorhandene Code aber in zwei konkreten Fällen direkt verändert werden.
Erstens bedurfte es einer Möglichkeit, alle Metriken eines Graphen auszulesen. Ursprünglich war es nur möglich den Wert aller Metriken auszulesen, falls der Name dieser bekannt war. Da dies nicht der Fall war, musste die Liste der Namen aller Metriken öffentlich lesbar gemacht werden, was während der Entwicklung bei einem Refactoring auf dem Hauptbranch übernommen wurde.
Zweitens wurde parallel zu dieser Arbeit ein neues Eingabesystem entwickelt, welches auf den Hauptbranch übernommen wurde. Dieses sollte von direkten zugriffen auf Unitys eigenes Inputsystem oder dem von SteamVR abstrahieren und im Code Zugriffe auf Eingaben verallgemeinern, indem nur noch auf beispielsweise eine Bewegungskomponente eines Skripts zugegriffen wurde. Die Unterscheidung zwischen Bewegung in VR und Bewegung am Desktop musste so nicht mehr stattfinden. Damit dieses System bei Abgabe der Arbeit auch genutzt werden konnte, waren viele Anpassungen an den vorhandenen Skripts nötig. Die Steuerung der Graphen musste außerdem in das neue Eingabesystem mit eingebaut werden.

4.6 Tests

Der Erfolg einer Software hängt von vielen Faktoren ab. Darunter zählen die Nutzererfahrung, durchdachte Funktionalität, die Vermarktung, aber auch die Stabilität. Wie zuverlässig kann der Nutzer mit der Software die vorgesehenen Aufgaben abschließen? Kommt es oft zu Abstürzen, so wird die Arbeit mit der Software frustrierend und der Nutzer sucht sich einen anderen Weg seine Arbeit fertigzustellen. Zur Gewährleistung der Stabilität wird in größeren Teams oft und früh getestet. In Scrum meist nach jedem Sprint, mindestens aber vor jeder Auslieferung.
In dieser Arbeit musste möglichst effizient vorgegangen werden, um in der kurzen Zeit eine möglichst funktionale Software zu Entwickeln. Deshalb war der Plan zu Anfang, dass nach der Implementierung und vor der Evaluation eine große Testphase stattfinden sollte, in der die gesamte Funktionalität auf die häufigsten Fehler geprüft wird. Schnell wurde klar, dass das aus zwei Gründen nicht funktionieren würde. Erstens ist es schwer möglich die Software um Funktionen zu erweitern, die auf bestehenden Funktionen aufbauen, wenn diese bestehenden Funktionen nicht zuverlässig laufen. Würde nicht bereits während der Entwicklung regelmäßig getestet werden, so bestünde die Gefahr, dass in der großen Testphase Fehler mit schwer auffindbarem Ursprung auftreten. Diese bereits zu beseitigen bevor sie sich im neuen Code ausbreiten können, minimiert den Folgeaufwand. Zweitens nimmt eine Testphase, vor

welcher noch nicht getestet wurde, viel nicht vorhandene Zeit in Anspruch. Der Plan änderte sich schnell und es wurde durch frühzeitige Tests eine bereits möglichst fehlerfreie Software bis zur Evaluation entwickelt. Darauf hin sollte je nach verbleibender Zeit eine weitere, ausführlichere Testphase durchgeführt werden.

4.6.1 Automatische Tests

Um die Test leichter erneut durchführbar zu machen, wurden automatische Testfälle erstellt. Mit diesen lassen sich viele Interaktionen mit der Software simulieren, indem eine Testszene geladen wird um in dieser beispielsweise onClick() Events auf Schaltflächen auszulösen oder Verhalten bei verschiedenen geladenen Städtetypen zu Prüfen.

4.6.2 Manuelle Tests

Da einige Interaktionen mit dem System nur schwer über den Code zu Simulieren sind, wurden diese verhalten durch manuelle Testfälle überprüft. Sie sind in einer separaten Datei zu finden, in welcher sie jeweils mit Beschreibung, Vorbedingungen, Durchführung, erwartetem Ergebnis und tatsächlichem Ergebnis dokumentiert sind. Diese Tests wurden für beide Versionen der Software durchgeführt.

4.6.3 Bis zur Abgabe nicht behebbare und nennenswerte Fehler

Leider ist seit einem Update des Graphen Branches auf den Stand des Hauptbranches das auslesen von Metriken der Straßen Knoten nicht mehr funktionsfähig. Auf Commit 7765d930 vom 21.04.2020 funktionierte dies noch.
Schon länger von Fehlfunktionen versehen sind die Hervorhebungen. Dazu siehe Abschnitt 4.4.
Nicht fehlerhaft aber verbesserungswürdig ist die Bedienung der Graphen per Pointer in VR. Dazu siehe Abschnitt 6.4.1.

KAPITEL 5

Evaluation

Mit einer abgeschlossenen Implementierung, welche bisher jedoch nur auf meiner Vision und dem gelegentlichen Feedback der Prüfer und einigen Kommilitonen bestand, blieb die Frage offen, wie gut das Ergebnis seinen Zweck erfüllt. Anhand einer Evaluation konnte dies überprüft werden.

5.1 Was sind Evaluationen?

Eine Evaluation wird von Hegner (2003) als systematisches Sammeln, Auswerten und Interpretieren von Daten, um eine reliable und valide Bewertung der Benutzungsschnittstelle zu ermöglichen, beschrieben. „Dabei wird aus den Ergebnissen der Evaluation abgeleitet, ob ein vorab definiertes Designziel erreicht ist bzw. ob und wo weitere Verbesserungsmöglichkeiten ausgeschöpft werden können.“ (Hegner, 2003) Dies wird oft anhand einer Nullhypothese gezeigt, welche widerlegt werden sollte, um eine zweite Hypothese das Erwartete Ergebnis darstellende Hypothese zu erfüllen. Dabei können Evaluationen zu allen Zeitpunkten während der Entwicklung durchgeführt werden. Schon vor Beginn der Implementierung mit Mockups[1], während der Implementierung mit Prototypen oder, wie in meinem Fall, mit dem fertigen Produkt.

5.2 Variablen Typen

Bei vielen Tests möchte man die Auswirkungen von Anpassungen am System auf den Nutzer messen. Diese getätigten Anpassungen, sowie Reaktionen darauf, werden Variablen genannt. Sie werden in drei verschiedene Typen eingeteilt.

[1] siehe Abschnitt 3.2

5.2.1 Abhängige Variablen

Dies sind die Variablen, welche die Ergebnisse der Evaluation darstellen. Ihr Wert verändert sich nach Hegner (2003) in Bezug auf unabhängige- und Störvariablen. Es könnte zum Beispiel die Zufriedenheit der Probanden oder die Zeit, welche für eine Aufgabe gebraucht wird, beziehungsweise die Benutzbarkeit des Systems als Abhängige Variable gemessen werden.

5.2.2 Unabhängige Variablen

In einer Evaluation soll der Einfluss der unabhängigen Variablen auf die abhängigen Variablen gemessen werden. Diese werden also während des Versuches absichtlich variiert (vgl. Hegner (2003)). In dieser Evaluation ist die hauptsächliche unabhängige Variable die Version der Software, also ob der Proband in VR oder der Desktop Version testet.

5.2.3 Kontrollierte Variablen

Natürlich beeinflussen nicht nur die unabhängigen Variablen, was für Werte die abhängigen Variablen annehmen. Deshalb soll von Proband zu Proband und von Aufgabe zu Aufgabe möglichst wenig verändert werden was Einfluss auf die Ergebnisse haben kann. Zum Beispiel sollte der Raum beim VR Test immer die gleichen oder am besten gar keine Hindernisse enthalten, die dem Probanden in den Weg kommen könnten. Auch sollte immer der gleiche Bildschirm, die gleiche Maus und Tastatur und der gleiche Stand der Software verwendet werden. All solche Variablen werden als kontrolliert bezeichnet.

5.2.4 Störvariablen

Wird eine Variable auf keine Weise gemessen oder kontrolliert, so fällt sie nicht weg und kann trotzdem Einfluss auf die Ergebnisse nehmen (vgl. Hegner (2003)). Auf die leicht erhöhte Temperatur im Raum kann an heißen Tagen beispielsweise kaum Einfluss genommen werden, auch wenn der Proband möglicherweise mit einer anderen Laune die Fragebögen ausfüllt. Solche Variablen werden als Störvariablen bezeichnet.

5.3 Aufbau der Evaluation

Zur Durchführung der Evaluation wurden Dokumente benötigt, welche den Ablauf begleiten sollten oder auf welchen die Erfahrung der Probanden festgehalten werden konnten. In den folgenden Abschnitten wird der Aufbau und Zweck dieser Dokumente erläutert.

5.3.1 Einverständniserklärung

Zur eigenen Sicherheit und der Sicherheit der Probanden geschieht vor jeder Durchführung einer Erklärung der Rechte. Den Probanden wird in diesem[2] Dokument erklärt, warum welche Daten gesammelt werden und wie sie verarbeitet werden. Für genauere Details sollte das Dokument selbst aufgerufen werden. Die Probanden geben am Ende eine Unterschrift ab, mit welcher sie zustimmen freiwillig an der Evaluation mit Daten- und, auf explizite Zustimmung, Videoaufnahme teilzunehmen.

5.3.2 Einführung

Nicht nur die Elemente, welche vor der Evaluation vorbereitet werden können haben als Variablen Einfluss auf die Ergebnisse. Der Proband selbst bringt Eigenschaften mit sich, welche nach Möglichkeit am besten mit unter die kontrollierten Variablen[3] gebracht werden sollten. Dazu wird auf diesem[4] Fragebogen einiges an Vorwissen abgefragt, welches relevant sein könnte. Außerdem dient eine Zeile für sonstige Anmerkungen zur Notation vom Tragen einer Brille oder ähnlichem. Dazu kann der Proband hier entscheiden, welcher Testgruppe er zugeordnet werden möchte. Je nach Wahl wird er die Einführung in die erste Aufgabe in der VR oder Desktopversion durchführen. Das Ziel ist jedoch eine ungefähr ausgeglichene Zahl an Probanden in beiden Gruppen zu haben, weshalb diese Entscheidung dem Probanden bei Bedarf genommen wird.

5.3.3 Aufgaben

Der wichtigste Teil der Evaluation auf einem scheinbar unwichtigen Dokument. Aber nur auf den ersten Blick. Denn warum sollten die Aufgaben nicht einfach während oder kurz bevor der Proband diese erledigt erklärt werden? Dadurch könnte besser darauf eingegangen werden, wie die Person die Fragestellung versteht und so ein besseres Verständnis erzielt werden. Das ist das Problem. In der Evaluation sollen möglichst alle Variablen kontrolliert[3] sein. Liest jeder Proband die Aufgaben für sich selbst und stellt gegebenenfalls Fragen dazu, sind in der Hinsicht weniger Störvariablen[5] vorhanden. Werden die Aufgaben aber jedes Mal mündlich erklärt, so erfährt in diesem Bereich jeder Proband eine, wenn auch minimal, unterschiedliche Aufgabenstellung. Da dies sich auf die Ergebnisse der gemessenen abhängigen Variablen auswirken könnte, existiert eine Niederschrift der Aufgaben[6].
Dem Probanden werden noch einmal einige Hinweise zur Durchführung der Aufgaben gegeben, wie dass die Software getestet wird und nicht er selbst. Darauf folgt die richtige

[2] siehe Anhang A.1
[3] siehe Abschnitt 5.2.3
[4] siehe Anhang A.2
[5] siehe Abschnitt 5.2.4
[6] siehe Anhang A.3

Aufgabenstellung mit dem Hinweis, diese Aufgabe nach Abschluss noch einmal im jeweils anderen System (VR oder Desktop) durchzuführen.

5.3.4 Fragebogen

Der Fragebogen[7] ist das Herzstück dieser Evaluation. Die meisten Informationen zur Bewertung der Software werden hier erhoben. Er ist in zwei Teile aufgeteilt. Hauptsächlich werden Fragen mit quantitativen Antwortmöglichkeiten gestellt, aber auch einige mit qualitativen Antwortmöglichkeiten. Das ermöglicht es, die beiden Antworttypen in Verbindung zu bringen, um genauere Schlüsse zu ziehen. Die quantitativen Daten alleine würden bei der geringen erwarteten Teilnehmerzahl wenig aussagen. Schreibt jedoch ein Proband zu den qualitativen Antworten, dass er unzufrieden mit der Bedienbarkeit war, so kann dies wahrscheinlich in den quantitativen Ergebnissen erkannt werden. Ist dieser Grund häufiger angegeben, so kann eventuell bei Probanden die nicht nicht den selben Grund angeben die gleiche Tendenz in den quantitativen Ergebnissen gefunden werden. Andernfalls gehört der Proband, welcher sich über die Bedienbarkeit beschwert wahrscheinlich zu einer Ausnahme.

Fragen mit quantitativen Antwortmöglichkeiten

Garbarino und Holland (2009) beschreibt die quantitative Analyse wie folgt:

> „Quantitative research produces data in the form of numbers [...]“

Es werden also Daten in Form von Zahlen erfasst. Das kann zum Beispiel durch multiple Choice Fragen erreicht werden, bei denen zu jeder möglichen Antwort die Anzahl, wie oft diese gewählt wurde, aufsummiert wird. Es kann aber auch auf einer Skala eine Anzahl von Punkten vergeben werden, wie auf dem Fragebogen[7] der für diese Evaluation entworfen wurde. Genauer genommen wurde ein standardisiertes Verfahren namens System Usability Scale (SUS) verwendet. Dieses zeichnet sich vor allem durch seine als häufig „quick and dirty“ beschriebene Charakteristik aus. Da den Probanden nicht allzu viel Zeit abverlangt werden sollte, aber die durch das SUS Verfahren erlangten Informationen vielsagend genug sind, wurde es gewählt. Andere Verfahren wurden aufgrund von zu vielen oder zu spezifischen Fragen die zu lange dauern würden oder Fragen welche irrelevante Informationen erhoben ausgeschlossen.
Die Fragen des SUS Verfahrens sollen von den Probanden je ein mal nach Abschluss der Desktop-Variante und nach Abschluss der VR Variante ausgefüllt werden. Dabei wird auf einer Skala von „stimme gar nicht zu“ bis „stimme voll zu“ angekreuzt. Die gewählten Werte können in der Auswertung dann als Werte von eins bis fünf verrechnet werden. So liegen nach einigen Durchläufen Durchschnittswerte für Erlernbarkeit, Konsistenz, Integration,

[7] siehe Anhang A.4 und A.4

Komplexität etc. für beide Varianten der Software vor, welche untereinander verglichen werden können.

Fragen mit qualitativen Antwortmöglichkeiten

Zur qualitativen Analyse schreibt (Garbarino und Holland, 2009) folgendes:

> „[...] qualitative research tends to produce data that are stated in prose or textual forms."

Es handelt sich also um erhobene Daten in schriftlicher oder zum Beispiel gesprochener Form. Auch Skizzen oder ähnliches würden in diese Kategorie von Daten zählen. Die durch diesen Fragebogen aufgenommenen qualitativen Daten sollen dazu dienen, die durch die SUS erfassten Daten sinnvoll zu ergänzen. Genauer sollen die Probanden hier die Aspekte der Software nennen, die ihnen am stärksten negativ oder positiv aufgefallen sind. Auch ist hier Raum für Vorschläge von Ergänzungen zu der Software. Das ermöglicht es Rückschlüsse auf die durch SUS erhobenen quantitativen Daten zu ziehen. Gibt ein Proband zum Beispiel an, dass nach Frage 8 die Software umständlich zu benutzen fand, so geht er bei könnte noch verbessert werden eventuell genauer ins Detail und nennt vielleicht dazu passend zu kleine Schaltflächen.

5.4 Hypothese zur Evaluation

Dieser Abschnitt wurde geschrieben, bevor die Evaluation durchgeführt wurde, um meine Erwartungen an die Ergebnisse und die Software an sich unvoreingenommen von diesen widerzuspiegeln.

Allgemeines

Es wird angenommen, dass die Durchführung der Evaluation fehlerfrei und sauber verlaufen wird, da die Software vorher auf Fehler getestet wurde und der Ablauf der Evaluation ein paar mal geprobt wurde. Nichtsdestotrotz ließen sich nicht alle Fehler beseitigen und die Graphen wurden nicht gänzlich in die bestehende Software eingebunden, was aber für die Evaluation eventuell von Vorteil ist. So sind einige Funktionen, welche andernfalls ablenkend wirken könnten deaktiviert und die Probanden können sich auf die Benutzung der Graphen fokussieren. Ein bekannter Fehler ist, dass die Hervorhebung der Einträge und Knoten falsch funktioniert, sobald der Nutzer über mehrere Einträge in der Seitenleiste hovert. Dies stellt aber kein Problem für die Bedienbarkeit dar.

Erwartete Ergebnisse des Fragebogens

Bezogen auf den Fragebogen[8], welcher im Großteil auf eine Quantitative Analyse der Nutzbarkeit des Systems fokussiert ist, nehmen die im vorherigen Abschnitt genannten Erwartungen wahrscheinlich auch Einfluss auf diesen. Die Fehler werden höchstwahrscheinlich die Ergebnisse von Fragen drei, fünf und sechs beeinflussen, da das System schwerer zu benutzen ist wenn falsche Einträge hervorgehoben werden. Da die Highlights inkonsistent sind wirkt das System schlechter integriert. Frage eins wird an die Vorerfahrung der Probanden gebunden sein. Wenn diese wissen, was sie mit einem System dieser Art für Aufgaben bearbeiten können, so ist es wahrscheinlicher dass diese eher zustimmen das System häufig zu benutzen. Im Falle der VR Bewertung der selben frage werden weniger erfahrene Probanden wahrscheinlich eher zustimmen um noch einmal in VR arbeiten zu können. Dieser „Zauber“ verfliegt aber mit der häufigeren Benutzung. Da das System andernfalls möglichst simpel gehalten wurde sollten die restlichen Fragen im Vergleich eher positiver ausfallen.

5.5 Auswertung

Mit der Durchführung der Evaluation wurden Ergebnisse in verschiedensten Formen gesammelt. In den folgenden Abschnitten werden diese Ergebnisse vereinfacht und übersichtlich dargestellt um diese darauf hin auszuwerten.
Leider wurde die Evaluation durch die Corona Krise beeinflusst. Weil auch nach langem warten eine Durchführung an der Universität nicht möglich war, musste diese privat mit Familie und Bekannten durchgeführt werden. Das führte dazu, dass sich ein Großteil der Probanden wenig unter möglichen Anwendungsfällen der Software vorstellen konnten.

5.5.1 Fachausdrücke zur Auswertung

In der Auswertung der Ergebnisse werden einige Begriffe genutzt, welche einige Leser wohl erst noch einmal nachschlagen müssten. Deshalb werden diese hier kurz erklärt.

Mittelwert ($\bar{X}$)

Lange und Bender (2007) definieren diesen im allgemeinen Sprachgebrauch als „Durchschnitt“ bekannten Wert als „die Summe aller beobachteten Werte geteilt durch die Gesamtzahl der Beobachtungen“. Im Gegensatz zum Median fließen hier die Ausreißer, also stark von der Standardantwort abweichende Werte, deutlicher mit ein (Lange und Bender, 2007).

[8] Siehe Anhang A.4 und A.4

Standardabweichung (σ)

Um den Einfluss der Ausreißer auf den berechneten Mittelwert zu bestimmen, wird die Standardabweichung mit angegeben. Diese gibt die durchschnittliche Entfernung aller Messungen zu einem Merkmal vom Durchschnitt an (Statista, 2018).

5.5.2 Ergebnisse

Um eine bessere und schnelle Übersicht über die Ergebnisse zu erhalten, werden diese im Folgenden zusammengefasst. Wie bereits erwähnt werden „stimme gar nicht zu“ und „stimme voll zu“ mit Werten von eins bis fünf ersetzt.

Vorwissen

Quantitative Ergebnisse wurden entworfen um leicht auswertbar zu sein. Deshalb werden hier nicht alle Antworten, sondern nur der Mittelwert dieser, sowie die zugehörige Standardabweichung angegeben.

Frage	$\bar{X}$	σ
Software Visualisierung	1,38	0,52
VR	2	1,07
Tastatur/Maus Gaming	3,25	1,75
Grafische Darstellungen	2,38	0,74

Tabelle 5.1: Mittelwert ($\bar{X}$) und Standardabweichung (σ) der Ergebnisse des Vorwissens

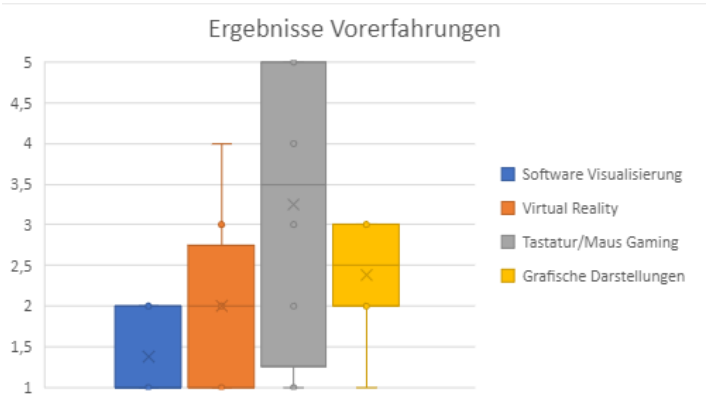


Abbildung 5.1: Visuelle Darstellung der Vorkenntnisse

Beobachtungen Da nicht mit Nutzern der Zielgruppe getestet wurde hatten diese wenig Erfahrung, was Software-Visualisierung angeht. VR hatten einige Nutzer schon einmal

ausprobiert. Anhand der Vorerfahrungen in Tastatur und Maus Gaming lassen sich die Probanden aus den zwei Testgruppen unterscheiden. Die Informatiker haben hier für gewöhnlich einen deutlich höheren Wert angegeben. Wie die Probanden selber sagten, kamen die Vorerfahrungen welche sie zu grafischen Darstellungen hatten meist aus dem Mathematik Unterricht aus der Schule.

Quantitative Ergebnisse des Fragebogens

Hier wird ähnlich vorgegangen, wie in Kapitel Vorwissen. Es wird der Mittelwert ($\bar{X}$) mit Standardabweichung (σ) der Antworten angegeben. Gerundet wird auf zwei Nachkommastellen.

	Frage	Desktop $\bar{X}$	σ	VR $\bar{X}$	σ
1	Ich denke, dass ich das System gerne häufig benutzen würde.	2,13	0,99	3,13	0,83
2	Ich fand das System unnötig komplex.	2,13	0,64	2	1,07
3	Ich fand das System einfach zu benutzen.	3,88	0,83	3,75	1,04
4	Ich glaube, ich würde die Hilfe einer technisch versierten Person benötigen, um das System benutzen zu können.	3,25	0,89	3,5	0,93
5	Ich fand, die verschiedenen Funktionen in diesem System waren gut integriert.	3,43	0,79	3,71	0,76
6	Ich denke, das System enthielt zu viele Inkonsistenzen.	2	1,07	1,88	1,13
7	Ich kann mir vorstellen, dass die meisten Menschen den Umgang mit diesem System sehr schnell lernen.	4,38	0,74	4,13	0,99
8	Ich fand das System sehr umständlich zu nutzen.	2,13	0,35	2,38	0,92
9	Ich fühlte mich bei der Benutzung des Systems sehr sicher.	3,75	1,04	4,13	0,99
10	Ich musste eine Menge lernen, bevor ich anfangen konnte das System zu verwenden.	3	1,07	3	0,93

Tabelle 5.2: Mittelwert ($\bar{X}$) und Standardabweichung (σ) der Ergebnisse des Fragebogens

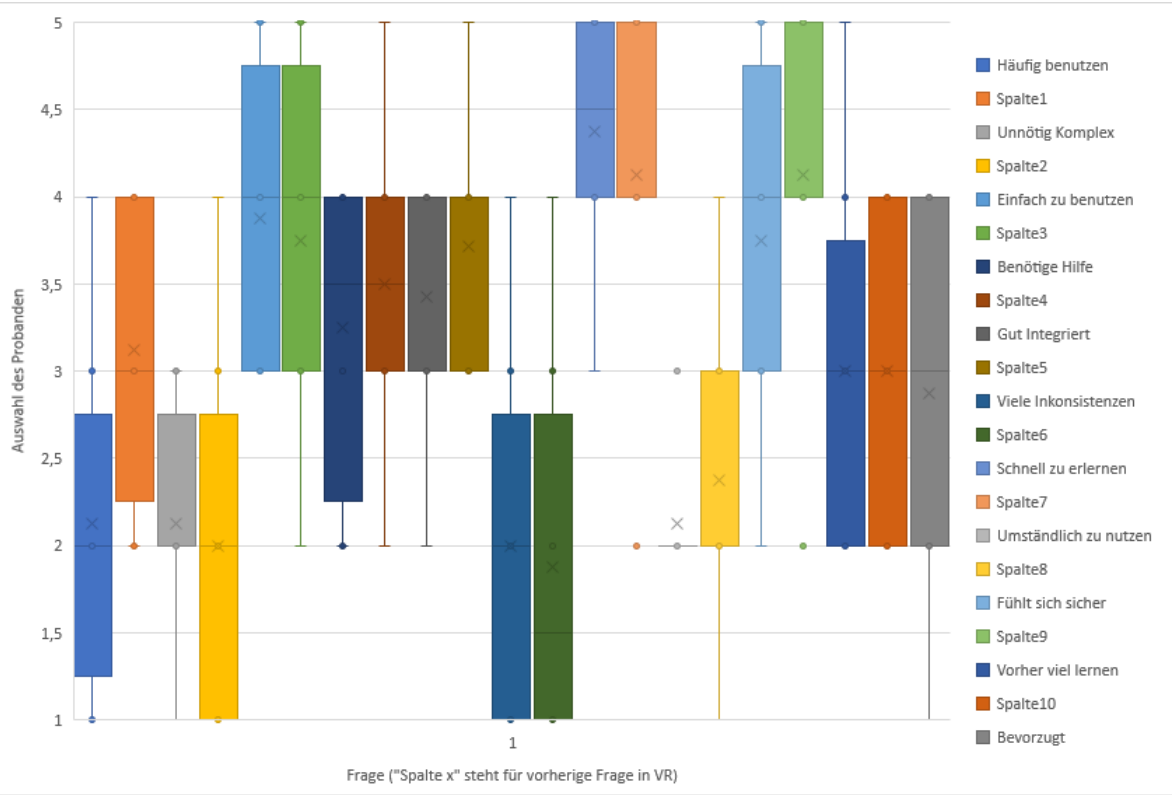


Abbildung 5.2: Visuelle Darstellung der Ergebnisse

Qualitative Ergebnisse des Fragebogens

Es gab auf dem Fragebogen mehrere Fragen zu welchen qualitative Antworten geleistet werden konnten. Zu jeder dieser Fragen wurden die häufigsten und wichtigsten Antworten entnommen und hier aufgelistet. Außerdem wurden während der Durchführung kleinere Interviews mit den Probanden durchgeführt, von welchen die wichtigen Ergebnisse auch mit hier aufgenommen werden. Diese sind von mir nach deren Relevanz und Häufigkeit geordnet.

1. Warum wurde die gewählte Version der Software bevorzugt?
   (a) Desktop
      i. Effizienter
      ii. Mehr Vorerfahrungen
   (b) VR
      i. Intuitive Bedienung
      ii. Bessere räumliche Orientierung
      iii. Gewohnte Bewegung
      iv. Mehr Spaß

2. Was kann noch verbessert werden?
   (a) Design
   (b) Scrollgeschwindigkeit der Maus
   (c) Lesbarkeit der Schrift
   (d) Auswahl der Metrik für die y-Ache auch horizontal
   (e) Bewegung in der Stadt (beim Test durch einen Sensor beschränkt)
   (f) Erklärungen der wählbaren Metriken
3. Was hat gefehlt?
   (a) Mehr Erklärungen, zum Beispiel von Metriken, für unerfahrene Nutzer
   (b) Light/Dark Mode
   (c) Musik und Sounds
4. Was sollte sich bei einer Verbesserung möglichst wenig ändern (was war gut)?
   (a) Die Bedienung
   (b) Die Wahl der Farben
5. Liegt dir noch etwas auf dem Herzen?
   (a) Der Sinn der Stadt wurde nicht verstanden. Warum wird nicht nur die grafische Darstellung genauer ausgearbeitet?

5.5.3 Auswertung der Ergebnisse

Betrachtet man die qualitativen, so wie sie quantitativen Ergebnisse der Evaluation in Kombination mit den Beobachtungen während der Durchführung, so lassen sich einige Aussagen über die getestete Software treffen. Vor allem soll geprüft werden, ob kleinere Unterschiede von Desktop zu VR Version in den quantitativen Ergebnissen aus den qualitativen Ergebnissen hergeleitet werden können und wie diese sonst miteinander zusammen hängen.

Von den Probanden bevorzugte Version

Aus den Ergebnissen lässt sich erkennen, dass die Probanden die Desktop Version der Software bevorzugten. Diese Wahl fiel jedoch nur bei Berücksichtigung der Ausreißer knapp aus. Andernfalls schnitten beide Versionen ähnlich ab. Viele konnten sich nicht zwischen den Versionen unterscheiden oder eine der beiden nur minimal bevorzugen. Für die Desktop Version wurden jedoch mehr Gründe gegeben, warum diese gewählt wurde. Sie stellt sich als effizienter dar und die Probanden hatten mehr Vorerfahrungen bei der Nutzung dieser. Für die VR Version sprach häufig der Spaß bei der Nutzung.

Beobachtungen und Zusammenhänge des Fragebogens

1. Das System häufig benutzen Hier schneidet die VR Version deutlich besser ab. Das hängt vermutlich damit zusammen, wie wenige der Probanden vorher Erfahrungen mit HMDs gesammelt haben und dass die meisten keine Verwendung für die Software an sich haben. Somit überwiegt für sie der Faktor Spaß über Effizienz, was deutlich in der Aussage „VR macht mehr Spaß, aber müsste ich Professionell damit arbeiten so würde ich die Desktop Version benutzen“ eines Probanden zu erkennen ist. Wohl durch diesen Spaß an der Nutzung fällt auch die Antwort für Frage 9[9] entsprechend aus.

2. Komplexität der Software Mit Absicht wurde die Benutzung der Graphen möglichst simpel gehalten. Somit unterscheiden die Probanden hier wohl anhand der Eingabemöglichkeit. Wie mit allen neuen Dingen, gibt es Menschen die schnell lernen und die, welche langsamer lernen. Somit fällt die Antwort für die VR Version deutlich breiter gestreut aus als die bekannte Bedienung über Maus und Tastatur. Der Mittelwert liegt aber für beide Varianten ungefähr im selben Bereich.

4. Benötigt Hilfe Generell galt für beide Versionen der Software, dass viele Probanden nicht direkt verstanden wie genau die Stadt Daten darstellt, und wo diese Daten her kommen. In der VR Version kommt zusätzlich hinzu, dass viele Probanden beim Aufsetzen und Einstellen des HMD und der Nutzung der Controller unsicher waren.

5. Die Funktionen des Systems waren gut Integriert Auch hier ist das minimal positivere Ergebnis für die VR Version wahrscheinlich auf die Erstnutzung dieser[9] zurückzuführen.

7. Schnell zu erlernen Für eine zukünftige Umsetzung von Funktionen, welche auf VR ausgelegt sind, spricht die hohe Erlernbarkeit der Technologie. Diese fiel höher aus als die der Desktop Version. Das liegt wahrscheinlich daran, dass sich hier auf grundlegende Bedienung konzentriert wurde, während auf dem Desktop eher die Erlernbarkeit der Software an sich bewertet wurde.

8. Das System war umständlich zu Nutzen Auch hier fällt die Streuung der Ergebnisse deutlich größer für die VR Version aus, wobei die Mittelwerte ähnlich ausfallen. Diese Beobachtung hängt wahrscheinlich stark mit „2. Komplexität der Software“ zusammen und ist ähnlich zu begründen.

[9] siehe „9. Fühlt sich sicher“

9. Fühlt sich sicher Entgegen meiner Vermutungen haben viele Probanden angegeben, dass sie sich in der VR Version sicherer fühlten. Ich denke jedoch, dass diese Aussage mit dem „Zauber“ welcher auf der ersten Benutzung von VR liegt zusammenhängt. Wer erst abgeschreckt von der Benutzung ist wird schnell von dem leichten Eintauchen in die Virtuelle Welt beeindruckt, so dass die Kontrolle in dieser erst einmal zu hoch eingeschätzt wird.

10. Vorher viel lernen Da beide Werte sehr ähnlich ausfallen gilt auch hier wie in „4. Benötigt Hilfe“, dass die Probanden relativ unerfahren waren, was Visualisierung von Software angeht. Hätten die Probanden diese Frage auf andere Aspekte der Software bezogen, so gäbe es wahrscheinlich deutlichere Unterschiede zwischen den Versionen.

Design und Zugänglichkeit

Die Probanden haben außerdem einige interessante Anmerkungen gemacht, welche aus den quantitativen Ergebnissen nicht direkt auslesbar gewesen wären. Von kleineren Verbesserungsvorschlägen bei der Bedienung, der Schriftgröße oder der Farbwahl der Graphen wurde sich von einigen eine bessere Zugänglichkeit gewünscht. Die Probanden schlugen zum Beispiel Beschreibungen zu den Einzelnen Metriken vor um besser zu Verstehen was genau von diesen dargestellt wird.

KAPITEL 6

Diskussion

Auch wenn aufgrund der aktuellen Corona-Krise nur wenige, eingeschränkte Evaluationen stattfinden konnten, war es möglich aus den Ergebnissen einige Schlüsse zu ziehen. Wie brauchbar diese sind und ob bei der Implementierung die gewünschten Ziele erreicht wurden soll in diesem Kapitel diskutiert werden.

6.1 Signifikanz der Ergebnisse

Bevor genauer auf die Ergebnisse eingegangen wird, muss folgendes klargestellt werden:

6.1.1 Demografie

Zum einen überwiegen bei den durchgeführten Evaluationen aufgrund der Einschränkungen zwei Gruppen von Nutzern. Circa die Hälfte der Nutzer kann zu den Technik-affinen Menschen gezählt werden, da diese dem Informatikbereich der Uni Bremen angehörten. Die meisten dieser Nutzer beschäftigen sich Täglich mit ihrem Computer und hatten schnell verstanden wie und warum aus Software Städte gebaut werden.
Die andere Hälfte der Nutzer musste aus Mangel an Testpersonen aus Freunden und Familienmitgliedern gebildet werden, wodurch aber auch interessante Eindrücke außerhalb der Informatik in die Ergebnisse einflossen. Nutzer dieser Gruppe haben zwar Erfahrung mit dem Umgang mit Computern, hatten jedoch Schwierigkeiten das Konzept der Stadt zu verstehen und sich an die Bewegung durch die Stadt und Bedienung der Graphen, welche der von Spielen stark ähnelt, zu gewöhnen.

6.1.2 Gewissheit der Aussagen

Zum anderen sollten die Ergebnisse und Deutungen dieser[1] nicht als eindeutig gesehen werden. Zwar können gewisse Tendenzen interpretiert werden, welche sich aber auch aus Zufall

[1] siehe Abschnitt 5.5.3

entwickelt haben könnten. Mit der Durchführung von weiteren Tests sollten sich diese Tendenzen jedoch in erkennbare Unterschiede entwickeln, sodass die jetzigen Interpretationen der Ergebnisse mit mehr Klarheit bestätigt werden können.

6.2 Bedeutung der Ergebnisse

Schaut man sich alle Ergebnisse aus Abschnitt 5.5.3 an, so können aus diesen mehrere Aussagen geschlossen werden.

6.2.1 „VR macht Spaß“

Eine Aussage die in Gesprächen mit den Probanden und als Begründung für die Bevorzugung einer der beiden Versionen häufig gefallen ist lautet „VR macht mehr Spaß.“. Eine Tendenz zu dieser Aussage ist auch anhand der Beobachtung aus „1. Das System häufig benutzen“ der Auswertung bei den Probanden zu erkennen, welche sich nicht dazu geäußert haben. Leider ist es nicht möglich aus den Ergebnissen zu schließen, wie lange sich die VR Version durch den Spaß-Faktor behaupten kann.

6.2.2 Desktopversion ist effizienter

Eine weitere Aussage einiger Probanden lautete „Mit der Desktop Version kann ich effizienter Arbeiten.“. Auch hier lässt sich die Aussage in den quantitativen Ergebnissen erkennen. Die Frage welche Version leichter zu bedienen ist, wurde mit minimalem Unterschied zugunsten der Desktopversion beantwortet. Ob die Desktopversion unnötig komplex ist haben die Nutzer einheitlicher beantwortet als bei der VR Version, bei welcher die Verteilung der Ergebnisse deutlich höher ausfiel. Ein einheitliches Erlebnis, auch wenn nicht besser, gibt nach Stone u. a. (2005) ein Gefühl von Konsistenz. Deutliche Unterschiede sind bei der benötigten Hilfe, der Sicherheit bei der Benutzung und der Umständlichkeit zugunsten dieser These zu erkennen.

6.2.3 VR ist gewöhnungsbedürftig

Im Verhalten der Probanden während des Versuches und in den Ergebnissen einiger der Fragen des Fragebogens fällt auf, dass die Nutzung von VR manchen leichter fällt während andere Schwierigkeiten beim Einstieg haben. Sei es also durch die zum Desktop abweichende Art der Interaktion mit Daten oder das ungewohnte Bewegungsgefühl, welches bei zwei Probanden sogar ein leichtes Schwindelgefühl hervorgerufen haben, so ist die Benutzung in VR, wie auch Rüdel u. a. (2018) in ihrer Studie feststellten, erst einmal deutlich gewöhnungsbedürftiger.
In den Ergebnissen der Fragebögen kann beobachtet werden, dass keiner der Probanden vorher viel Erfahrungen in VR hatte. Eine geringe Standardabweichung spricht für geringe

Unterschiede in der Erfahrung der Probanden. Einige haben jedoch schon viele Erfahrungen in Spielen am Computer, welche mit Tastatur und Maus bedient werden und höchst wahrscheinlich haben alle Probanden einen Computer schon einmal zum Arbeiten genutzt. Dies spiegelt sich in den Antworten auf die Frage wie umständlich die Software zu nutzen war wider. In der Desktop-Variante haben hier bis auf einen alle Probanden dieselbe Antwort von zwei oder „nicht sehr umständlich“ gegeben. Dass die Gewöhnung an VR von Person zu Person unterschiedlich lange dauert, lässt sich daran erkennen, dass die Antworten zu der VR Version zwar im Schnitt ähnlich ausfielen, jedoch die Standardabweichung dreimal so hoch war. VR ist also keine Technologie, mit der jeder sofort etwas anfangen kann.

6.2.4 Mehr Probleme in VR

Generell leidet VR darunter, dass sich durch das viel breitere Eingabe und Wahrnehmungsspektrum deutlich mehr Hürden für die Benutzung auftun. Das sind zum einen Dinge wie die schon vorher angesprochene Vorbereitung der Brille für den Einsatz welche je nach Umfeld und Nutzer variiert. Zum anderen treten aber auch aufseiten der Software Probleme auf, die in der Desktopversion nicht präsent wären. So war während der Evaluation nur ein Sensor aufgebaut. Das bedeutet, dass die Probanden sich in der Spielwelt nicht um 360° Umsehen konnten, sondern nur so weit wie die Brille nach vorne hin erkannt wurde. Solch ein Grenzfall lässt sich zwar schnell durch eine Rotationsmöglichkeit per Controller umgehen, während der Durchführung der Evaluation hatten Probanden aber keine andere Wahl als die Stadt nur aus einem Blickwinkel zu betrachten.

6.2.5 Benutzeroberfläche und Bedienung

Die von den Probanden genannten Probleme bei der Bedienung wurden bereits größtenteils behoben. Zu der Benutzeroberfläche gab es einige Anmerkungen, wie dass Texte in VR zu klein wären oder das Scrollen nicht gut funktioniere. Solche Bemerkungen wurden schon in einem sehr frühen Zustand der Software geäußert. Die Schwierigkeit war, die Software automatisch zwischen VR und nicht VR wechseln zu lassen. Eine Möglichkeit wäre gewesen, zwei Versionen aller Prefabs anzufertigen und den Code darauf anzupassen. Dadurch wäre dieser allerdings komplexer und Änderungen an der Benutzeroberfläche hätten jedes Mal doppelt ausgeführt werden müssen. Da versucht wurde eine gemeinsame Benutzeroberfläche für beide Versionen der Software zu bauen, mussten hier Kompromisse in Bezug auf Bedienung und Lesbarkeit eingegangen werden. Die negativen Auswirkungen dieser wurden zum Ende jedoch minimal gehalten.

6.2.6 Zugänglichkeit

Wie in „Design und Zugänglichkeit“ aus Kapitel 5 beschrieben, haben sich einige Probanden bessere Erklärungen für Metriken gewünscht. Dieser Wunsch könnte generalisiert werden, um die gesamte Software anhand von Tutorials und Beschreibungen zugänglicher zu gestalten.
Leider waren kaum Probanden teil der Zielgruppe der Software. Aus diesem Grund sollte bevor solche Funktionen umgesetzt werden ein Test mit Probanden der Zielgruppe durchgeführt werden. Diese sollten wahrscheinlich besser ohne Hilfe mit der Software umgehen können. Ist dies der Fall, so ist eine Umsetzung der genannten Funktionen weniger wichtig bis unnötig.

6.2.7 Fazit zur Evaluation

Zwar kann aus der geringen Menge an Daten keine handfeste Aussage geschlossen werden, jedoch lässt sich eine Tendenz erkennen, wie das Produkt dieser Arbeit von den Probanden aufgenommen wurde. Die wohl stärkste Aussage, welche für die VR Version spricht, ist der Spaß beim Benutzen dieser. Da es sich bei der Software aber um ein Produkt zur Nutzung am Arbeitsplatz handelt und nicht um ein Spiel ist das der falsche Grund für die Bevorzugung von VR. Viele Probanden haben ähnliche aussagen wie „VR macht mehr Spaß, aber müsste ich professionell damit arbeiten so würde ich die Desktopversion benutzen“ getroffen.
Diese Ergebnisse in Beachtung gezogen ist die VR-Version der Software also noch nicht bereit die Desktop-Variante zu ersetzen. Bis sich entweder VR neben Desktop PC und Mobiltelefon in moderne Haushalte etabliert hat und somit vertrauter und weniger umständlich ist oder die VR-Version Features bietet, welche sie von der Desktopversion absetzen, ist die ebenso mächtige Desktop Version also das Mittel der Wahl.

6.3 Was spricht für VR?

Die durchgeführte Evaluation bezog sich größtenteils auf die Nutzung der Graphen. Den Probanden wurde zwar gezeigt, wie sie sich in der Stadt bewegen können, jedoch wurde kein Fokus auf diese Funktion gelegt. Im momentanen Zustand der Software lässt sich die Nutzung von VR anstelle der Desktop Variante rechtfertigen, wenn die Stadt ohne die Graphen ausgewertet oder bearbeitet werden soll. Mit VR hat der Nutzer nach Chance u. a. (1998) eine bessere Orientierung und kann Größenverhältnisse besser einschätzen als über den Computerbildschirm.
Sollen in Zukunft die Vorteile von VR für die Graphen genutzt werden, so könnte man die in Abschnitt 4.1.7 erwähnte 3D-Darstellung implementieren um Graphen mehr wie richtige Bestandteile der Stadt darstellen, welche vom Nutzer angefasst, gedreht und aus mehreren Richtungen betrachtet werden können. Dies wäre in der Desktopversion schwieriger umzusetzen.

6.4 Verbesserung der Software

Aus den Ergebnissen der Evaluation und den eigens durchgeführten Tests und Beobachtungen lassen sich abseits der bereits genannten Makel noch andere Verbesserungsmöglichkeiten finden.

6.4.1 Performance

Laden der Daten für einen Graphen

Ein Problem der Graphen ist, dass sie mit großen Städten schlecht klarkommen. Zwar wird ab einer gegebenen Anzahl an Knoten in der Stadt die Berechnung angepasst, jedoch finden zu viele Aufrufe innerhalb von Schleifen statt, welche wiederum in einer Schleife aufgerufen wird. Im Folgenden ist ein Beispiel dafür zu sehen, welches bei jedem Erstellen eines Graphen aufgerufen wird:

Listing 6.1: Schleife innerhalb einer Schleife in ChartContent.

```
private void GetAllFloats()
    {
      foreach (var data in _dataObjects)
      foreach (var key in data.GetComponent<NodeRef>().node.
          FloatAttributes.Keys)
        if (!AllKeys.Contains(key))
          AllKeys.Add(key);
    }
```

Fachlich ausgedrückt läuft diese Methode nach Krone u. a. (2003) mit einer Komplexität von $\mathcal{O}(n^2)$. Das bedeutet, dass die Laufzeit exponentiell mit der Anzahl der Knoten wächst. Eine lineare Komplexität, welche hier für ein besseres Verhalten angestrebt werden sollte, wird mit $\mathcal{O}(n)$ beschrieben. Jeder zusätzliche Knoten würde also die Laufzeit um dieselbe Menge an Zeit verlängern.

Raycasts

In Unity werden Raycasts genutzt, um zum Beispiel bei einem Mausklick eine Gerade von der Kamera aus in die Spielwelt zu ziehen und kollidierende Objekte anzuzeigen. In VR werden diese genutzt, um den Pointer am rechten Controller zu realisieren. Dieser dient zur Interaktion mit Objekten, wäre ohne Raycasts jedoch rein visuell.
Klickt der Nutzer auf ein Objekt, so wird ein Strahl vom Controller gesendet, welcher dann das erste kollidierende Objekt zurückliefert. Leider benötigt dieser Prozess viel Performance, sodass beim Verschieben von Objekten nicht schnell genug neue Strahlen gesendet werden.

Das Objekt wird dann nicht schnell genug mit der Verschiebung des Nutzers mit bewegt und der Nutzer „rutscht“ von diesem ab. Damit wird die Verschiebung abgebrochen.
Leider müsste wahrscheinlich die gesamte Eingabemethode per Pointer anders umgesetzt werden, um dieses Problem zu beheben.

6.4.2 Refactoring

Wie bereits an einigen Stellen erwähnt, ist der Code nicht optimal strukturiert. Die Klassen ChartMarker und ChartContent beinhalten zu viele Funktionen, welche ausgelagert werden könnten. Dies würde für die Zukunft die Wartbarkeit und Erweiterbarkeit der Software verbessern.

6.5 Bewertung der Vorgehensweise

Um aus den Erfahrungen welche in dieser Arbeit gewonnen wurden zu lernen, sollen die angewandten Arbeitsweisen noch einmal diskutiert werden.
Alles in allem wurde für die Implementierung absichtlich weniger vorausgeplant, damit während der Arbeit besser auf Änderungen eingegangen werden konnte. Dies hat sich als sinnvoll erwiesen, da der Aufwand bei dem vergleichsweise geringen Umfang der Arbeit minimiert wurde. Einzig bei den Hervorhebungen hätte es geholfen im Voraus alle Funktionen für welche diese eingesetzt werden zu kennen und diese darauf auszulegen. Da während der Entwicklung neue Funktionen hinzukamen für welche der Code nicht ausgelegt war, weisen die Hervorhebungen einige Fehler auf.
Die freie Entwicklung hat unter anderem deshalb so gut funktioniert, da ein Kanban-Board genutzt wurde. Mit diesem war es sehr leicht den Überblick über derzeitige Baustellen zu behalten und neue Ideen schnell als Eintrag auf diesem festzuhalten. Anfangs wurde jedoch in die Beschreibungen und Übersicht der Karten so viel Zeit investiert, als würden andere diese verstehen müssen. Da alleine mit dem Board gearbeitet wurde, war das jedoch nicht nötig.
Während der Entwicklung wurden einige male Kommilitonen eingeladen, um die Software im Entwicklungsstand zu testen. Dies hat meist sehr geholfen und hätte öfter geschehen sollen. So wurden viele kleinere Makel vor allem in der Bedienung gefunden und behoben.

KAPITEL 7

Zusammenfassung und Ausblick

7.1 Zusammenfassung

Im Verlauf dieser Arbeit wurde die bestehende SEE Software um eine Funktion erweitert, die in der Stadt dargestellten Metriken auch über Graphen anzuzeigen. Dies kann entweder über eine klassische Desktop-Ansicht, oder über ein HMD in VR geschehen.
Durch eine Evaluation, welche die beiden Versionen miteinander vergleichen sollte hat sich ergeben, dass die Desktopversion der Software zum aktuellen Stand wahrscheinlich die zu bevorzugende Version ist, da mit dieser effizienter und fehlerfreier gearbeitet werden kann. Dennoch bot auch VR Vorteile und keine der beiden Versionen erwies sich als gänzlich unbrauchbar. Da in dieser Arbeit ein System für den universellen Einsatz über Desktop und VR entworfen werden sollte, kann dies als Erfolg gesehen werden. Trotz dessen bleiben Optionen zur Verbesserung offen.

7.2 Ausblick

7.2.1 Unterscheidung zwischen VR und Desktop

Im Kapitel „Diskussion“ wurden schon einige Verbesserungsmöglichkeiten für die Graphen genannt. Dazu gehörten eine bessere Umsetzung in VR, indem Graphen dreidimensional dargestellt werden. Generell können die beiden Versionen untereinander weiter separiert werden, um die jeweiligen Vorteile weiter auszunutzen. So könnte am Desktop eine eher informelle Ansicht der Graphen dargestellt werden, während in der VR Version eher auf Interaktivität und eine Integration mit der Spielwelt geachtet wird.

7.2.2 Sinnvolle ergänzende Funktionalitäten

Auch eine Zoomfunktion könnte nach den ursprünglichen Anforderungen eingeführt werden, um Teile eines stark befüllten Graphen genauer zu analysieren. Über anpassbare Minimal- und Maximalwerte der Achsen eines Graphen, welche eine ähnliche Funktion erfüllen würden,

wurde während der Implementierung auch nachgedacht. Zusätzlich wäre eine Möglichkeit Graphen oder Listen zu den Metriken eines bestimmten Knotens zu erzeugen interessant. So könnten alle Metriken dieses Knotens mit einem schnellen Blick untereinander verglichen werden, ohne für jede einen Graphen zu erzeugen.
Um über die bereits integrierten Darstellungsarten mehr Informationen einsehen zu können, könnte das Ablesen der genauen Werte von Einträgen, zum Beispiel mit Achsenbeschriftungen und Hilfslinien, vereinfacht werden.

7.2.3 Fehlerbehebung und aufräumen des Codes

Sollten die Graphen in Zukunft weiterentwickelt werden, so gäbe es keinen Umweg um die Behebung der bekannten Probleme, wie die Fehlfunktionen bei den Hervorhebungen. Dazu sollte der Code durch Auslagerungen von Funktionen Lesbarer und Wartbarer gemacht werden.

7.2.4 Integration mit SEE

Außerdem wird das SEE Projekt stets angepasst und um neue Funktionen erweitert. So sollten auch die Graphen an diese Änderungen angepasst werden und für neue Funktionen wie den Mehrspieler-Modus oder das Entwerfen von Städten nutzbar gemacht werden.

ANHANG A

Unterlagen zur Evaluation

Abbildung A.1: Einverständniserklärung

Grafische Darstellung der Metriken des SEE City Projekts
Durchgeführt durch: Robert Bohnsack (robboh@gmx.de)
Universität Bremen

Einverständniserklärung

Herzlich Willkommen und vielen Dank für die Teilnahme!

Dir wurde die Nummer auf den nächsten Zetteln aus zwei Gründen zugeordnet. Erstens möchte ich damit meine aufgenommenen Ergebnisse von dir als Person abstrahieren, zweitens soll sie dazu dienen, die Ergebnisse dieses Tests später noch zueinander zuordnen zu können. Da dir unter anderem die Ergebnisse trotzdem noch zugeordnet werden könnten, bitte ich dich diese Einverständniserklärung aufmerksam zu lesen und bei Einwilligung zu unterzeichnen.

Ablauf des Versuches:

Du bist heute hier, um mir als Testperson zu helfen, einen Vergleich zwischen zwei Versionen einer Software aufzustellen und die generelle Benutzbarkeit dieser zu evaluieren. Dazu wirst du ein paar kleine Aufgaben mit der Software erledigen. Ein Teil des Tests findet in Virtual Reality statt, deshalb denk daran:

Es ist, vor allem bei dem Test in VR aber auch generell während des ganzen Versuches, kein Problem eine Pause einzulegen oder ihn abzubrechen.

Zwischendurch darfst du ein paar Fragen beantworten und das war es dann auch schon.

Die folgenden Daten werden gesammelt und deinem Testdurchlauf anhand deiner Probandennummer zugeordnet:

Während du die verschiedenen Aufgaben erledigst, werde ich mir Notizen dazu machen, wie du mit dem System interagierst und wie effizient sich die Aufgaben damit erledigen lassen. Dazu gehören auch Aufnahmen wie die Zeit pro Aufgabe oder die Anzahl der Interaktionen mit bestimmten Schaltflächen und Ähnlichem.

Zwischen den Aufgaben werde ich dich bitten, ein paar Fragen auf einem Fragebogen zu beantworten.

Aufnahme von Bildschirmvideos:

Zum besseren Verständnis bei der Auswertung würde es mir sehr helfen, deine Interaktionen mit dem System in Form von mehreren Bildschirmvideos aufzunehmen. Dabei bist du nicht auf den Videos zu sehen und diese werden nur deiner Probandennummer zugeordnet. Beim VR Teil des Versuches wird zwar die virtuelle Realität in der du dich befindest aufgenommen, jedoch nur aus deiner Sicht. Auch in der virtuellen Realität bist du also nicht selbst zu sehen. Dem musst du allerdings nicht zustimmen.

- ❑ Ja, ich stimme der Aufnahme von Videos zu.
- ❑ Die aufgenommenen Videos dürfen im Rahmen der Bachelorarbeit veröffentlicht werden.

Abbildung A.2: Einführung

Einführung

Fangen wir an, Proband # __! Damit du den Aufgaben die du später bearbeiten darfst gleich besser gewappnet bist, möchte ich dir einige Minuten Zeit geben um dich mit dem System vertraut zu machen. Dabei gebe ich dir die Wahl, ob du das in VR oder am Desktop machen möchtest:

- ❑ VR
- ❑ Desktop

Bevor wir anfangen, würde ich gerne ein bisschen zu deiner Vorerfahrung mit einigen Elementen der Software wissen. Beantworte dafür bitte folgende Fragen bzw. Aussagen:

Studierst du? Wenn ja, was?

_ _

Ich habe vorher schon mit Software Visualisierung (zum Beispiel als Stadt) gearbeitet und kenne mich damit aus:

Stimme gar nicht zu				Stimme voll zu
❑	❑	❑	❑	❑

Ich habe bereits Vorerfahrungen mit Virtual Reality über Head Mounted Displays und habe diese bereits oft genutzt:

Stimme gar nicht zu				Stimme voll zu
❑	❑	❑	❑	❑

Ich spiele regelmäßig First- oder Third Person Spiele mit Tastatur und Maus Eingabe:

Stimme gar nicht zu				Stimme voll zu
❑	❑	❑	❑	❑

Ich arbeite häufiger in anderer Software mit grafischen Auswertungen und Darstellungen (zum Beispiel Excel, R oder GeoGebra):

Stimme gar nicht zu				Stimme voll zu
❑	❑	❑	❑	❑

Sonstige Anmerkungen:

_ _

Abbildung A.3: Aufgaben

Deine Aufgaben

Hallo! Um die Ergebnisse möglichst unabhängig von der Erklärung der Aufgaben zu machen erhältst du diese in Papierform. Bei den Aufgaben geht es außerdem nicht darum, dass du sie richtig löst, sondern wie du sie löst und dabei mit der Software umgehst. Stelle währenddessen gerne Fragen.

Erste Aufgabe

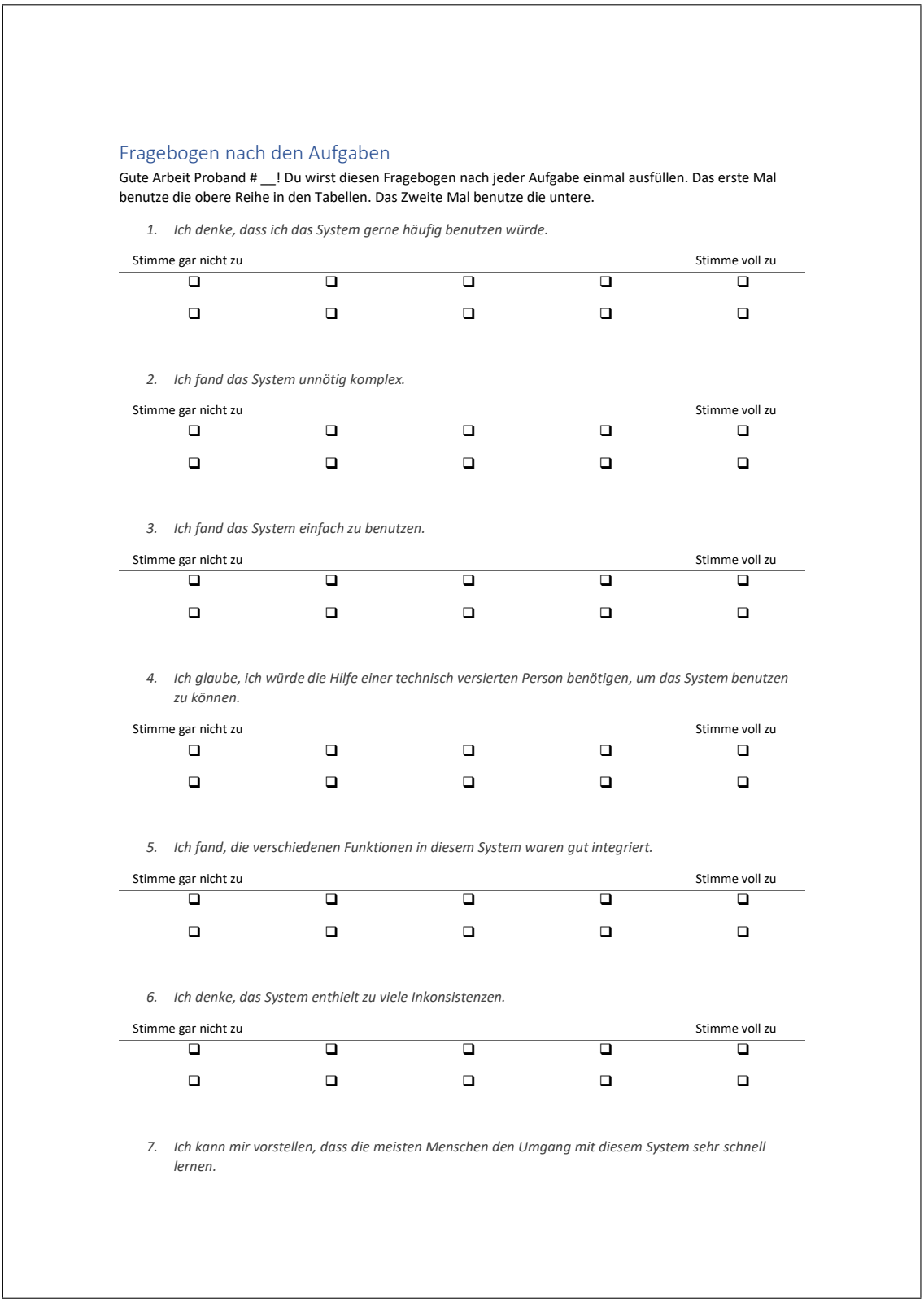
Angenommen du hast ein kleines Programm geschrieben und möchtest herausfinden, wo du am besten noch Verbesserungen einbringen kannst. Dazu könntest du SEE City benutzen. Versuche bei der Lösung der Aufgaben möglichst die Graphen zu nutzen, auch wenn es andere Möglichkeiten geben könnte um eine Lösung zu finden.

1. Zuerst solltest du die Auswahl an Gebäuden anhand der Leiste rechts ein wenig einschränken. Lasse deinen Graphen nur Gebäude mit mehr als 0 “Dead_Code” (Metrik) anzeigen.
2. Jetzt möchtest du herausfinden, wo es in deinem Code viel duplizierten Code gibt. Die Menge davon wird von der Metrik “Clone” angegeben. Sag Bescheid, wenn du das zugehörige Gebäude gefunden hast.
3. Es könnte sein, dass dieses Gebäude zwar viel duplizierten Code hat, aber gleichzeitig auch sehr viele Zeilen. Die Metrik “LOC” gibt Auskunft über die Zeilenanzahl. Versuche herauszufinden, bei welchem Gebäude der duplizierte Code pro Zeile am höchsten ist. Dazu solltest du einfach schätzen aber könntest zwei Graphen für eine bessere Übersicht benutzen.

Zweite Aufgabe

Bearbeite die erste Aufgabe nun noch einmal, nur in der jeweils anderen Version der Software zu der die du in der Ersten Aufgabe genutzt hast. Du wirst zwar den Ablauf schon kennen, sollst aber in der Lage sein, beide Versionen vergleichen können.

Abbildung A.4: Fragebogen: Erste Seite



Fragebogen nach den Aufgaben

Gute Arbeit Proband # __! Du wirst diesen Fragebogen nach jeder Aufgabe einmal ausfüllen. Das erste Mal benutze die obere Reihe in den Tabellen. Das Zweite Mal benutze die untere.

1. *Ich denke, dass ich das System gerne häufig benutzen würde.*

Stimme gar nicht zu				Stimme voll zu
❑	❑	❑	❑	❑
❑	❑	❑	❑	❑

2. *Ich fand das System unnötig komplex.*

Stimme gar nicht zu				Stimme voll zu
❑	❑	❑	❑	❑
❑	❑	❑	❑	❑

3. *Ich fand das System einfach zu benutzen.*

Stimme gar nicht zu				Stimme voll zu
❑	❑	❑	❑	❑
❑	❑	❑	❑	❑

4. *Ich glaube, ich würde die Hilfe einer technisch versierten Person benötigen, um das System benutzen zu können.*

Stimme gar nicht zu				Stimme voll zu
❑	❑	❑	❑	❑
❑	❑	❑	❑	❑

5. *Ich fand, die verschiedenen Funktionen in diesem System waren gut integriert.*

Stimme gar nicht zu				Stimme voll zu
❑	❑	❑	❑	❑
❑	❑	❑	❑	❑

6. *Ich denke, das System enthielt zu viele Inkonsistenzen.*

Stimme gar nicht zu				Stimme voll zu
❑	❑	❑	❑	❑
❑	❑	❑	❑	❑

7. *Ich kann mir vorstellen, dass die meisten Menschen den Umgang mit diesem System sehr schnell lernen.*

Abbildung A.4: Fragebogen: Zweite Seite

Stimme gar nicht zu				Stimme voll zu
☐	☐	☐	☐	☐
☐	☐	☐	☐	☐

8. *Ich fand das System sehr umständlich zu nutzen.*

Stimme gar nicht zu				Stimme voll zu
☐	☐	☐	☐	☐
☐	☐	☐	☐	☐

9. *Ich fühlte mich bei der Benutzung des Systems sehr sicher.*

Stimme gar nicht zu				Stimme voll zu
☐	☐	☐	☐	☐
☐	☐	☐	☐	☐

10. *Ich musste eine Menge lernen, bevor ich anfangen konnte das System zu verwenden.*

Stimme gar nicht zu				Stimme voll zu
☐	☐	☐	☐	☐
☐	☐	☐	☐	☐

Nach dem Zweiten ausfüllen fahre hier fort

Welche Version der Software bevorzugst du?

Desktop				Virtual Reality
☐	☐	☐	☐	☐

Gibt es einen ausschlaggebenden Grund zur Beantwortung der vorherigen Frage?

_ _

Was kann noch verbessert werden?

_ _

Was hat dir gefehlt?

_ _

Was sollte sich bei einer Verbesserung möglichst wenig ändern?

_ _

Liegt dir noch etwas auf dem Herzen?

_ _

ABBILDUNGSVERZEICHNIS

LITERATURVERZEICHNIS

[Alam und Dugerdil 2007] Alam, Sazzadul ; Dugerdil, Philippe: EvoSpaces Visualization Tool: Exploring Software Architecture in 3D. In: *14th Working Conference on Reverse Engineering (WCRE 2007)*, 2007, S. 269–270

[Burdea und Coiffet 2003] Burdea, Grigore C. ; Coiffet, Philippe: *Virtual Reality Technology*. 2. Wiley Interscience, 2003

[Chacon und Straub 2014] Chacon, Scott ; Straub, Ben: *Pro Git*. 2. Apress, 2014

[Chance u. a. 1998] Chance, S. S. ; Gaunet, F. ; Beall, A. C. ; Loomis, J. M.: Locomotion Mode Affects the Updating of Objects Encountered During Travel: The Contribution of Vestibular and Proprioceptive Inputs to Path Integration. In: *Presence* 7 (1998), Nr. 2, S. 168–178

[Christensson 2020] Christensson, Per: *TechTerms*. 2020. – URL `https://techterms.com`. – Zugriffsdatum: 17.03.2020

[Garbarino und Holland 2009] Garbarino, Sabine ; Holland, Jeremy: Quantitative and Qualitative Methods in Impact Evaluation and Measuring Results. (2009), S. 7. – URL `http://www.gsdrc.org/docs/open/eirs4.pdf`. – Zugriffsdatum: 09.03.2020

[Hegner 2003] Hegner, Marcus: *Methoden zur Evaluation von Software*. 2003

[IEEE 1993] IEEE: IEEE Standard for a Software Quality Metrics Methodology. In: *IEEE Std 1061-1992* (1993), S. 1–96

[Johnson und Shneiderman 1991] Johnson, B. ; Shneiderman, B.: Tree-maps: a space-filling approach to the visualization of hierarchical information structures. In: *Proceeding Visualization '91*, 1991, S. 284–291

[Krone u. a. 2003] Krone, Joan ; Ogden, William F. ; Sitaraman, Murali: *OO Big O: A Sensitive Notation for Software Engineering*. 2003

[Lange und Bender 2007] Lange, Stefan ; Bender, R.: Median oder Mittelwert? 132 (2007). – URL `https://www.thieme-connect.com/products/ejournals/pdf/10.1055/s-2007-959024.pdf`. – Zugriffsdatum: 10.07.2020

[Object Management Group 2017] Object Management Group: *Unified Modeling Language*. 2017. – URL `https://www.omg.org/spec/UML/`. – Zugriffsdatum: 20.03.2020

[Rivero u. a. 2010] Rivero, José M. ; Rossi, Gustavo ; Grigera, Julián ; Burella, Juan ; Luna, Estaban R. ; Gordillo, Silvia: From Mockups to User Interface Models: An Extensible Model Driven Approach. In: *Lecture Notes in Computer Science* Bd. 6385, Springer, 2010, S. 13 – 24. – ISBN 978-3-642-16985-4

[Rüdel u. a. 2018] Rüdel, M. ; Ganser, J. ; Koschke, R.: A Controlled Experiment on Spatial Orientation in VR-Based Software Cities. In: *2018 IEEE Working Conference on Software Visualization (VISSOFT)*, 2018, S. 21–31

[Statista 2018] Statista: *Definition Standardabweichung.* 2018. – URL `https://de.statista.com/statistik/lexikon/definition/126/standardabweichung/`. – Zugriffsdatum: 10.07.2020

[Steinbrückner 2013] Steinbrückner, Frank: *Consistent Software Cities : supporting comprehension of evolving software systems*, Dissertation, 06 2013

[Stone u. a. 2005] Stone, Debbie ; Jarrett, Caroline ; Woodroffe, Mark ; Minocha, Shailey: *User Interface Design and Evaluation.* Morgan Kaufmann Publishers, 2005

[Trümper 2014] Trümper, Jonas: *Visualization Techniques for the Analysis of Software Behavior and Related Structures*, Universität Potsdam, Dissertation, 2014

[Unity Technologies 2019] Unity Technologies: *Unity Scripting API (2019.3).* 2019. – URL `https://docs.unity3d.com/ScriptReference/index.html`. – Zugriffsdatum: 28.02.2020

[Waller u. a. 2013] Waller, Jan ; Wulf, Christian ; Fittkau, Florian ; Döhring, Philipp ; Hasselbring, Wilhelm: SynchroVis: 3D Visualization of Monitoring Traces in the City Metaphor for Analyzing Concurrency. In: *2013 First IEEE Working Conference on Software Visualization (VISSOFT)*, 2013, S. 1–4