Universität Bremen

Fachbereich 3: Mathematik und Informatik

Bachelorarbeit

Orientierung in VR-basierten Software-Cities mit hierarchisch gebündelten Abhängigkeiten

– Evaluation spatialer Orientierung in den Immersionsäquivalenzklassen Desktop und Head Mounted Display –

Johannes Ganser, Marc O. Rüdel

Matrikel-Nr. 265 801 1, 810 705

14. August 2017

1. Gutachter: Prof. Dr. rer.nat. Rainer Koschke

2. Gutachter: Prof. Dr. Gabriel Zachmann

Betreuer: M.Sc. Marcel Steinbeck

Johannes Ganser, Marc O. Rüdel

Orientierung in VR-basierten Software-Cities mit hierarchisch gebündelten Abhängigkeiten

– Evaluation spatialer Orientierung in den Immersionsäquivalenzklassen Desktop und Head Mounted Display –

Bachelorarbeit, Fachbereich 3: Mathematik und Informatik

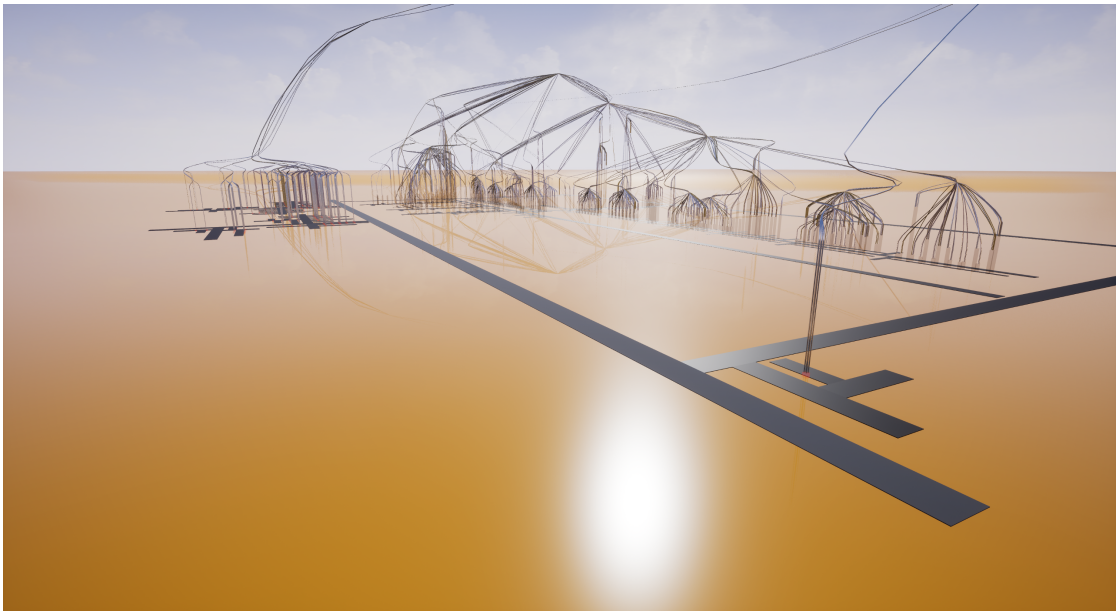
Universität Bremen, April 2018

Selbstständigkeitserklärung

Hiermit erkläre wir, dass wir die vorliegende Arbeit selbstständig angefertigt, nicht anderweitig zu Prüfungszwecken vorgelegt und keine anderen als die angegebenen Hilfsmittel verwendet haben. Sämtliche wissentlich verwendete Textausschnitte, Zitate oder Inhalte anderer Verfasser wurden ausdrücklich als solche gekennzeichnet.

Bremen, den 14. August 2017

Johannes Ganser, Marc O. Rüdel



Danksagung

Wir danken für Ihre direkte oder indirekte Unterstützung bei der Realisierung unser Bachelorarbeit allen, die daran beteiligt waren. Zuerst möchten Prof. Dr. rer.nat. Rainer Koschke dafür danken, dass er unsere Arbeit angenommen, betreut und durch technische Unterstützung ermöglicht hat. M.Sc. Marcel Steinbeck und Dipl.-Inf. Dipl.-Psych. Martin Schröer danken wir für ihre inhaltiche Unterstützung und dafür, dass sie immer ein offenes Ohr für uns hatten. Ingrid Bode danken wir für ihre organisatorische Unterstützung. Johannes Gronewold danken für für seine Bereitschaft, uns einen Einblick in seine Arbeit als Software Qualitätsmanager zu geben. Wir danken unseren Familien und Freunden, die uns zur rechten Zeit den Rücken frei gehalten und unter die Arme gegriffen haben. Und – last-but-not-least – danken wir den anonymen Probanden, die sich bereits erklärten, an unserer Evaluation teilzunehmen.

Wenn wir in dieser Arbeit aus Gründen der Einfachheit das generische Maskulinum verwenden, wir beziehen uns damit aber stets ausdrücklich auf beide Geschlechter.

Zusammenfassung

Die Realität ist perfekt, sie hat nur ein Problem: Man kann Vorstellungen und Träume nicht real machen.
— Jaron Lanier, Zitat aus 'Nightline', 1993

In dieser Arbeit befassten wir uns mit der Fragestellung, wie stark sich die menschliche Orientierung in VR-basierten Software-Cities mit hierarchisch gebündelten Verbindungen zwischen zwei Systemen unterschiedlicher Immersionsäquivalenzklassen unterscheidet. Wir realisierten dazu für die Game Engine *Unreal Engine 4.15* eine Visualisierungserweiterung, die dynamisch aus GXL-Dateien einen Softwaregraphen einlesen und parametrisierbar visualisieren kann. Das GXL-Dateiformat ist ein Standard-Austauschformat im Software-Reengineering-Bereich, welches Sourcecode als Graph, erweitert um Metrikdaten, enthält. Unser Versuchsplan stellte zwei Software-Modelle gleicher Lines-Of-Code, aber unterschiedlicher Klassen- und Methodenanzahl, mit jeweils zwei Metriken einander gegenüber. Wir konnten für unsere Evaluation 12 Probanden gewinnen, die in den Software-City-Visualisierungen Orientierungsaufgaben erledigen sollten. Wir maßen die Effektivität, Effizienz und Fehler, wobei wir Effizienz als Geschwindigkeit definierten. Wir konnten zeigen, dass sich die Orientierungseffizienz zwischen den beiden Immersionsäquivalenzklassen nicht unterscheidet. Allerdings scheint die Komplexität des Modells im Sinne der Gebäudehöhe die Orientierungseffizienz zu beeinflussen.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Definitionen

Beispiele

Kapitel 1

Einleitung [MOR]

Virtual Reality ist bereits seit über 25 Jahren ein wissenschaftliches Forschungsgebiet. In den letzten Jahren gewann Virtual Reality allerdings durch die rasante Weiterentwicklung der Display- und Grafikkartenhardware zunehmend an praktischer Bedeutung. David Eden fragt sogar Anfang des Jahres, ob 2017 eventuell das Jahr der Virtual Reality werde. Virtual Reality-Systeme werden in immer mehr nicht-spielerischen Zusammenhängen eingesetzt. So werden Virtual Reality-Anwendungen zu Schulungszwecken, für medizinische Anwendungen, als Informationsplattform, und zur kooperativen Arbeit eingesetzt.

Das Grundprinzip heutiger Virtual Reality-Systeme ist recht simpel und auch nicht wirklich neu[1] Neu allerdings ist, dass die negativen Effekte älterer Systeme – namentlich allgemeine Übelkeit und die sogenannte Motion Sickness bzw. Cyber Sickness – der ausgereiften Hardware zum Dank überwunden zu sein scheinen. Damit lassen sich VR-Systeme auch in betrieblichen Zusammenhängen einsetzen, was zuvor aufgrund der gesundheitlichen Bedenken problematisch war.

Auf der Meta-Ebene betrachtet erzeugen Informatiker interessanterweise einerseits hochkomplexe und flexible Systeme (wie Steuerberechnungssysteme, Flugzeugsteuerungen oder auch Virtual Reality-Systeme), wir nutzen dazu aber andererseits oftmals Entwicklungswerkzeuge, die sich seit dem Beginn des Computerzeitalters funktional nicht wesentlich verändert haben.[2] Die Volksweisheit, wonach der Schuster die schlechtesten Schuhe hat[3] gilt offenbar auch für uns Informatiker: Auch heute greifen wir immer noch gelegentlich zu einfachen Texteditoren.

Neben vielen anderen Themen sind in den letzten Jahren die Themen *Wartung* und *Teamarbeit* im Zusammenhang mit der Entwicklung von Informatiksystemen in den Fokus gerutscht: Wartungsaufgaben fällt durch neue Projektorganisationsmethoden ein stärkeres Gewicht zu, die Wartung

[1] mehr dazu in Kapitel 2.1

[2] Hier könnte man jetzt eine Grundsatzdiskussion darüber führen, wie deutlich heutige Entwicklungssysteme denen aus den Anfangsjahren überlegen sind. Diese Diskussion wollen wir nicht führen, wir wollen nur attestieren, dass der Fortschritt der Entwicklungswerkzeuge dem der entwickelten System deutlich hinterher hinkt.

[3] „Der Schuster hat die schlechtesten Schuhe."

von bestehenden Altsystemen wird zunehmend wichtiger und damit auch die Einarbeitung in diese oftmals schlecht dokumentierten oder in nicht mehr gelehrten Programmiersprachen verfassten Systeme. Der Themenkomplex des Software-Reengineerings befasst sich mit diesen Herausforderungen und versucht Lösungen zu finden, mit denen die Wartbarkeit von Informatiksystemen verbessert werden kann.

Mit Metriken als Teil des Software-Reengineerings hat der Programmierer oder der Qualitätsmanager etablierte Methoden zur Qualitätskontrolle durch ermittelte, interne Systemkennzahlen an der Hand. Mithilfe dieser Metriken lassen sich normalerweise unsichtbare System-Eigenschaften ingenieurmäßig „messen“ und vergleichen. Durch den Vergleich der „Messwerte“[4] mit Erfahrungswerten lassen sich damit Aussagen über die Qualität der Software treffen. Heutzutage werden oftmals Metriken auch als Vertragsgrundlage bei der Auftragsprogrammierung verwendet.

Bei großen System werden entsprechend viele Daten zu unterschiedlichen Metriken erzeugt. Die schiere Menge an Daten schmälert die Interpretationsfähigkeit und damit deren Aussagekraft. Die Lösung des Problems scheint die *Informationsgrafik* zu sein.

Mit grafischen Visualisierungen werden seit dieser Zeit Informationen auf die wesentlichen Aspekte abstrahiert und einfacher zugänglich gestaltet. Informationsgrafiken sind aus unserer Zeit nicht mehr wegzudenken und auch in der Informatik wurden sie schon immer eingesetzt, um Sachverhalte oder Abläufe zu verdeutlichen[5]. Dabei versuchten sich die Informatik-Visualisierungen an den Möglichkeiten der Computergrafiken orientierten: in letzter Zeit 3D und hochauflösend. Steinbrückner baute in »Consistent Software Cities : supporting comprehension of evolving software systems« auf einer interessante Metapher für die Softwarevisualisierung auf: Software als Stadt. In seiner Arbeit stellt der Autor dar, wie Software und auch deren Entwicklungsstadien in Form einer Stadt konsistent visualisiert werden können.

1.1 Aufbau der Arbeit [MOR]

Fasziniert von den Möglichkeiten heutiger Virtual Reality-System stellten wir uns nun auf der Suche nach einem Einsatzgebiet, geleitet vom Traum an ein besseres Softwareentwicklungssystem, die Frage, welchen Effekt die Übertragung der Software City-Metapher auf ein Virtual Reality-System hätte. Haben Entwickler oder Qualitätsmanager oder vielleicht sogar Kunden einen Vorteil davon, wenn sie sich *in* Ihrer Software bewegen könnten? Hier bei interessierte uns besonders, ob man seine Software wieder erkennen könnte. Sie die Metapher tatsächlich anbieten würde sich in einer Software zurecht zu finden und mit anderen Stakeholdern darüber zu reden.

[4] Wir setzen hier die Begriffe *messen* und *Messwerte* in Anführungsstriche um darauf hinzuweisen, dass die Werte nicht wirklich *gemessen* werden. Mehr zur Ermittelung der Metriken und zu den Unterschieden zur klassischen Ingenieurwissenschaft in Kapitel 2.3.3

[5] siehe dazu Kapitel 2.2.4)

Aufgabenstellung Wir bekamen die Aufgabe, ein Virtual Reality-System zur Visualisierung der Software City-Metapher in Anlehnung an Steinbrückner zu erstellen, in dem basierend auf der angezeigten Hierarchieebene auch die vorhandenen Assoziationen zwischen den Software-Elementen anzuzeigen sind. Die Assoziationen sollen gebündelt werden. Als Datenbasis für das System sollen graphbasierte, mit metrischen Informationen versehene Daten im GXL-Format dienen, die von der Sourcecode-Analyse-Software Bauhaus Toolkit erzeugt wurden. Das System soll gegen ein Desktop-System mit einer identischen Darstellung evaluiert werden. Des weiteren sollten die Ansichten der Stadt verschiedene Hierarchische Ebenen anzeigen können und speicherbar sein. Die Informationen aus dem GXL-Dateien sollten außerdem innerhalb der Visualisierung interaktiv einsehbar sein.

In dieser Arbeit beschränkten wir uns auf die Visualisierung von statischen Softwaresichten. Für die grafische Darstellung und die Generierung der Virtual Reality-Sichten nutzen wir eine Game Engine. Unsere Arbeit beschreibt nun neben einige Grundlagen, als ersten großen Punkt wie wir die Daten in die Engine einlesen (siehe Kapitel A.6) und wir eine Software City generieren (siehe Kapitel 3.1.3).

Als zweiten großen Punkt wollen wir daher die Software City um eine Darstellung der Assoziationen erweitern.Zur Verbesserung der Übersichtlichkeit ermöglichen wir auch ein hierarchische Kantenbündelung (siehe dazu Kapitel 3.1.4).

Um die Effekte der Virtual Reality auf die Software City beurteilen zu können, schlossen wir an die Entwicklung unserer Virtual Reality-Systems eine Evaluation an. Das Ziel der Evaluation war einerseits, möglich Effekte von Virtual Reality-Umgebungen gegenüber Desktop-Systemen herauszuarbeiten, andererseits aber auch unser System, welches es in dieser kombinierten Form zuvor noch nicht gab, bewerten zu lassen. Die Evaluation selbst beschreiben wir in Kapitel 4 dort findet sich auch unsere These[6]. Die Ergebnisse sind im Kapitel 5, welche in Kapitel 6 diskutiert werden. Im Kapitel 7 ziehen wir unsere Schlüsse und im Kapitel zu Future Work geben wir einen Ausblick(siehe Kapitel 8).

[6] siehe Kapitel 4.4

Kapitel 2

Grundlagen und Stand der Forschung

Unsere Arbeit deckt verschiedene Wissensdomänen ab. In diesem Kapitel wollen wir Aspekte der jeweiligen Domänen erläutern, die für das weitere Verständnis unserer praktischen Arbeit und unserer Evaluation hilfreich sein können. Wir werden uns zunächst dem Begriff Virtual Reality nähern und seine Eigenschaften bestimmen und gegen andere Begriffe abgrenzen. Hiernach folgt ein Ausblick über die Geschichte und den Stand der Forschung im Bereich *Softwarevisualisierung*, bevor wir uns dem Thema *Software-Reengineering* widmen. Im Zusammenhang mit Virtual Reality kommt man häufig auf den Begriff *NUI* zu sprechen, den wir kurz darstellen um anschließend noch auf *Game Engines* zu sprechen zu kommen, die Grundlage unserer praktischen Arbeit bilden. Zum Schluss des Grundlagenkapitels stellen wir noch kurz wichtige Aspekte der Evaluation vor, die für unsere *Evaluation* von Bedeutung waren.

2.1 Virtual Reality [JG]

> Realität ist nicht, was geschieht.
> Realität ist, was wir empfinden.
> — Walter (Billy) Fürst, Schweizer Aphoristiker

Woher kommt der Begriff Virtual Reality? Um diese Frage zu beantworten, betrachtet wir als erstes die Wortherkunft und gehen danach tiefer auf die wissenschaftliche Bedeutung im Zusammenhang mit unserer Arbeit ein.

Was ist überhaupt Realität? Dem Duden zufolge versteht man im Deutschen darunter vor allem, „Wirklichkeit“[1] und darunter versteht man „[alles] das, Bereich dessen, was als Gegebenheit,

[1] *http://www.duden.de/rechtschreibung/Realitaet*, Aufruf: 28.09.2017

Erscheinung wahrnehmbar, erfahrbar ist“[2]. Realität hat also einen direkten Zusammenhang mit unserer Wahrnehmung, genauso wie Walter Fürst es im Eingangszitat anlauten lässt.

Betrachten wir als nächstes die Bedeutung von „virtuell“. Es bedeutet im allgemeinen: nicht echt, nicht in Wirklichkeit vorhanden, aber echt erscheinend. Bildungssprachlich meint es, entsprechend seiner Anlage als Möglichkeit vorhanden, die Möglichkeit zu etwas in sich begreifend.[3]

Verbinden wir diese beiden Bedeutungen, so erhalten wir für eine virtuelle Realität die Bedeutung einer Wirklichkeit, die nicht in Wirklichkeit vorhanden ist aber echt erscheint. Es handelt sich also um etwas Erfahrbares, Wahrnehmbares, dass nicht in der Wirklichkeit existiert, aber echt wirkt bzw. etwas, dass möglich wäre, wird zur Realität – also zu einer erfahrbaren, wahrnehmbaren Gegebenheit oder Erscheinung. Unsere Umwelt halten wir für real[4]. Wir tun dies, weil wir sie wahrnehmen und unseren Sinnen vertrauen, es sei denn wir gehen der philosophischen Idee nach, das diese Welt nur ein Traum sei.[5]

Wenn die wahrgenommene Umgebung nicht real ist, handelt es sich um eine *virtuelle* Umgebung, sofern sie real erscheint. Doch erst, wenn auch das, was in der virtuellen Umgebung passiert, ebenfalls für real gehalten werden kann, handelt es sich um eine virtuelle Realität. Doch wie kann erkannt werden, dass jemand eine Virtual Reality für real hält? Die Antwort ist: Wenn er sich natürlich verhält, also das gleiche Verhalten zeigt, wie er es auch zeigen würde, wenn das, was er wahrnimmt, Realität wäre.

2.1.1 Virtual Environment

> Jeder meint, dass seine Wirklichkeit die wirkliche Wirklichkeit ist.
>
> — Paul Watzlawick

Aber was sind alles virtuelle Umgebungen? Nach Ellis (1994) wird ein Virtual Environment durch das Zusammenspiel dreierlei Hardwarearten bewerkstelligt: Erstens Sensoren, die Körperbewegungen erfassen, zweitens Geräten, die Sinneseindrücke erzeugen und drittens einer Hardware, die beide vorigen Geräte so verknüpft, dass Eindrücke entstehen, wie es in der Realität der Fall wäre. Das bedeutet, dass eine Form von Interaktion vorhanden sein muss, mit der das Virtual Environment erforscht werden kann und dass die Interaktion konsistent ist. Wir interpretieren diese Definition von Virtual Environment als Spanne, so dass sowohl ein 2D Side-Scroller-Spiel[6] als

[2] *http://www.duden.de/rechtschreibung/Wirklichkeit*, Aufruf: 28.09.2017
[3] *http://www.duden.de/rechtschreibung/virtuell*, Aufruf 28.09.2017
[4] also zum Beispiel halte ich den Tisch an dem ich gerade Sitze, während ich dies schreibe für real
[5] vgl. Platons Höhlenmetapher
[6] Die Finger oder die Hand werden von Tastatur oder Maus erfasst, somit haben wir Sensoren, die Körperbewegungen erfassen. Das 2D-Display bietet den Sensoroutput, die Lautsprecher erzeugen eine Geräuschkulisse. Das

auch 3D-Bearbeitungsprogramme wie CAD oder Blender sowie Applikationen, die Head Mounted Displays und hochmoderne Sensoren verwenden, Virtual Environments sind. Am oberen Ende dieser Spanne wären StarTrek Holodecks bzw. das ultimative Display, wie es von Sutherland (1965) beschrieben wird:

> „The ultimate display would, of course, be a room within which the computer can control the existence of matter. A chair displayed in such a room would be good enough to sit in. Handcuffs displayed in such a room would be confining, and a bullet displayed in such a room would be fatal. With appropriate programming such a display could literally be the Wonderland into which Alice walked."

In diese Spanne ordnet sich unsere Anwendung, eine Software City, bestehend aus 3D Modellen, wunderbar ein. Sowohl bei der Steuerung mit Tastatur und Maus am Desktop als auch mit den Controllern und Kopfbewegungen bei Nutzung der HTC Vive, übertragen wir Körperbewegungen um ein konsistentes Virtual Environment zu bieten.

Damit ein Virtual Environment als Virtual Reality erkannt wird, muss der Nutzer ein Gefühl haben, tatsächlich dort sein. Wie oben erwähnt müssen die Reaktionen und Aktionen die mit dem Virtual Environment interagieren so sein als wäre diese real. Mit anderen Worten: Das Virtual Environment wird als wirklich (genug) akzeptiert. Folgendes Zitat beschreibt den Kerngedanken dieses Verständnissen von Virtual Reality:

> Im Zentrum der VR steht eine Erfahrung – die Erfahrung in einer virtuellen Welt oder an einem fremden Ort zu sein. (Rheingolf (1992))

An einem fremden Ort zu sein, ist ein Gefühl, dass auch in anderen Bereichen vorkommt. Nicht zuletzt gibt es im deutschen die Redewendung „Ich war in meinen Gedanken gerade wo anders.".

Auch bei Büchern und Filmen ist das Phänomen bekannt, dass jemand voll und ganz in eine Geschichte eintaucht und nach dem Lesen eine Zeit braucht, bis er wieder in der dieser Realität zurück ist. Aber auch durch Aufnahmen von Konzerten oder Naturklängen sich an einen anderen Ort versetzen zu lassen ist möglich. Bei letzterem werden oft die Augen geschlossen, damit man sich ganz auf die Musik einlassen kann.[7] Menschen können in Gedanken – aber auch in Sinneseindrücke – eintauchen und dies wird leichter, je mehr von der realen Welt abgeschottet wird. Dieser Beobachtung folgt der Begriff Immersion auch in Bezug auf Virtual Environments.

was man sieht ist ein langes Bild, das seitlich vor und zurück durch den Sichtbereich läuft, während bewegliche Objekte oben drüber gezeichnet werden. Somit erinnert es an ein Brettspiel, bei dem nie hingeguckt wird, während die Figuren umgestellt werden, sondern immer nur dann, wenn sie an der neuen Position stehen.

[7] nach eigener Erfahrung

2.1.2 Immersion

Bei einer Virtual Environment lässt sich Anhand des Systems, dass die Virtual Environment zugänglich macht, bemessen inwiefern ein Eintauchen möglich ist. Wir verwenden zur Bemessung die Methode von Slater (2009), die Immersion wie folgt definiert wird:

Immersion
Die Immersion eines Systems ergibt sich aus Summe der validen, natürlichen oder antrainierten Aktionen verstanden, mit denen die Virtual Environment wahrgenommen werden kann. Die Genauigkeit, bekannt als Fidelity, von Sinnesinformationen gibt hierbei die Grenzen der Aktionen an, deren Ziel eine genaue Untersuchung des jeweiligen Sinneseindrucks ist. Eine höhere Auflösung bei einer Anzeige geht, erhöht also die Immersion für den Sehsinn.

Definition 2.1 Immersion Definiton nach Slater (2009)

Den Blick bei der Arbeit an einem Desktop-Computer neben den Bildschirm zu richten (weder durch Kopfdrehen noch durch Augenbewegung) ist keine valide natürliche Aktion, um die Virtual Environment, die auf dem Bildschirm dargestellt wird, zu erfahren. Die Maus zu bewegen um in einem Spiel oder 3D-Bearbeitungsprogramm den Sichtbereich anzupassen, ist eine valide, antrainierte Aktion um die dargebotene Virtual Environment zu erforschen. Sich mit einem Head Mounted Display, das getracked wird, umzugucken, in dem der Kopf dreht wird oder man sich zur Seite beugt, ist eine valide natürliche Aktion. Den Kopf jedoch so nah an einen Gegenstand der Virtual Environment zu bewegen, dass erkennbar ist, dass die Auflösung des Displays des Head Mounted Display nicht erlaubt, den Gegenstand noch näher zu betrachten, ist nicht valide.

Beispiele 2.1 Beispiele für valide Aktionen zur Immersion

Da Immersion im Bereich Virtual Environment und Virtual Reality als Möglichkeit zum Eintauchen gesehen wird, wurde der Begriff Presence eingeführt. Dieser beschreibt den Zustand, wenn die Person ein natürliches Verhalten in der Virtual Environment zeigt. Wir nutzen die Definition von Steuer (1992) (siehe Abbildung 2.1), bei der zur besseren Unterscheidbarkeit die Presence in der virtuellen Welt als *Tele-Presence* bezeichnet wird. Von dieser Unterscheidung rücken wir ab und bezeichnen beides als Presence.

In der Abbildung 2.1 zerlegt Steuer die Einflüsse, von denen er ausgeht, dass sie zu Presence führen. Wir bilden diese Zerlegung auf die bereits erklärten Begriffe ab. Die Lebendigkeit (Vividness) bezieht sich auf den Gehalt der sensorischen Informationen, die eine Virtual Environment dem Nutzer bietet. Sie leitet sich aus zwei Attributen ab: Zum einen aus der Fidelity, welche in Steuers Zerlegung auf die Tiefe (Depth) der Informationen abgebildet werden muss und zum anderen

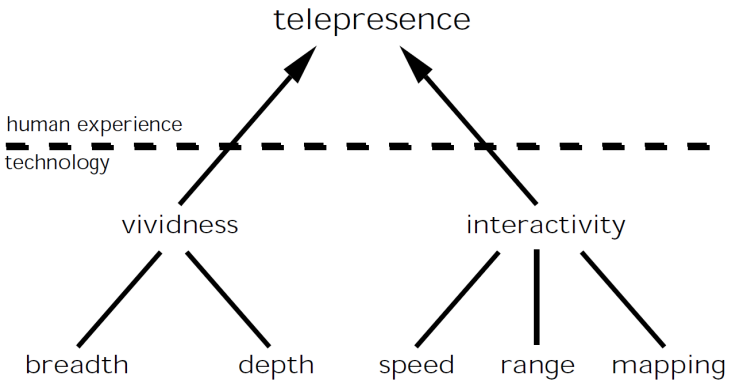


Abbildung 2.1 Definition technologischer Einflüsse auf Telepräsenz nach Steuer (1992)

aus der Immersion, die auf die Breite (Breadth) der generierten Informationen aus der Virtual Environment abbildet wird. Die Interaktivität trägt ebenfalls stark zur Presence bei. Ihr ist förderlich, wenn die Reaktionen des System flüssig sowie konsistent sind, und an natürlichen, die wir aus dem Alltag gewohnt sind, orientiert sind. Auch sollten die Aktionsmöglichkeiten, die das System manipulieren, an den natürlichen orientiert sein. Für weitere Informationen siehe Kapitel 2.4.

Betrachtet man Presence im Zusammenhang mit Software Cities, so stellt man fest, dass ein natürliches Verhalten gegenüber einer Stadt für die Anwendung gar nicht immer interessant ist. Im Gegensatz zu z.B. psychiatrischen Diagnosewerkzeugen oder Therapien[8], ist eine natürliche Reaktion auf Ereignisse nicht direkt für die Interaktion in der Software City interessant. Stattdessen interessiert vor allem, dass der Betrachter sich auf die Metapher der Software City einlässt und somit im vollem Umfang seiner Fähigkeiten eine räumliche Vorstellung dieser erwirbt.

2.1.3 Orientierung

Maier (1999) nimmt an, dass die wesentlichen Komponenten räumlicher Vorstellungskraft folgende sind:

Räumliche Wahrnehmung
: Identifizierung von Horizontale und Vertikale und Bestimmung der Lage des eigenen Körpers, sowie des Körperschemas

Räumliche Veranschaulichung
: gedankliches Verschieben Falten oder Schnitte von räumlichen Objekten benötigt wird;

[8] siehe Renaud, Trottier, Rouleau, Goyette, Saumur, Boukhalfi und Bouchard (2014) und Hasler, Spanlang und Slater (2017)

Vorstellungsfähigkeit von Rotationen
gedankliches Rotieren und Verdrehen von Körpern

Räumliche Beziehungen
das Erfassen räumlicher Konfigurationen von Körpern

Räumliche Orientierung
die richtige räumliche Einordnung der eigenen Person in eine räumliche Situation

Für das Erfassen von räumlichen Konfigurationen werden die Indikatoren für die Tiefenwahrnehmung verwendet. In Reichelt, Häussler, Fütterer und Leister (2010) werden diese erläutert und gezeigt, dass Licht und Schatten für Tiefenwahrnehmung dort eine wichtige Rolle spielen.

Wann immer eine Veränderung in der räumlichen Anordnung stattfindet, findet nach J. Rieser (1989) eine Neuorientierung[9] statt. Dieser Prozess ist im weitesten Sinne automatisch, wie unter anderem Wang (2004) untersucht hat. Bei genauerer Betrachtung fällt auf, dass keine Komponente des räumlichen Vorstellungsvermögens darauf aufbaut, dass Interaktionen besonders natürlich ablaufen.

Erfahrungsgemäß haben Menschen einen unterschiedlich gut ausgebildeten Orientierungssinn. Gemäß Prof. Münzer – zitiert in Kathrin Hollmer (2013) – ist Orientierung „eine Kompetenz, die sich erlernen und trainieren lässt“. Allerdings führt Münzer auch an, dass die heutzutage jederzeit verfügbaren Navigationsgeräte uns davon abhalten, „uns selbst zu lokalisieren und unsere Richtung festzustellen, Karten zu studieren, selbst Routen zu planen und benötigte Zeiten abzuschätzen. Lässt man sich führen, denkt man nicht mehr mit und lernt nichts über die Umgebung.“[10] Räumliche Orientierung ist – neben den eigenen Möglichkeiten in seiner Freizeit – ein Kompetenzbereich, der im Rahmen der allgemeinen Schulbildung vermittelt wird.[11]

Auch in Virtual Realities wird die räumliche Vorstellungskraft genutzt, dies zeigten Regian, Shebilske und Monk (1992), indem sie Probanden 'in' einer Virtual Reality Navigationsfähigkeiten erwerben ließen. Obwohl in der Realität für den Erwerb einer räumlichen Vorstellung alle Sinne genutzt werden, haben Riecke, Cunningham und Bülthoff (2007) zeigen können, dass der Sehsinn zur Orientierung und Neuorientierung ausreicht.

In unserem Versuch testen wir ebenfalls Navigationsfähigkeiten. Navigation lässt sich nach Bowman, Davis, Hodges und Badre (1999) in zwei Grundkomponenten aufteilen: Die Wegfindung und die Reise. Für die Wegfindung ist eine räumliche Vorstellung des Ist-Zustands sowie eine Einschätzung der Zustände, die zu jedem Zeitpunkt der tatsächlichen Bewegung vorgefunden werden, notwendig. Während der Reise findet – wie oben angesprochen – automatisch die Neuorientierung statt.

[9] z.B. durch einen Schritt nach vorne, eine Kopfdrehung, die Bewegung, Verdrillung etc. eines Körpers
[10] Kathrin Hollmer 2013.
[11] siehe dazu u. a. *https://www.uni-muenster.de/Geographiedidaktik/forschung/rok.html*

2.1.4 Aufteilung von Presence

In Slater (2009) wird vorgeschlagen, den Begriff Presence in Place Illusion und Plausibility Illusion aufzuspalten. Erstere bezieht sich danach auf Verhalten in der Virtual Reality, welches der natürlichen Erkundung einer (virtuellen) Umgebung entspricht. Letzteres bezieht auf die Natürlichkeit von Interaktionen zwischen Nutzer und Umgebung.[12]

Wir beziehen uns im weiteren auf die von Slater vorgeschlagene Trennung der Begriffe, denn in seiner Arbeit wurde aufgeführt, dass die Place Illusion immer auftritt, solange das System im Rahmen der durch Immersion gegebenen Grenzen erkundet wird. Solange also keine wahrnehmungsbezogenen Aktionen ausgeführt werden, die aufzeigen, dass die Umgebung nicht real ist, bliebt die Place Illusion intakt und kehrt sogar wieder, wenn nach einem Bruch in der Illusion wieder valide Aktionen ausgeführt werden. Dies ist für uns von besonderem Interesse, denn dies bedeutet, dass unsere Probanden während der Evaluation auf jeden Fall auf eine Place Illusion bei ihrer Navigation aufbauen werden.

In unserem Versuch vergleichen wir dieselbe Anwendung – einmal dargestellt an Desktop-Arbeitsplatz und einmal in einem Head Mounted Display-System. Uns interessiert dabei, ob das Head Mounted Display zu einer besseren räumlichen Vorstellung in einer Software City führt. Slater weist darauf hin, dass Place Illusion sich nicht direkt zwischen Systemen vergleichen lässt – besonders nicht, wenn sie in unterschiedlichen Äquivalenzklassen für Immersive Virtual Reality Systemen liegen. Diese definiert Slater durch die unterschiedlichen Grade der Immersion, bemessen an der Technik des Systems. Die oberste Äquivalenzklasse ist die Realität und alle Systeme (von denen es heute noch keine gibt), die sich in keinster Weise von der Realität unterscheiden lassen. Danach Stufen sie sich nach der gebotenen Immersion, gemessen an den validen Erkundungsaktionen, ab.

Chance, Gaunet, Beall und Loomis (1998) zeigten in ihrem Labyrinthszenario, dass die Nutzung eines Head Mounted Display mit kompletter Bewegungsfreiheit des Probanden, also der Immersion die auch bei der von uns eingesetzten HTC Vive gilt, die beste räumliche Vorstellung erwarben.

[12] Ein Beispiel für letztere ist, wenn automatisch das Lächeln einer digitalen Person zurück gelächelt wird.

2.1.5 Abgrenzung gegen ähnliche Begriffe [JG]

Die Idee eine dreidimensionale Metapher für ein mentales Modell zur Unterstützung der Softwareanalyse, wie es bei Software Cities der Fall ist, zu nutzen, hat auch einen kollaborativen Aspekt. Diese Kollaboration kann in einem Virtual Environment oder augmentierten Umwelt stattfinden. Wir haben uns dafür entschieden, die Metapher in unserer Arbeit in einem Virtual Environment darzustellen, die sich (fast) nicht mit physischen Welt überlappt[13]

Wir wollen den Begriff Virtual Reality klarer machen, indem wir diesen von folgenden Begriffen abgrenzen und uns der Taxonomie nach Milgram, Takemura, Utsumi und Kishino (1995) gemäß Abbildung 2.2 für diese Abgrenzung bedienen. Auch wenn einige der Begriffe ebenfalls das Wort *Reality* innehaben, handelt es sich dabei im Verständnis um Umwelten.

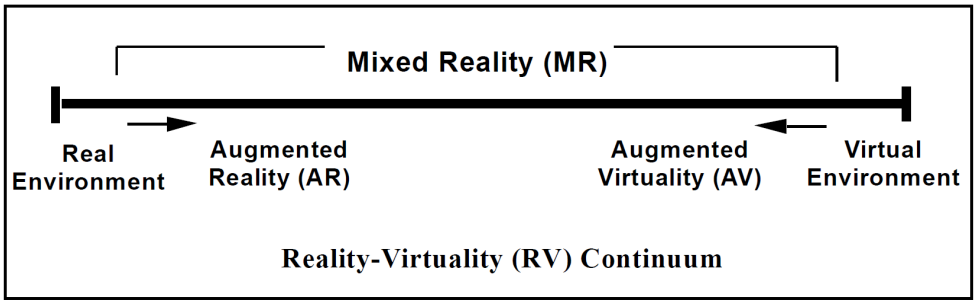


Abbildung 2.2 Reality-Virtuality (RV) Continuum
(Milgram, Takemura, Utsumi und Kishino (1995))

AR - Augmented Reality

„Augmentierte Realität ist eine (unmittelbare, interaktive und echtzeitfähige) Erweiterung der Wahrnehmung der realen Umgebung um virtuelle Inhalte (für beliebige Sinne), welche sich in ihrer Ausprägung und Anmutung soweit wie möglich an der Realität orientieren, so dass im Extremfall (so das gewollt ist) eine Unterscheidung zwischen realen und virtuellen (Sinnes-) Eindrücken nicht mehr möglich ist.“ schriebt Dörner, Broll, Grimm und Jung (2013, S. 246). Die reale Umwelt durch virtuelle Objekte zu erweitern, unterscheidet sich stark von dem Gedanken der VR. Es geht hier nicht um das Gefühl an einem anderen Ort zu sein, sondern um Zusatzinformationen im Sinne von Sinneseindrücken. Diese Grenze fängt an etwas zu verschwimmen, wenn z. B. ein leerer Raum mit virtuellen Möbeln gefüllt wird. Fühlt sich die Anwender dann noch als wäre er in dem selben Raum? Was, wenn die Tür, durch die er reingekommen ist, überlagert wird mit einer virtuellen Wand und in eine Wand eine virtuelle Tür eingebaut wird? Diese Beispiele zeigen, dass es durchaus möglich sein kann, dass die verschiedenen Ziele zu einem ähnlichen

[13] bis auf Controller und Raumbegrenzungensgitter der HTC Vive

Ergebnis führen können. In den meisten Augmented Reality-Anwendungen passiert dies jedoch nicht. Zu den bekannteren Augmented Reality-Technologien gehören z. B. Google Glasses, die z. B. für die Arbeitserleichterung im Paketdienst ausprobiert und eingesetzt wurde. Dort sollten Informationen darüber, wo ein Paket hingehört durch Scannen des Codes auf dem Paket direkt im Sichtfeld angezeigt werden. [14]

AV - Augmented Virtuality

Von Augmented Virtuality spricht man, wenn nicht die reale Umwelt mit virtuellen Objekten überlagert wird, sondern eine Virtual Environment mit realen (Milgram, Takemura, Utsumi und Kishino (1995)). Ein Beispiel für dieses bietet unsere Anwendung, denn zur Interaktion in unserer Software City nutzen wir die Vive Controller zusammen mit dem Head Mounted Display und zeigen diese Controller als Objekte in unserer Anwendung dem Nutzer, damit er ein besseres Feedback seiner Aktionen bekommt. Dies gibt der Tatsache, dass wir die Orientierung in einer Virtual Reality messen wollen keinen Abbruch, denn die realen Objekte tragen zwar zu Presence bei – immerhin ermöglichen sie eine indirekte Verortung der eigenen Gliedmaßen in der Virtual Environment – jedoch findet dadurch kein Abbruch der Place Illusion statt.

Auch wenn durch Motion Tracking z. B. die eigenen Hände in die Virtual Reality gebracht werden, handelt es sich um einen Fall von Augmented Virtuality

MR - Mixed Reality

Unter MR ist eine Verschmelzung von Virtual Environment und realer Umgebung zu verstehen. Sowohl Augmented Reality als auch AV sind Formen dieses Prinzips (Milgram, Takemura, Utsumi und Kishino (1995)).

2.1.6 Virtual Reality für Non-Game-Anwendungen [JG]

Virtual Reality blickt auf eine lange Geschichte zurück, in der im Gegensatz zu heute selten Spiele im Vordergrund standen. Wie Zimmermann in seinem Artikel beschreibt, hat z.B. die Autoindustrie eine lange Virtual Reality-Geschichte. Schon seit knapp 30 Jahren wird nach Zimmermann (2008) von der Autoindustrie Virtual Reality zu Demonstrationszwecken verwendet. In der Verhaltensforschung wird sie dafür eingesetzt mehr über den Menschen herauszufinden.

[14] *https://sven.meyer.works/google-glass-in-der-dhl-paketzustellung-mein-bachelor-projekt/*, urldate: 2017-09-25

Auch in unserer Anwendung geht es nicht um ein Spiel, sondern um ein Visualisierungswerkzeug. Virtual Reality ist ein Konzept und keine Technologie – das Konzept so zu reagieren, als wäre das Erlebte real. Durch die Immersionkann man sich intensiv auf das Gesehene ein lassen und mit den Sinnen möglichst viele Eindrücke aufzunehmen. Dies macht das Erlebte jedoch keineswegs zu einem Spiel. In einer Virtual Reality ließe sich z.B. ein Büro-Alltag mit dem dazu gehörigen Stress simulieren. Unser Visualisierungswerkzeug dient zur Graphvisualisierung und ist eine seriöse Anwendung.

Graphenvisualierung

Das Virtual Reality eine vielversprechende Ergänzung im Repertoire der Visualisierungswerkzeuge darstellt, zeigte unter anderen Ware und Franck (1994). In ihrer Arbeit wurden Graphen untersucht bzgl. der Fragestellung, ob zwei Knoten miteinander verbunden sind. In diesem Zusammenhang konnten sie herausfinden, dass Probanden mit der Virtual Reality[15] dreimal so viele Informationen zur selben Zeit erheben konnten wie auf einem 2D-Display.

2.1.7 Motion- und Cybersickness

Ein wichtiger Faktor beim Design von VR-Applikationen ist, dass der Nutzer – solange es sich nicht um eine VR der Äquivalenzklasse 0 handelt – inkonsistente Sinneseindrücke erfährt. Das bedeutet bspw., dass er sieht, dass er sich bewegt, dabei aber vollkommen stillsteht. Besonders auffällig ist es, wenn Bewegungen gebremst werden und keine Trägheit auftritt. Durch das Fahren mit Autos mag man es zwar gewohnt sein, allerdings treten dort ständig Trägheitserlebnisse auf. In diesen Fällen spricht Kennedy, Drexler und Kennedy (2010) von *Visual Induced Motion Sickness*(VIMS). Dabei geht der Begriff Motion Sickness weit zurück und beschreibt im allgemeinen jegliches durch von Bewegung induziertes Unwohlsein, bei dem die Bewegung zu einer unbewussten Verschiebung zwischen echten Zentrum der Gravitation und erwartetem führt.[16]

Motion Sickness ist jedoch nicht die einzige Inkonsistenz, die auftreten kann. Aus Untersuchungen mit Simulatoren ist bekannt, dass eine zu Virtual Reality-Erlebnissen laut Kennedy, Drexler und Kennedy (2010) vergleichbare Form von *Sickness* dort bei der Nutzung von Virtual Environments auftritt, nach McCauley und Sharkey (2005) Cyber Sickness genannt.

Weitere Inkonsistenzen, die zu Cyber Sickness führen, sind zum Beispiel Bild-Verzögerungen. Dort tritt dies auf, wenn nicht das Sichtbar ist, was zu erkennen sein sollte. Ein Beispiel ist, wenn mit einem Head Mounted Display der Kopf zu schnell zur Seite bewegt wird und am Rand

[15] hier: im Sinne eines Head Mounted Display mit vollständigem Tracking der Kopfbewegung
[16] siehe Chardonnet, Mirzaei und Mérienne (2015)

(schwarze) Flecken von nicht-gerenderten Grafikelementen auftauchen, oder wenn Standbilder auch nach einer Kopfbewegung noch bleiben, so als hätte man sich nicht bewegt.[17]

Unwohlsein kann die Orientierung erschweren, daher ist Cybersickness ein unerwünschter Faktor in unserer Anwendung ist. Wir werden setzen Strategien zur Vermeidung von Cybersickness ein.[18]

2.2 Visualisierung [MOR]

> Wenn ich es nicht visualisieren kann,
> kann ich es nicht verstehen.
> — Albert Einstein

> vi · su · al · ize
> 1. To form a mental image of
> 2. To make visible
> — *The American Heritage Dictionary of the English Language* (2017)

Der Begriff Visualisierung ist die Substantiierung des Verbs *visualisieren*, welches auf *visualis* (lat., „zum Sehen gehörig“) abstammt. Grundsätzlich soll *visualisieren* Unsichtbares sichtbar machen.

Unter einer Visualisierung wird entweder der Prozess des Visualisierens oder das Ergebnis dieses Prozesses verstanden. An einer Visualisierung beteiligt sind – wie bei einer Kommunikation – stets ein Sender (der Visualisierende) und ein Empfänger (der Betrachter). Eine Visualisierung führt der Visualisierende entweder für sich selbst[19] oder für jemand anderen durch[20]. Die Visualisierung eines Prozessergebnisses ist eine grafische, abstrahierende Darstellung von bspw. in Texten verborgenen Zusammenhänge und Erkenntnissen.[21] Visualisierungstechniken und -methoden wurden erdacht, die – basierend auf den Grundlagen verhaltens- und wahrnehmungspsychologischer Erkenntnisse – helfen sollen, das Ziel der Visualisierung zu erreichen. Spätestens, wenn Sender und Empfänger nicht identisch sind, muss sich der Sender auch Gedanken über den Kontext

[17] siehe Hettinger und Riccio (1992)
[18] siehe Kapitel 2.5.2
[19] bspw. „Ich visualisiere, am Meer zu sein.“, auch *Imagination*, *Vorstellung* – moderner und werbewirksam – u. a. auch als *mentales Training* tituliert
[20] bspw. „Der Meister visualisierte ihm den Weg ins Licht“, kann auch als Präsentation oder Manipulation verstanden werden
[21] bspw. „Sie visualisiert die Unternehmensumsätze der letzten 20 Jahre in einem Diagramm.“ bspw. in Form einer Zeichnung oder eines Diagramms

Abbildung 2.3 "Immaterials: Light painting WiFi"
Durch eine Langzeitaufnahme der Stärke eine WiFi-Signals an bestimmten Positionen machten im Video *https://vimeo.com/20412632* Forscher die Verteilung eines unsichtbaren WiFi-Signals sichtbar.

und eine Anleitung machen.[22] Diehl (2007, S. 149) hebt als vorrangiges Ziel der Visualisierung die Übermittlung von Informationen hervor.

Unsere Arbeit beschäftigt sich mit der Visualisierung von Software. Softwarevisualisierung wird von der Literatur als Teil des Themengebiets Informationsvisualisierung verstanden, welches wiederum ein Unterbereich der Visualisierung ist. Über die Beteiligten der Softwarevisualisierung sowie deren Ziele schreiben wir in Kapitel 2.2.4.

2.2.1 Daten- und Informationsvisualisierung [MOR]

The purpose of visualization is insight, not pictures.
— Card, Mackinlay und Shneiderman 1999, S. 6

Was ist gute Datenvisualisierung?
Gute Datenvisualisierung ist ein Darstellung von Daten,
die es ermöglicht, Dinge zu sehen, die verborgen blieben,
würden nur die nackten Quelldaten betrachtet.
— Yau und Hesse-Hujber (2014, S. 11)

[22] vgl. Yau und Hesse-Hujber (2014, S. 67)

Die ältesten noch heute erhaltenen *Informationsvisualisierungen*, die grafisch dem entsprechen, was wir heute darunter verstehen, wurden nach heutigem Wissen 1786 von Playfair (1801) veröffentlicht.[23] Playfair veröffentlichte offenbar als erster eine ganze Reihe von noch heute genutzten Diagrammtypen wie Balken-, Linien- und später Kreisdiagramme, teilweise mit Indikatorbalken. Von Charles Joseph Minard ist aus dem Jahr 1812 noch eine Karte zu Napoleons Russland-Feldzug erhalten, aus dem Jahre 1854 existiert eine Karte über eine Häufung von Cholera-Fällen von Dr. John Snow und 1858 präsentierte Florence Nightingale die Erfolge Ihrer Arbeit bzgl. der Sterblichkeit in Krankenhäusern mittels aufwändig von Hand erstellten Diagrammen (siehe Abbildung 2.4).

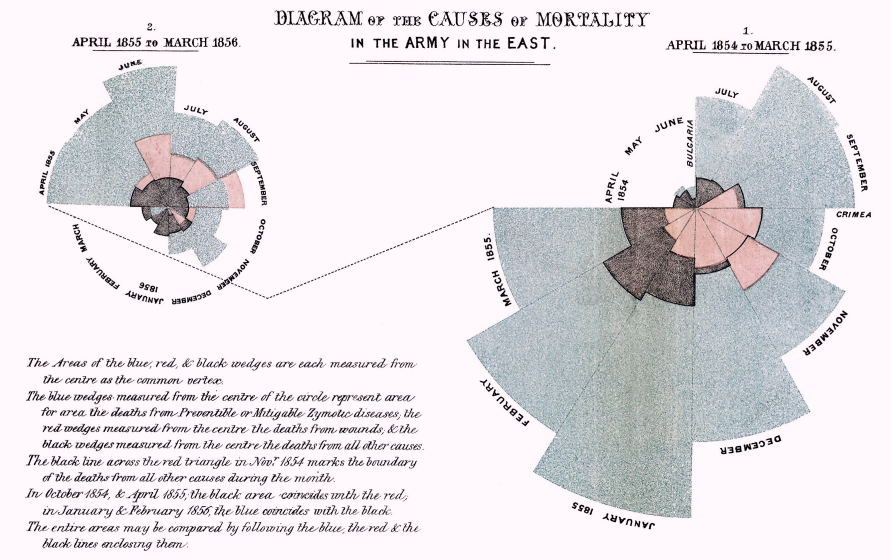


Abbildung 2.4 Florence Nightingales Diagramm über die Senkung der Sterblichkeitsrate in den Krankenhäusern von Skutari (1858)

Das Besondere an diesen ersten Informationsvisualisierungen ist die Verbindung zweier bis dahin ideologisch getrennter Bereiche – Wissenschaft und Werbung – in speziellen, zuvor noch nie präsentierten, jugendstilartigen Visualisierungsformen. Sie repräsentieren die positivistische Erkenntnis, dass Wissenschaft sich präsentieren muss, um Entscheidungsträger von der Handlungsnotwendigkeit zu überzeugen.[24]

Der Begriff Informationsvisualisierung selbst ist noch relativ jung: 1974 wurde die erste internationale SIGGRAPH-Konferenz über Computergraphiken und interaktive Techniken einberufen, auf der sich erstmals internationale Wissenschaftler mit Themen der Computergrafik auseinandersetzten und seitdem untrennbar mit der Computertechnik verbunden. Mit der steigenden Leistungsfähigkeit der Rechner entwickelten sich Darstellungskonzepte und Interaktionsmecha-

[23] Andere Informationsvisualisierung sind weit älter; in Rigamonti (2015) werden sogar Diagramme aus dem 10 Jhrd angeführt.
[24] vgl. Zwecker (2010, S. 215)

nismen[25], die später teilweise auch im Bereich der Softwarevisualisierung eingesetzt wurden.

Nach heutigen Definitionen hat die Informationsvisualisierung zum Ziel, sprachliche Informationen zum Zwecke der besseren Verständlichkeit in visuelle zu transformieren. Die Transformation erfolgt dabei auf einer höheren Abstraktionsebene, die Details ausblendet. Bei der Transformation sind gestalterische Aspekte zu berücksichtigen. McCormick, DeFanti und Brown (1987, S. 3) formulieren dies – damals noch unter dem Begriff Visualisierung – wie folgt:

Definition für Informationsvisualisierung
Visualization is a method of computing. It transforms the symbolic into the geometric, enabling researchers to observe their simulations and computations.
Visualization offers a method for seeing the unseen. It enriches the process of scientific discovery and fosters profound and unexpected insights. In many fields it is already revolutionizing the way scientists do science.

Definition 2.2 Definition Informationsvisualisierung nach McCormick, DeFanti und Brown (1987, S. 3), damals noch unter dem Begriff Visualisierung

Wong und Bergeron (1997) differenzieren in ihrem Überblick verschiedene Entwicklungsphasen, die die Informationsvisualisierung zwischen 1967 und 1997 durchlaufen hat. Erst in der letzten Phase etablierte sich der Begriff „Visualisierung“ für das Themengebiet der grafischen Darstellung[26] und die Informationsvisualisierung als eigenständiger Forschungsbereich durch die beiden Konferenzen *1. EUROGRAPHICS Workshop on Visualization in Scientific Computing* und die *1. Annual IEEE Visualization Conference* im Jahr 1990.

2.2.2 Verhaltens- und wahrnehmungspsychologischer Grundlagen von Visualisierungen

> Alle großartigen Designer von Diagrammen wecken Gefühle bei mir:
> Angst, Verwunderung, Überraschung, Freude … irgendetwas.
> Sogar Ruhe und Gelassenheit […]
> — Amanda Cox nach Yau und Hesse-Hujber (2014)

Bei der Gestaltung von Visualisierungen sind eine Vielzahl menschlicher wahrnehmungspsychologischer Prinzipien zu berücksichtigen. Die wohl bekanntesten sind die sogenannten *Gestalt-Gesetze* nach Wertheimer (1923). Sie definieren wahrnehmungspsychologische Regeln zur mensch-

[25] vorangetrieben in den Anfangsjahren insbesondere durch Tukey (1977)
[26] offenbar durch die Arbeit von McCormick, DeFanti und Brown (1987)

lichen Wahrnehmung von Formen im Verbund. 1967[27] veröffentliche Bertin eine Arbeit über *graphische Variablen*, die viele der bis dahin intuitiv angewandten Zuordnungen von Werten zu Attributen zweidimensionaler Objekte bei der Gestaltung von bspw. Diagrammen zusammenfasste (siehe Abbildung 2.5). Bertin arbeitete heraus, welche der visuellen Variablen zu welcher der Eigenschaftsgruppen *Auswahl*[28], *Reihenfolge*[29] und/oder *Menge*[30] zuzuordnen ist.

Die Bertin'schen Semiologien sind gleichwohl ein Spiegel ihrer Zeit. So wundert es auch nicht, dass sie später von verschiedenen Autoren um Aspekte wie Länge, Volumen, Farbton, Sättigung, Winkel, Verbindung, Enthaltung, Blinken und Bewegung konkretisiert wurde; und die Erweiterungen dauern an.[31]

2.2.3 3D-Visualisierungen

> Graphics is the visual means of resolving logical problems.
>
> — Jacques Bertin, 1977

Während die bisher vorgestellten Grundlagen sich zumeist auf 2D-Visualisierungen beziehen, sind natürlich auch Visualisierungen in der dritten Dimension möglich, in die Teile der Grundlagen übertragen werden können, andere noch gefunden werden müssen. Heute unterscheiden wir zwischen Pseudo-3D- oder 2,5D- und echten 3D-Darstellungen. Unter Pseudo-3D ist dabei die grafische Projektion von 3D-Objekten auf eine 2D-Oberfläche zu verstehen.[32] Darüber hinaus existieren in der Display-Technologie *3D-Displays* (oder neuer: *Stereodisplays*) Monitore oder Fernseher, die durch verschiedenste Technologien einen räumlichen Eindruck beim Betrachter hervorrufen können. In unserer Arbeit vergleichen wir Desktop-Darstellungen mit einem Head Mounted Display-System. Bei unserem Desktop-System handelt es sich um einen handelsüblichen Monitor[33], wir setzen kein Stereodisplay ein.

1994 haben Ware und Franck zeitgleich mit Vion-Dury, Santana und Bull (1994) gezeigt, dass 3D-Darstellungen komplexer, abstrakter Daten bei bestimmten Aufgabenstellungen 2D-Darstellungen ab eine gewissen Datenmenge überlegen sind. Später zeigten Ware und Franck (1996), dass in einer stereoskopisches 3D-Darstellung abstrakter, vernetzter Informationen um den Faktor 3 mehr Informationen vom Betrachter wahrgenommen werden können als in einer

[27] Im Literaturverzeichnis: deutsche Ausgabe von 1974: Bertin (1974)
[28] sélection; spontan vom menschlichen Betrachter zu Gruppen zusammengefasst; Visualisierung nominaler Daten
[29] ordre; spontan vom menschlichen Betrachter als Ordnung wahrgenommen; Visualisierung ordinaler Daten
[30] quantité, spontan vom menschlichen Betrachter mit einem Wert assoziiert; ordinale und quantitative Daten
[31] siehe zu Erweiterungen u. a. Albertz (1997), Vlahos (1965) oder Sieber (1996)
[32] In der Geowissenschaft geht die Definition dahingehend weiter, dass in der Vertikalen nur ein Datensatz vorliegt, so das bspw. Straßen unter Brücken nicht dargestellt werden können.
[33] Technische Daten, siehe Tabelle 4.5

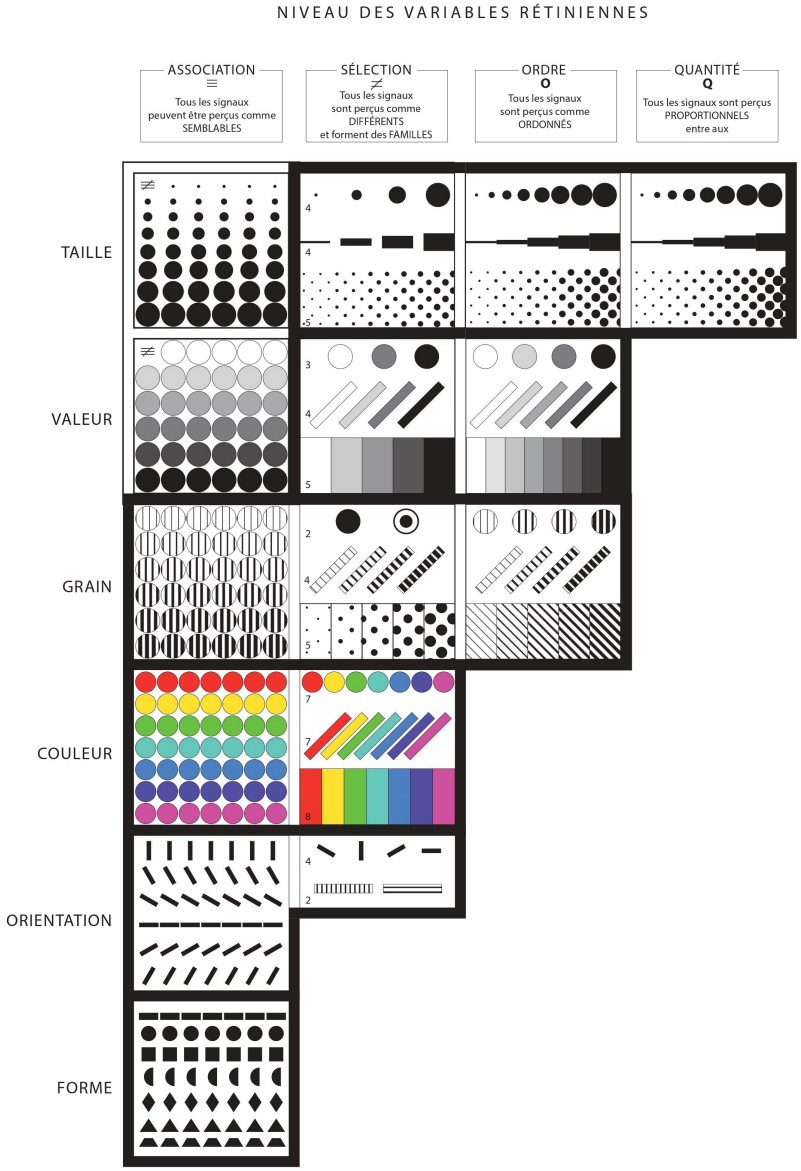


Abbildung 2.5 Semiologie nach Bertin (1974)
Hier dargestellt: Position (x,y), Textur, Fläche, Größe, Helligkeit, Neigung / Orientierung, Form, Gestalt und Farbe bzgl. Auswahl, Reihenfolge und Verhältnis

inhaltsgleichen 2D-Darstellung. Gleichzeitig merkten sie an, dass die Erzeugung der 3D-Daten vergleichsweise aufwändig wäre und die Produktion von 3D-Darstellungen – der offenbaren Vorteile zum Trotz – nicht rechtfertigen würde. Seitdem sind rund 20 Jahre vergangen.

Heute stellt die Erzeugung vergleichsweise komplexer dreidimensionaler Daten keine Schwierigkeit mehr dar – abgesehen von langen Einarbeitungszeiten in die entsprechende Software. Eine Reihe Grafikprogrammen ermöglichen die Gestaltung von 3D-Grafiken und -Animationen, Grafik-Bibliotheken und Grafik-Frameworks ermöglichen eine hardware-nahe Programmierung der Grafikkarten, die häufig leistungsfähiger als die Rechner sind, in denen sie verbaut sind. Induziert durch den Erfolg des Marktes der (Virtual Reality-)Computerspiele entstanden und entstehen zeitgleich sogenannte Game Engines[34], die die Erstellung von grafiklastigen 3D-Spielen vereinfachen, indem sie die Möglichkeiten (vieler) Grafikbibliotheken auf ein höheren Abstraktionniveau heben. Endkundentaugliche Virtual Reality- und -Augmented Reality-Systeme mit Head Mounted Displays drängen seit einigen Jahren auf den Markt, die stereoskopische, bewegte, interaktive Sichten in virtuelle Welten ermöglichen[35] – Welten die nicht selten Teil von Virtual Reality-Spielen sind.

2.2.4 Software-Visualisierung [MOR]

> Imagination or visualization, and in particular the use of diagrams, has
> a crucial part to play in scientific investigation.
> — René Descartes, 1637

Eine der ältesten heute noch existenten Softwarevisualisierungen sind offenbar die Aufzeichnungen von Lady Ada Lovelace aus dem Jahre 1843 (das Programm *Note G* ist in Abbildung 2.6 zu dargestellt), in welchen sie Algorithmen in tabellarischer Form dokumentierte. Abgesehen von der tabellarischen Form und der Visualisierung von Algorithmen entstanden mit der Zeit weitere Visualisierungsvarianten, die allerdings lange noch nicht als Softwarevisualisierungen bezeichnet wurden. Eine seit den frühen Tagen des Computerzeitalters eingesetzte Form ist das aus anderen Disziplinen entliehene *Ablaufdiagramm*[36], welches lange Zeit als grafische Notationen genutzt wurde und deren Nachkommen in Deutschland schließlich in der DIN 66001[37] genormt wurden: der Programmablaufplan. Selbstverständlich entstanden eine Vielzahl von weiteren Visualisierungsmethoden und -Notationen, auf die wir im Großen und Ganzen nicht weiter eingehen wol-

[34] siehe Kapitel 2.5
[35] siehe dazu Kapitel 2.1
[36] Unter *https://commons.wikimedia.org/w/index.php?title=Flow_chart&oldid=256796949* finden sich eine Vielzahl von Ablaufdiagrammen, die belegen, in welchen Bereich diese Form der Visualisierung bereits Jahrzehnte zuvor eingesetzt wurde.
[37] siehe Rekhter und Li (1983). Eine Beschreibung dieser DIN findet sich u. a. in Hering (1984)

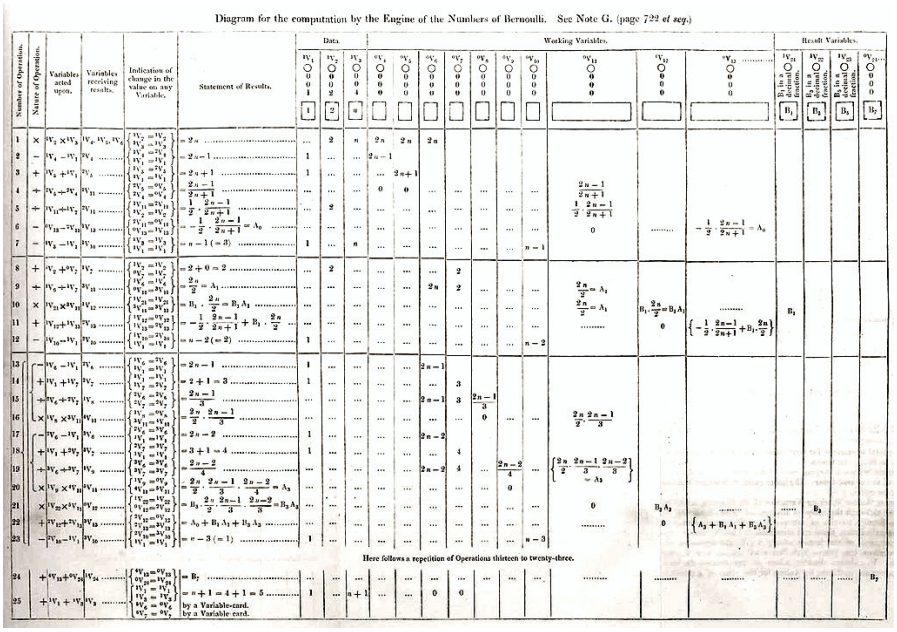


Abbildung 2.6 Ada Lovelace: Note G
frühe überlieferte Visualisierungen eines Algorithmus, hier: der Berechnung der Bernoulli-Zahlen

len. Insbesondere in den 1990er Jahren wurden im Rahmen der sogenannten „Software-Krise“ eine beachtliche Anzahl verschiedener grafischer Notationen vorgestellt, aus denen schließlich der marketingtechnische raffiniert mit Unified Modeling Language bezeichnete Ansatz der sogenannten *drei Amigos* Grady Booch, James Rumbough und Ivar Jacobsen hervorging, welcher inzwischen zum Standard erhoben und in den meisten Büchern zum Thema Softaretechnik als einzige Visualisierungsvariante vorgestellt wird.[38]

Zu der Frage, was denn nun eigentlich Softwarevisualisierung sei, gibt es eine Vielzahl unterschiedlich ausgerichteter und intensiver Definitionen. So beschreiben Domingue, Price und Eisenstadt (1994) Softwarevisualisierung über den Begriff *Sofwarevisualisierungsystem* als Möglichkeit, das Verständnis eines Programmierer (s)einer Arbeit zu vertiefen[39] und konkretisieren ihre Begriffsbestimmung in 1992 über die Tätigkeit *Softwarevisualisierung* als ein Misch-Handwerk aus Design, Typographie, Animation, Filmatographie und Mensch-Maschine-Interaktionen mit dem Ziel, das Verstehen und effektive Nutzen von Computer Programmen durch den Menschen zu fördern.[40]. Neben dieser Definition gibt es noch weitere und auch spezialisierte Definitionen, die bestimmte Aspekte des Software-Entwicklungsprozesses abdecken, bspw. die Verfolgung des Ablaufs von Algorithmen, rein statische Ansichten oder lediglich andere Begriffe wie „program

[38] Dies mag für die Ausbildung von Vorteil sein, jedoch nicht für die Pflege einer kreativen Vielfalt.
[39] siehe Definition 2.3
[40] siehe Definition 2.4

visualization“[41] nutzen.

[Software visualization systems are] systems that use visual (and other) media to enhance one programmer's understanding of another's work (or his own).

Definition 2.3 Definition Informationsvisualisierungssystem nach Domingue, Price und Eisenstadt (1992, S. 53)

[Software visualization describes] work that uses the crafts of typography, graphic design, animation, cinematography and modem humancomputer interaction technology to facilitate the human understanding and effective use of computer programs.

Definition 2.4 Definition Informationsvisualisierung nach Eisenstadt, Price und Domingue (1992, S. 53)

[Software visualization is] the visualization of artifacts related to software and its developing process.

Definition 2.5 Definition Informationsvisualisierung nach Diehl (2007)

UML 3D

Für die Unified Modeling Language wurde 2001 von Dwyer untersucht, inwieweit die Übertragung der bekannten und standardisierten Unified Modeling Language in eine Desktop-Umgebung Vorteile gegenüber der 2D-Darstellung hat. Grundsätzlich sind danach 3D-Visualisierungen flexibler in der Anordnung und Darstellung, was wohl einfach daran liegen mag, dass die dritte Dimension einen Freiheitsgrad mehr als die zweite hat. Im Einzelnen führten Dwyer folgende Aspekte an:

- In 2D-Darstellungen ist die **Darstellungskomplexität ggfs. exponentiell** bei genügend großen Systemen. In einer 3D-Darstellung würde die Daten auch den Raumachse nutzen können, dadurch bleibt die Komplexität länger gering.
- in 2D-Darstellungen muss mehr **Zeit für Planarisierung und Vermeidung von Überschneidungen** eines graphbasierten Modells[42] eingeplant werden; der Vorgang ist hier wesentlich aufwändiger als in 3D-Darstellungen.[43]
- Die Darstellung **hierarchischer oder geschachtelter Strukturen** (bspw, Paket-, Modul- oder Vererbungsstrukturen) nimmt in 2D- mehr Platz ein als in 3D-Visualisierung.

[41] Myers 1986.
[42] zum Thema *Software als Graph* siehe Kapitel 2.3.1
[43] zum Planarisierungsproblem siehe auch Buchheim, Chimani, Gutwenger, Jünger und Mutzel (2012); zur Minimierung der Kanten und Kantenbündelung siehe Kapitel 2.2.6; angemerkt soll sein, dass sich durch die Nutzung der dritten Dimension das Problem lediglich um eine Dimension in eine andere Größenklassen verschiebt und andere Schwierigkeiten impliziert

Trotz der offenbaren Bedeutung der Softwarevisualisierung beschränkt sich in Software-Engineering-Fachbüchern das Thema Softwarevisualisierung lediglich auf die UML.[44] Dies könnte an der von Diehl (2007) angeführten fehlenden Standardisierung liegen. Es finden sich im Internet eine Reihe von Hinweisen auf realisierte, zumeist universitäre Softwarevisualisierungssystemene.[45] sind zumeist aus universitären Projekten entsprungen und befinden sich zumeist in einem prototypischen Zustand. Auch für weit verbreitete Managementsystem für Software wie SonarQube gibt es Visualisierungserweiterungen, die allerdings wenig bekannt sind und deren möglicher Einsatzzweck sich Entwicklern, mit denen wir gesprochen haben, nicht erschließt. Andere Systeme sind direkt in Entwicklungswerkzeuge integriert, verfügen aber häufig für Projekte, die über das minimale Maß hinausgehen, über zu wenige Filter und Konfigurationsmöglichkeiten.[46]

Gleichwohl Koschke (2003) die Frage nach der Bedeutung von Software-Visualisierung durch eine Befragung von Wissenschaftlern wissenschaftlich bestätigen konnte, scheint diese Bedeutung noch nicht in der Praxis angelangt zu sein.[47]

Nachfolgend wollen wir eine Variante der Software-Visualisierung kurz darstellen, die wir im weiteren Verlauf unserer Arbeit eingesetzt haben.

Treemap

TreeMaps sind eine Variante der Visualisierung, die zur Softwarevisualisierung gezählt wird, da sie aus diesem Bereich entstanden ist. TreeMaps wurden 1992 von Shneiderman erstmalig vorgestellt. Sie sind eine Möglichkeit, einen Bereich gemäß einer hierarchischen Graph-Struktur zu zerteilen und ermöglichen Visualisierungen, die die Verteilung von hierarchischen Daten in einem begrenzten Bereich verdeutlichen. Die bekanntesten TreeMaps sind im 2D-Bereich zu finden, in Abbildung 2.7 haben wir zwei Originalabbildungen von Shneiderman übernommen und angepasst, an denen die Zuordnung zwischen Graph und Fläche verdeutlicht wird.

Neben der ursprünglichen 2D-Darstellung von TreeMaps gibt es auch die Möglichkeit, die Visualisierung um weitere Parameter wie bspw. die Höhe zu erweitern.[48]

[44] siehe u. a. Braude und Bernstein (2011), Lichter (2007)
[45] bspw. Dwyer und Eckersley (2002) (auch zu finden auf Sourceforge *https://sourceforge.net/projects/wilma/?source=directory*, letztes Update: 2013), Projekt WALRUS (*http://www.caida.org/tools/visualization/walrus/*) oder Projekt GALICIA (*http://www-labs.iro.umontreal.ca/~galicia/visualization.html*)
[46] Wir haben bspw. interessehalber versucht, im von der Unreal Engine vorgegebenen Programmierwerkzeug Microsoft Visual Studio das Visualisierungswerkzeug unser Projekt visualisieren zu lassen. Da zu dem Projekt auch alle Dateien der Engine gehörten handeltes es sich um ca 35000 Klassen, die dargestellt werden müssen. Visual Studio hat aufgegeben, bevor die Visualisierung dargestellt wurde.
[47] So ist dann wohl auch die Aussage des Nutzers „Christian“ auf *https://www.phpgangsta.de/software-grafisch-darstellen-mit-code_swarm* zur Beschreibung, wie code-swarm für die Visualisierung von Software-Projekten genutzt werden kann, zu erklären: „wow inspirierend. kann ich mir gut als screensaver vorstellen. :-)“ (Smiley im Original)
[48] siehe Abbildung 3.3a

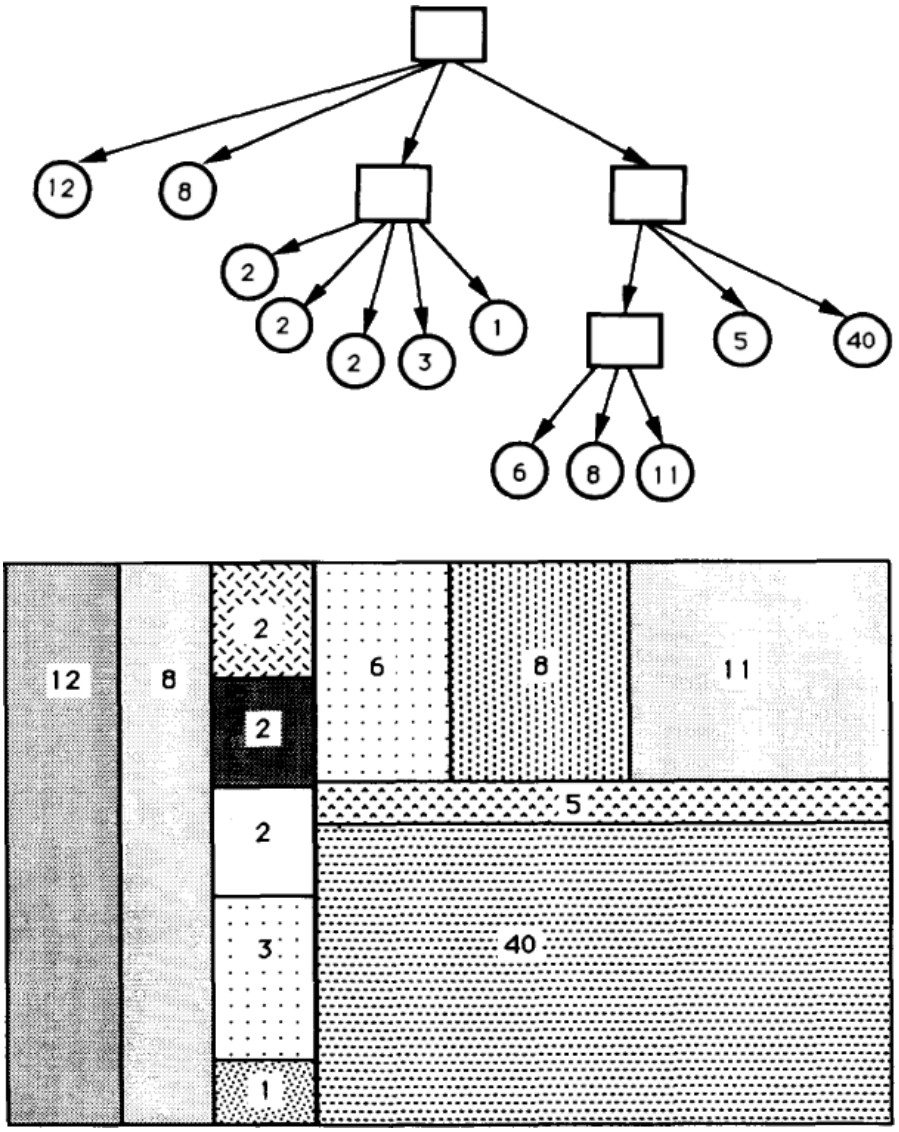


Abbildung 2.7 Treemap nach Shneiderman, leicht angepasst

2.2.5 Software-Cities [JG]

Eine Software als Stadt zu beschreiben ist eine Form von Softwarevisualisierung[49], bei der für die Visualisierung abstrakte Grundelemente einer Stadt verwendet werden, die vom Betrachter jedoch immer noch als Stadt erkannt wird.

Die Darstellung als Stadt ist dabei eine Metapher, durch deren Verwendung, gewohnte Kenntnisse aus dem Wissensbereich um Städte des Nutzers verwendet werden sollen. Ein Beispiel wäre z.B. das Häuser an Straßen stehen. Es also einen Zusammenhang zwischen der Position der Häuser gibt in dem Straßen eine Rolle spielen. Dies muss dann dem Nutzer nicht noch extra erklärt werden. Das Ziel es also den abgehobenen Begriff des Software-Graphen durch eine anschaulicheres, sprachliches Bild zu ersetzen[50], was die Wissensdomänen verknüpfen soll.

Abstrakt lässt sich eine Stadt als Sammlung von Blöcken die an Linien stehen, beschreiben. Bei einer solch abstrakten Darstellung wäre eine Metapher aus dem Spielen mit Bauklötzen vielleicht passender.

Nutzen der Stadtmetapher

Doch warum die Stadt- oder Bauklotzmetapher? Wie Steinbrückner (2013, p.2) in seiner Einleitung aufzeigt, eignet sich die Metapher der Software City aus zwei Gründen zur Visualisierung von Software-Graphen. Der erste ist ihre hohe Ausdrucksstärke, durch die große Vielfalt und Details die Städten zu Grunde liegt, gibt es viele Möglichkeiten sie auf Aspekte von Software zu übertragen und damit viele Informationen darzustellen. Zweitens die Effizienz der Darstellung, Steinbrückner betont, dass die Informationen in der Stadt Metapher schnell und in großer Zahl aufgenommen werden.

Jedoch ist die Nutzung von Metaphern in der Visualisierung ein zwei schneidiges Schwert. Es kann leicht passieren, dass nicht gewollte Assoziationen entstehen. Abbildung 2.8 zeigt ein Beispiel in dem die Methoden als Blöcke zu je 4 innerhalb auf der Grundfläche ihrer Klasse übereinander gestapelt werden. Dadurch entsteht der Eindruck von Stockwerken. Die Stockwerke sagen in diesem Fall jedoch nichts über den Zusammenhang zwischen den jeweiligen Methoden aus, außer dass diese in der Eingabe für den Algorithmus hinter einander lagen. Aus realen Städten ist jedoch zu erwarten, dass ein höherer semantischer Zusammenhang zwischen Stockwerken herrscht. Unbeabsichtigterweise fragt sich ein Betrachter also warum die Stockwerke weiter oben eingefärbt sind. Dabei muss die Höhe keine direkte Bedeutung haben.

[49] siehe Kapitel 2.2.4

[50] Eine Metapher ist ein sprachliches Bild, durch das der eigentliche Ausdruck durch etwas ersetzt wird, was anschaulicher, deutlicher oder sprachlich reicher ist, bspw. 'Baumkrone' für 'Spitze des Baumes', zum Teil auch zum Schliessen semantischer Lücken (Flaschenhals), das Prinzip der linguistischen Similarity wird genutzt (Wikipedia)

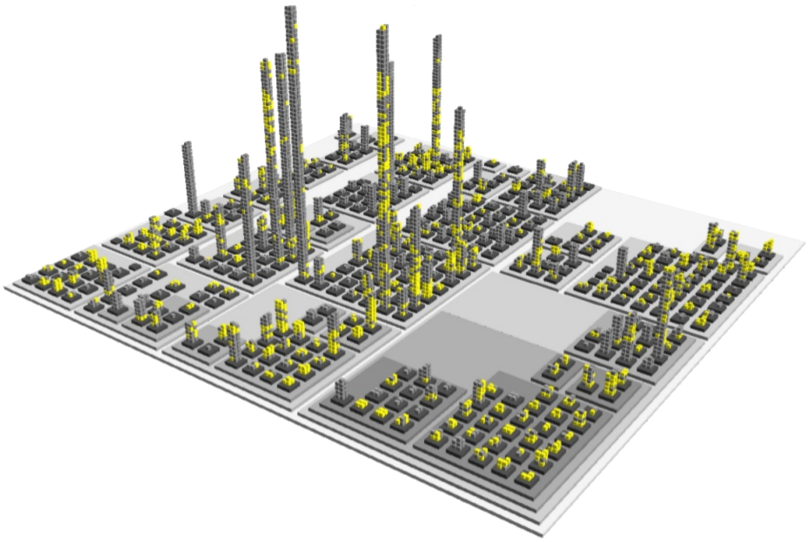


Abbildung 2.8 Unbeabsichtigte Mehrdeutigkeit bei einer Software City
Quelle: Wettel und Lanza (2008b)

Gründe eine Metapher zu nutzen um mehr Informationen zur selben Zeit abrufbar zumachen sind unter anderem, die verschiedenen Interessen der Stakeholder die an der Softwareentwicklung beteiligt sind. Ohne Metapher müssten sonst alle Stakeholder mühsam an eine komplexe Visualisierung gewöhnt werden, damit sie trotz unterschiedlicher Foki die selbe Visualisierung nutzen können und mithilfe dieser kommuniziert werden kann. Die Metapher ersetzt keines Wegs das Training, allerdings sollte sie ähnlich wie beim NUI(siehe Kapitel 2.4) Ansatz die Trainingszeit verkürzen.

Arten von Software Cities

Um also mehrere Stakeholder an einen gemeinsamen Tisch zu bringen, metaphorisch gesprochen, werden Metaphern genutzt. Eine Metapher lässt sich allerdings auf viele Arten anwenden. Dies gilt auch für die Stadtmetapher.

Einer der frühen Ansätze ist, der in diese Richtung geht ist die „Information Pyramid“ aus Andrews (2002). Dort werden Blöcke auf über einander gestapelten immer kleiner werdenden Ebenen angeordnet. Dabei stehen die unterliegenden Ebenen jeweils für Elternknoten in einer eindeutigen Hierarchie. Dieser Ansatz stellt nicht direkt eine Stadt da, ist aber im Kerngedanken immer wieder Teil von anderen Ansätzen.

Ein anderer, deutlich stadtähnlicherer war das *CyberNet Project* von Santos, Gros, Abel, Loisel,

Trichaud und Paris (2000).[51]. Eine Kernaufgabe dieser Arbeit war das Mapping von Informationen auf graphische dreidimensionale Objekte.

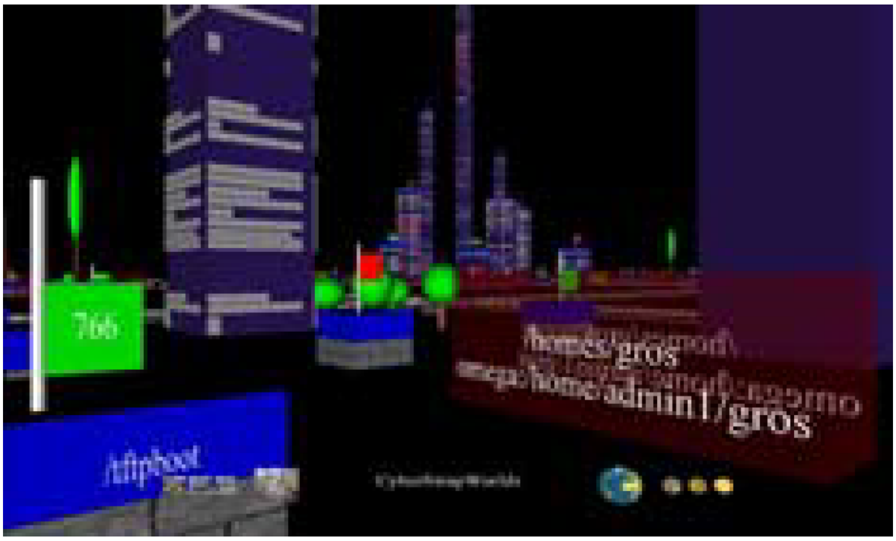


Abbildung 2.9 Informationsmapping auf 3D Objekte einer Stadtmetapher Software City
Quelle: Santos, Gros, Abel, Loisel, Trichaud und Paris (2000)

Die Stakeholder zu einer gemeinsamen Visualisierung zu führen war auch das Ziel in den Arbeiten Panas, Epperly, Quinlan, Saebjornsen und Vuduc (2007) und Panas, Berrigan und Grundy (2003) in diesen wird insbesondere großer Wert auf die mehreren Ebenen und verschiedene Aufgabenfelder gelegt. In Panas, Epperly, Quinlan, Saebjornsen und Vuduc (2007) gehen sie auf die Fragen der Verbindungen zwischen Elementen des Software-Graphen ein, die nicht zur Hierarchie gehören und in Panas, Berrigan und Grundy (2003) legen sie speziellen wert auf die Veranschaulichung der Kostenfaktoren um Personal zu sinnvoll wie möglich einsetzen zu können. Zu sehen sind diese beiden Varianten in Abbildung 2.10.

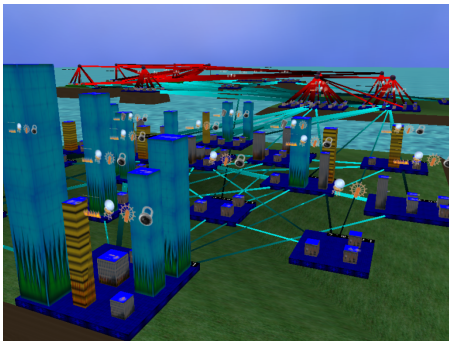
Der Ansatz der CodeCity von Wettel und Lanza (2008b) setzt seinen Fokus auf die Darstellung von sogenannten Disharmonien im Code auf Klassen und Methodenebenen. Abbildung 2.8 zeigt Disharmonien auf Methodenebene. Eine andere Einsatzmöglichkeit für die CodeCity beschrieb Wettel und Lanza (2008a). In diesem liegt der Fokus auf Evolution von Softwaresystemen. In Abbildung 2.11 ist eine Methoden Sicht mit farblicher Alterung der Software.

EvoStreets Approach

Bei genauer Betrachtung aller oben genannten und gezeigten Ansichten fällt auf, dass diese auf einer Rechteckigen Grundfläche stehen. Die Platzausnutzung ist ein wichtiger Aspekt für

[51] siehe Abbildung 2.9

(a) Kostenfaktoren in der Software City



(b) gebündelte Verbindungen

Abbildung 2.10 Ausdrucksstarke Software City
Quellen: Panas, Berrigan und Grundy (2003) und Panas, Epperly, Quinlan, Saebjornsen und Vuduc (2007)

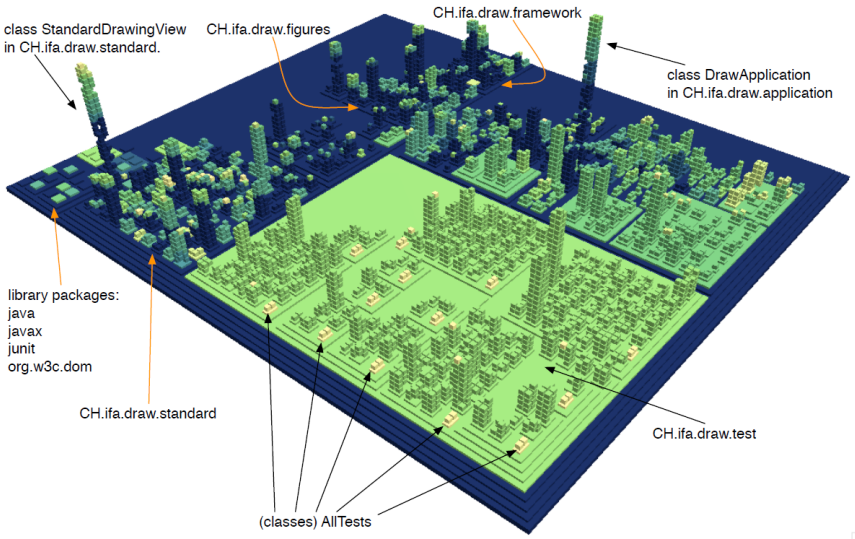


Abbildung 2.11 Evolution in im CodeCity Ansatz
Quelle: Wettel und Lanza (2008a)

Visualisierungen und muss sich nach den Darstellungsoptionen richten um optimiert zu sein. In Steinbrückner (2013, p.60) zeigt auf, dass aus den Perspektiven der Platzausnutzung, Ausdrucksstärke und Konsistenz der TreeMap Ansatz geeigneter ist, als nur ein Rechteck basierten Pack Algorithmus zu verwenden wie er z.B. von CodeCities (Steinbrückner (2013, p.24-26)) verwendet wird. Desweiteren entwickelte Steinbrückner um die Konsistenz zu steigern aus dem TreeMap Ansatz die EvoStreets. Im Fokus des EvoStreets Ansatzes liegt es auch nach Alterung und Versionsänderungen einer Software, die unter anderem beinhalten Knoten zu verschieben ein möglichst konsistentes Stadtbild zu schaffen, damit das mentale Model bei der Betrachtung weites gehend intakt bleibt. Interessant ist dabei, dass die rechteckigen Pack Algorithmen darauf ausgelegt sind, ähnlich wie bei den meisten TreeMap Implementationen, einen festen Platz auszunutzen. Durch diesen Top-Down sind sie hervorragend für Darstellungen in fester Größe zu generieren, die zum Beispiel in Fenstern am Desktop oder ausgedruckt auf Papier, zu erstellen.

In ein Virtual Environment ist es jedoch keines Wegs notwendig einen festen Platz voll auszunutzen. Es gibt auch die Möglichkeit die Städte nach einem Bottom-Up Ansatz zu berechnen, dass sie soviel Platz verbrauchen wie notwendig um auf diese Weise Skalierungen in einem gut sichtbaren Bereich zu halten. Eine Stadt sich ausbreiten zu lassen hat gewisse Vorteile im Bereich der Wahrnehmung (siehe Kapitel 2.2.2). Durch große Abstände können schneller Gruppen erkannt werden und einzelne Gruppen leichter räumlich von anderen zu unterscheiden sind.

An diesen Grundlagen haben wir uns in unserer Modellierung in Kapitel 3.1.3 orientiert.

2.2.6 Kantenbündelung [MOR]

Dem Bündeln der Kanten[52] in Visualisierungen von Graphen kommt bei großen Datenmengen eine wichtige Rolle zu: Durch die Bündelung der Kanten kann die Aussagekraft der Darstellung erhöht und die Visualisierung übersichtlicher erhalten werden. Ein guter Überblick ist in Zhou (2008) und Deore und Paikrao (2013) zu finden.

Für unsere Arbeit hatten wir die Aufgabe gestellt, den Verlauf der Kanten im Virtual Reality-System mittels Kantenbündlung zu vereinfachen. Die in der Fachliteratur vorgestellten Algorithmen haben je nach Anwendungsfall unterschiedliche technische Voraussetzungen, Eigenschaften und Resultate. Für deren Bewertung haben wir uns zunächst einen schnellen Überblick über verschiedenen Algorithmen verschafft und deren Eigenschaften mit denen verglichen, die für uns die beste Kombination darstellen würde.

Balzer und Deussen (2007) zeigen eine Methode, wie man *Level-Of-Detail*-Informationen beim

[52] Unter dem Begriff *Bündeln von Kanten* fassen wir alle computergrafischen Algorithmen zusammen, die eine Kanten-Verschiebung zur Erhöhung der Übersichtlichkeit in Graph-Visualisierungen zum Ziel haben. In der Fachliteratur ist dieses Themenfeld unter verschiedensten Begriffen zu finden, als da wären *edge bundling*, *magnetic edges* oder *spring edges*.

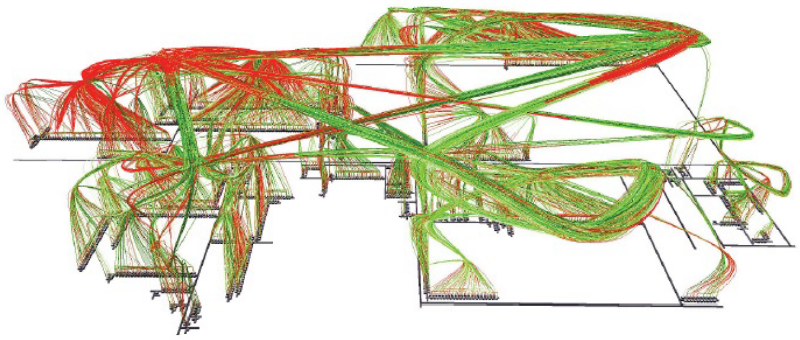
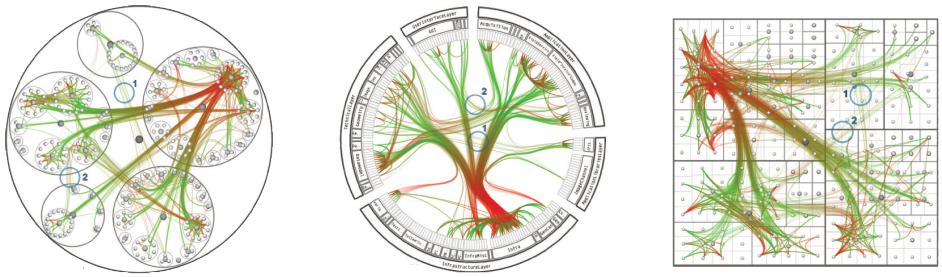


Abbildung 2.12 EvoStreets mit Verbindungen von Steinbrückner (2013)



(a) Draufsicht **(b)** Seitliche Perspektive **(c)** Seitliche Perspektive

Abbildung 2.13 Hierarchische Kantenbündelung
Die drei Darstellungen zeigen nach Holten (2006) die gleiche Software in unterschiedlichen Bündelungslayouts.

Layouting eines Graphen berücksichtigen kann. Athenstädt, Görke, Krug und Nöllenburg (2012) berichten, dass sie große Graphen zunächst in 2D erzeugen und und dann in die dritte Dimension 'liften'. Energiebasiertes Zusammenfassen (Bundling) von Verbindungen in Graphen ist in Zhou, Yuan, Cui, Qu und Chen (2008) beschrieben. Ein ähnliches Verfahren für die dritte Dimension beschreiben Zielasko, Weyers, Hentschel und Kuhlen (2016). In Guo, Zuo, Peng und Adhikari (2015) gehen die Autoren darauf ein, wie Attribute an Verbindungen für die Bündelung der Kanten nutzbar gemacht werden könnten. Gansner, Hu, North und Scheidegger (2011) beschreibt ein Möglichkeit zum Bündeln von Verbindungen in verschiedenen Ebenen. Dabei wird der Ansatz des 'tintesparens' verfolgt, welcher versucht, möglichst viele Verbindungen über die gleichen Wege zu führen, so der Weg nur einmal gezeichnet werden muss.

Neumann, Schlechtweg und Carpendale (2005) und später Holten (2006) beschreiben die Bündelung hierarchischer Graphen, die auch nicht-hierarchische Verbindungen enthalten. Holten nutzt dabei die hierarchische Struktur als Basis für das grundlegende Layout des Graphen und führt die anderen Verbindungen anhand der hierarchischen Struktur nach. Sansen, Lalanne, Auber und Bourqui (2015) erweitern die Konzepte auf gewichtete und gerichtete Graphen.

2.3 Software Engineering und Software-Reengineering [MOR]

> A science is as mature as its measurement tools.
> — Louis Pasteur (frz. Chemiker, 1822 - 1895)

Welche Art von Wissenschaft ist Software-Entwicklung? Ist sie überhaupt eine Wissenschaft? Und wenn ja, ist sie eher künstlerischer oder eher mechanistischer Natur? Doch welche Art von Wissen schafft sie eigentlich? Ist sie vielleicht *nur* Handwerk oder angewandte Mathematik? Über diese Fragen und weitere mehr philosophierten und philosophieren die wissenschaftlichen Disziplinen seit dem Beginn des Computerzeitalters.

Die Einordnung der Software-Entwicklung in den Kader der Wissenschaften war nicht leicht. Gegen die Tendenz, Software-Entwicklung als Ingenieur-Disziplin aufzufassen, da sie seinerzeit häufig in technischen und militärischen Zusammenhängen eingesetzt wurde, wehrten sich die Ingenieur-Disziplinen[53] und brachten die Vertreter der Software-Entwicklung in Erklärungsnot. Auf Initiative der NATO wurde 1969 eine Konferenz zum Thema „Software Engineering“ veranstaltet, die heute als Wendepunkt in der Wahrnehmung der Software-Entwicklung gesehen wird:

[53] vgl. bspw. den Leserbrief von Schmidt (1993); unterschlagen wurde und wird in dieser Diskussion stets, dass viele der Ingenieurdisziplinen rund 200 Jahre Vorlauf im Vergleich zur recht jungen Informatik haben

Software-Entwicklung ist eine Ingenieurdisziplin.[54]

Software-*Entwicklung* – worunter bis dato zumeist die *Neu*-Entwicklung von Software verstanden wurde – hatte im Laufe der Jahre in der Praxis der Software-Entwickler immer seltener mit der *Neu*-Entwicklung von Software zu tun. Insbesondere durch den verstärkten Einsatz von Software in wirtschaftlichen Zusammenhängen, in denen auf Kostenaspekte mehr Rücksicht als im militärischen Sektor genommen wurde, wurden Software-Entwickler immer häufiger mit der Überarbeitung, Anpassung oder Erweiterung bestehender, z.T. sogar unbekannter Software-Systeme statt mit deren Neu-Entwicklung beauftragt.

2.3.1 Software-Reengineering

Softwaresysteme wurden größer, verarbeiteten mehr Daten und konnten komplexer strukturiert werden. Immer neue, leistungsfähigerer Hardware mit immer höheren Speicherkapazitäten, neue Features der Programmiersprachen und neue Programmierparadigmen erhöhten die Komplexität der erstellten Software. Oftmals hielt die Struktur- und Source-Code-Dokumentation mit dieser Entwicklung nicht Schritt – Dokumentationen jeglicher Form wurden entweder vernachlässigt oder gar nicht erst beauftragt. Bei der Neuentwicklung der Systeme machte sich niemand Gedanken darüber, wie lange die Systeme denn laufen sollen: Das Jahr-2000-Problem war einer der Auswüchse der kurzsichtigen, nicht-nachhaltigen Softwareentwicklung.

Im Laufe der Jahre mussten daher neben Methoden zur Neu-Entwicklung von Software auch solche erdacht werden, mit denen bestehende Software auf verschiedensten Ebenen analysiert und wieder rekonstruiert werden kann. Nachdem diese Methoden eine Zeit lang unter verschieden Bezeichnungen, aufbauend auf verschiedenen Artefakten und mit unterschiedlichen Zielen entwickelt wurden, fassten Chikofsky und Cross (1990) die Methoden unter dem Begriff „Software-Reengineering“ zusammen. In Abbildung 2.14 sind Ihre Ideen zusammengefasst. Mit dem Begriff *Forward Engineering* bezeichneten Chikofsky und Cross 1990 das seinerzeit übliche, wasserfallartigen Vorgehen, um es vom *Reengineering* zu unterscheiden: Es beginnt bei einer Anforderungsspezifikation, diese wird im Anschluss in eine Modellierung überführt und schließlich als Programm umgesetzt.

Auf dem Weg von den Anforderungen zum fertigen Programm entstehen verschiedene Artefakte, die alle – soweit noch vorhanden – in ein Reengineering eingehen können. Dabei muss der Software-Reengineer[55] nicht den gesamten Weg wieder zurück gehen, sondern lediglich soweit, wie notwendig.

Ungefähr zeitgleich zu Chikofsky und Cross entstanden eine Reihe neuer sogenannter Vorgehensmodelle für die Software-Entwicklung: Das Modell, welches im größten Kontrast zum bis

[54] vgl. Naur und Randell (1969) sowie Bauer (1993)
[55] der *Software-Reengineering-Ingenieur*

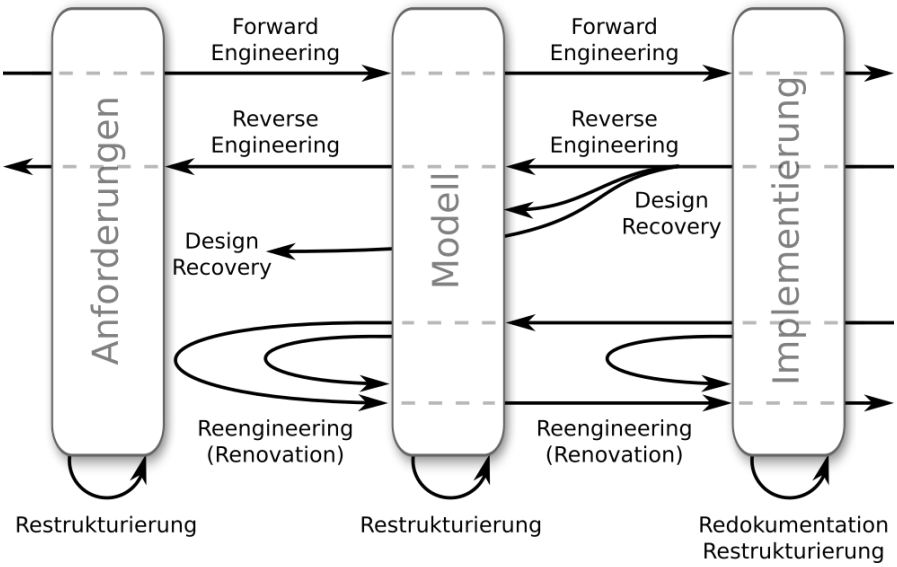


Abbildung 2.14 Begriffe des Reengineerings, drei Artefakten eines wasserfallartigen Software-Entwicklungsmodells zugeordnet; nach Chikofsky und Cross (1990)

dato verfolgten wasserfall-artigen Vorgehen stand, war wohl das Spiralmodell von Boehm (1988). Dieses empfahl, die geradlinige Vorgehensweise durch eine zyklisches abzulösen, um durch eine spiralförmige Annäherung an das Ziel flexibler auf Änderungen reagieren zu können.[56] Insofern war die Definition und Abgrenzung von Chikofsky und Cross eigentlich schnell überholt; da es sich allerdings um die erste ihrer Art handelte, wird sie bis als Referenzmodell angesehen.

Definition Software-Reengineering (Chikofsky/Cross)
Untersuchung und Modifikation eines Programmsystems, um es in einer neuen Form wiederherzustellen und diese Form nachfolgend zu implementieren.

Definition 2.6 Definition *Software-Reengineering* nach Chikofsky und Cross (1990), hier in der Überarbeitung von Müller (1997)

Definition Software-Reengineering (Arnold)
Alle Aktivitäten nach Inbetriebnahme eines Programmsystems zusammengefasst, die das Verständnis von Software erhöhen oder die Wartbarkeit, Wiederverwendbarkeit oder Weiterentwickelbarkeit von Software verbessern oder erst ermöglichen.

Definition 2.7 Definition *Software-Reengineering* nach Arnold (1993), hier in der Überarbeitung von Müller (1997)

[56] Diese Flexibilität passte den betriebswirtschaftlichen Abteilungen der Softwarehäuser und Anwenderunternehmen zunächst gar nicht, da sich die Kalkulation dadurch erschweren würde und Budget-Planungen nahezu unmöglich würden.

Definition Software-Reengineering (McClure)
Prozess der Untersuchung und/oder Modifikation eines Software-Systems mit Hilfe automatisierter Werkzeuge. Ziel ist die Verbesserung der Wartbarkeit und der verwendeten Technologie, die Erhöhung der Lebenserwartung und die maschinelle Verwaltung der Systemkomponenten zur Unterstützung von CASE-Tools.

Definition 2.8 Definition *Software-Reengineering* nach McClure (1992), hier in der Überarbeitung von Müller (1997)

Definition Software-Reengineering (Eicker)
Software-Reengineering (SRE) umfasst die Analyse von Anwendungssystemen und die anschließende grundlegende Überarbeitung der Systeme, um ihre Qualität signifikant zu verbessern. Analog zum Software Engineering erfolgen die Analyse und die Überarbeitung ingenieursmäßig, insbes. unter Einsatz geeigneter Vorgehensweisen, Methoden und Werkzeuge.

Definition 2.9 Definition *Software-Reengineering* nach Eicker (2011), hier in der Überarbeitung von Eicker (2011)

Eicker sieht eine zwingende Überarbeitung im Anschluss an die Analyse ein, offenbar allerdings frei von Qualitätszielen. Nach Arnold beginnt das Software-Reengineering erst nach der Auslieferung und McClure (1992) besteht auf dem Einsatz automatisierender Werkzeuge – wie auch Baumöl, Borchers, Eicker, Hildebrand, Jung und Lehner (1996) bereits dargelegt haben, ist das Thema *Software-Reengineering* schwer einzugrenzen. Jede dieser Definition scheint uns mit ihrem Zeitalter koloriert und ins Arbeitsumfeld ihrer Schöpfer hineingeboren. Unserer Ansicht nach lassen sich heute die Aufgaben des Forward-Engineerings nur noch schwer von denen des Reengineerings trennen; eine ganze Reihe von Aufgaben und Methodiken fallen im einen Bereich genauso an wie im anderen.[57] Unserer Ansicht nach unterscheiden sich die dahinter stehenden Tätigkeiten[58] heutzutage lediglich durch die unterschiedliche Intensität des Einsatzes einzelner Methoden und – das ist unserer Ansicht nach die Essenz – durch unterschiedliches Vorgehen (d. h. das Prozessmodell), auch wenn die Voraussetzungen und Artefakte beider Tätigkeiten unterschiedlich sind. Koschke (2013) drückt dies durch seine rhetorischen Unterstellung „Software engineering is reengineering on the empty system. Is it?“ aus.

In Abbildung 2.15 von Lichter (2007) sind Bestandteile des Reengineerings in eine Art Vorgehensmodell übertragen. Allerdings hat Lichter eine von Sneed, Hasitschka und Teichmann (2005) abweichende Meinung über die Bestandteile des Reengineerings, so dass einige Elemente in Tabelle 2.1 fehlen.

[57] Dies macht es natürlich auch schwer, konkrete Aussagen über den prozentualen Anteil an Wartungsaufgaben zu tätigen, was ein Grund sein mag, warum die entsprechenden Studien bereits älteren Datums sind.
[58] denn das sind sie nun mal primär: Tätigkeiten

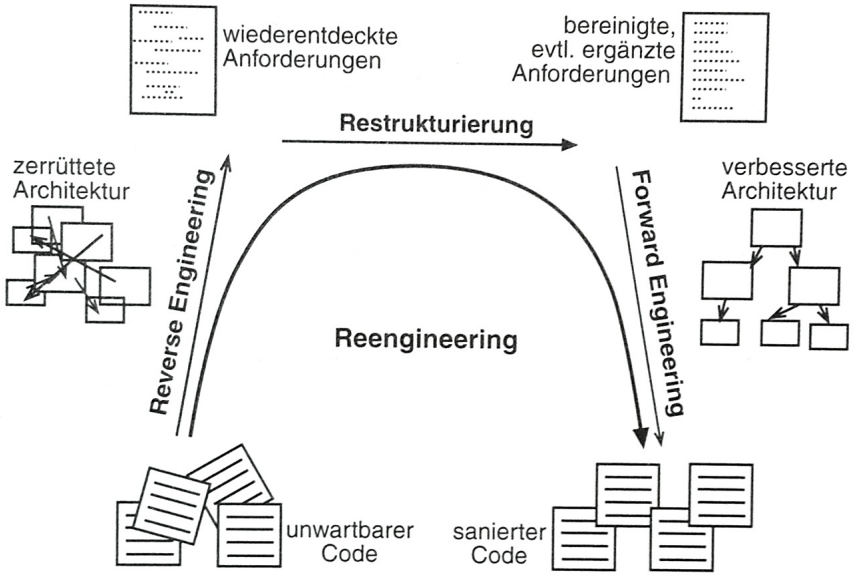


Abbildung 2.15 Vorgehensweise beim Reengineering
Quelle: Lichter (2007, S. 552)

2.3.2 Software-Reengineering: Aufgaben

Nachdem wir nun grob den Bereich Software-Reengineering in das Software-Engineering eingeordnet haben, wollen wir nun der Frage nachgehen, welche Methoden denn eigentliche im Software-Reengineering konkret zu finden sind. Dies ist für uns insbesondere deshalb interessant, da wir ein System entwickeln wollen, welches Entwickler bei bestimmten Tätigkeiten des Software-Reengineering unterstützen können soll.

Baumöl, Borchers, Eicker, Hildebrand, Jung und Lehner (1996) legten dar, dass es schwierig ist, die genauen Aufgaben des Software-Reengineerings zu fassen, was einerseits dem Versuch der scharfen Trennung zwischen Forward- und Re-Engineering liegen mag. Und so werden dem praktisch orientierten Leser in den folgenden Aufzählungen garantiert Aufgaben auffallen, die er in seiner Praxis als Software-Entwickler erledigt, obwohl er keine explizite Reengineering-Ausbildung erhalten hat.

Zu den Aufgaben des Software-Reengineerings zählen Sneed, Hasitschka und Teichmann (2005) die in Tabelle 2.1 aufgelisteten Maßnahmenkategorien. Konkreter wird – zumindest für quellcodebasierte Analysearbeiten – Koschke (2013) in seiner eher praxisorientierten Vorlesung und stellt die in Tabelle 2.2 aufgeführten Methoden vor.

Die Hypothese unserer Arbeit[59] bezieht sich auf die Visualisierung von Metriken von und Abhän-

[59] siehe Seite 105

Aufgabe	Beschreibung
Korrektive Wartung	sofortige Fehlerbeseitigung
(Release-getriebene) korrektive Systemerhaltung	versionsbasierte Fehlerbeseitigung
Erweiterung & Adaption	Anpassung bestehender Software an geänderte Anforderungen
Migration	Überführung eines Softwaresystems in ein anderes Environment
Integration	Zusammenfassung und Verzahnung verschiedener Softwaresysteme
Sanierung	Umstrukturierung der Software mit dem Ziel einer Erhöhung der Softwarequalität ohne Änderung der Funktionalität
Re-Dokumentation	nachträgliche Erstellung von Dokumentationen auf allen Ebenen
Ablösung	Prozess der Außerdienststellung einer Software

Tabelle 2.1 Maßnahmenkategorien des Reengineering nach Sneed, Hasitschka und Teichmann (2005)

gigkeiten zwischen Programmteilen, daher werden wir im folgenden lediglich diese Bereich etwas intensiver beleuchten. Nichtsdestotrotz wäre die Frage des Einsatzes von VR-Visualisierungen für die anderen Arbeitsbereiche des Software-Reengineers ebenso interessant.

2.3.3 Software-Metriken [MOR]

To measure is to know.
— James Clerk Maxwell (Physiker, 1831 - 1879)

Kennzeichnend für Ingenieur-Disziplinen ist u. a. die Messbarkeit der Prozesse und deren Ergebnisse. In der Software-Entwicklung sind Lines of Code eine frühe entstandene und weitverbreitete, statische und objektive Messgröße. Im Laufe der Jahre wurden immer neue Messgrößen vorgestellt, die beispielsweise neue Programmierparadigmen vermaßen oder bestehende Messgröße verfeinerten. Objektive Metriken werden dabei von subjektiven und Pseudo-Metriken abgegrenzt. In Tabelle 2.3 haben wir überblicksartig in Anlehnung an Lichter (2007) die Vorteile, Nachteile und Einsatzbereiche dargestellt.

Aufgabe	Beschreibung
Statische Programmanalysen und -repräsentationen	Analyse des statischen Aufbaus einer Software
Dynamische Programmanalyse	Analyse des dynamischen Verhaltens einer Software
Programm-Slicing	Analyse der Beeinflussung eines bestimmten Softwarebestandteils
Software-Produkt-Metriken	Ermittlung von Kennzahlen über die Software
Erkennung duplizierten Codes und anderer Bad Smells	Erkennung von *Copy-Paste*-Code und anderen schlechten Programmiermustern
Refactoring und Transformationen	(funktions- oder architekturerhaltende) Umstrukturierung einer Software
Codetransformationen	(halb-)automatische Transformation von Sourcecode
Software-Visualisierung	neue Sichten auf eine Software
Analyse und Restrukturierung von Vererbungshierarchien	Architekturrefactoring
Merkmalsuche (Feature-Location)	Elemente finden, die für bestimmte Funktionalitäten zuständig sind
Software-Clustering, Architekturrekonstruktion und -validierung	Rekonstruktion der Architektur oder von Teilen der Architektur

Tabelle 2.2 Quellcode-basierte Aufgabenbereiche des Software-Reengineerings nach Koschke (2013)

	objektive Metrik	subjektive Metrik	Pseudometrik
Verfahren	Messung, Zählung, eventl. nach Normierung	Beurteilung durch Gutachter, verbal oder auf vorgegebener Skala	Berechnung (auf Basis von Messungen und/oder der Beurteilungen)
Vorteile	exakt, reproduzierbar, automatisch erhebbar	nicht unterlaufbar, plausible Resultate, auch für komplexe Merkmale geeignet	liefert relevante, unmittelbar verwertbare Aussage über nicht sichtbares Merkmal
Nachteile	nicht sicher relevant, meist unterlaufbar, keine Interpretation	Erhebung aufwändig, Qualität der Resultate hängt stark von den Gutachtern ab	schwer nachvollziehbar, pseudoobjektiv
Beispiele[60]	LOC oder NCSI, Zahl der Fehler	Usability, Schwere eines Fehlers	Produktivität, Kostenschätzung

Tabelle 2.3 Arten von Metriken (nach Lichter (2007))

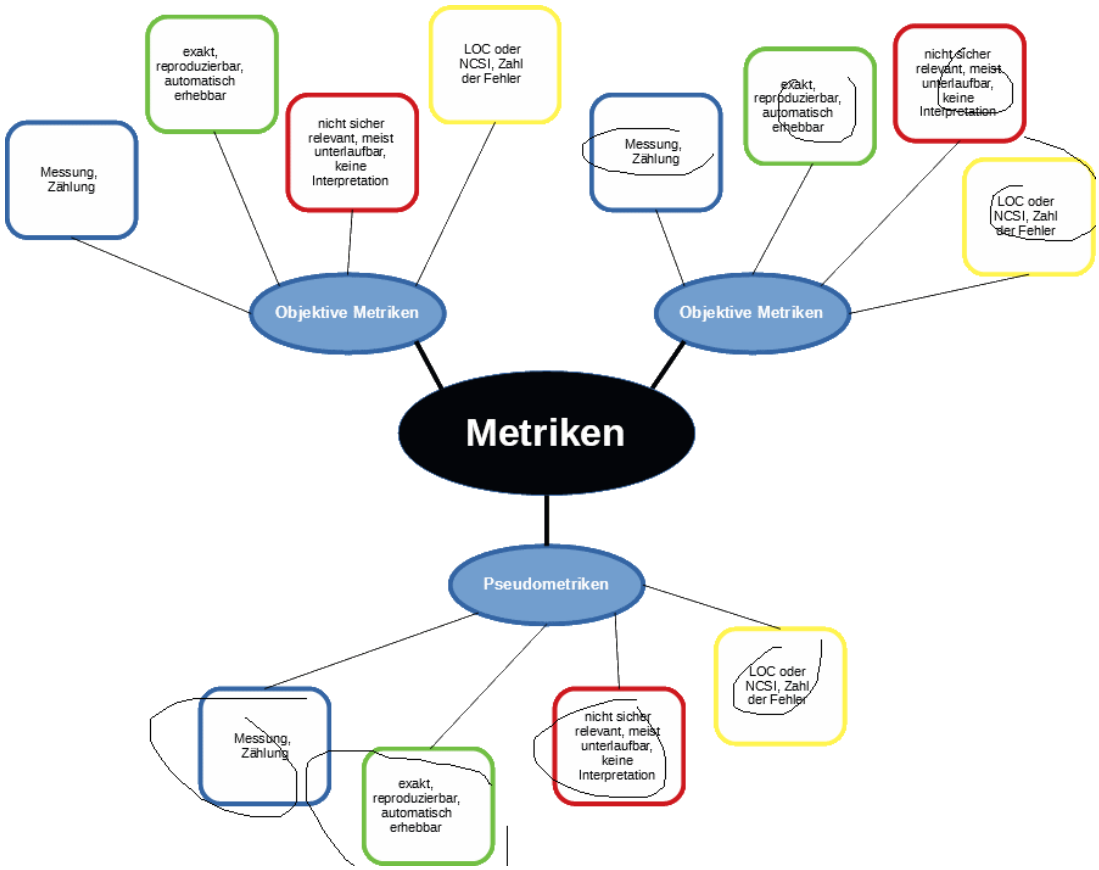


Abbildung 2.16 Metrikarten nach Lichter (2007)

Mittels dieser Messgrößensysteme (Metriken) konnten die Projektbeteiligten dynamische[61], strukturelle[62] oder Meta[63]-Aspekte des Software-Entwicklungsprojekts beurteilen. In Tabelle 2.3 finden sich einige Beispiel für Metriken der jeweiligen Aspekte.[64]

Aus dem vorherigen Absatz geht bereits indirekt hervor, dass das Erstellen von Metriken natürlich nicht zum Selbstzweck erfolgt – Dinge vergleichen zu können ist offenbar der primäre Zweck der Metrikerstellung. Und ein Vergleich erfolgt zumeist, um die Dinge kategorisieren und die Welt abstrahieren zu können. Mit einer Kategorisierung wiederum geht auf natürliche Weise eine emotionale oder technische Bewertung einher. Erst durch den Vergleich der gemessenen Werte eines Systems mit einem anderen lassen sich Aussagen über die Qualität in bestimmten Kategorien treffen.

Metriken lassen sich für die Erkennung von sogenannten Bad Smells oder Code Smells einsetzen. Der Begriff Code Smell wurde wahrscheinlich von Kent Beck 1990 geprägt[65] und steht für ein Symptom einer Software, welches auf ein (bspw. architekturelles) Problem schließen lässt. Bad Smells können als Qualitätsmerkmal einer Software verstanden werden. So zeigt die Erfahrung und Untersuchungen beispielsweise, dass sich sehr lange Methoden oder Klassen mit vielen Methoden nachteilig auf das Programmverstehen auswirken können. Fowler (2003) beschrieb bestimmte Muster, die Programmen zwar häufig vorkommen, sich aber eigentlich negativ auf die Qualität der Software oder deren Entwicklungs- oder Wartungsprozess auswirken. Diese Muster frühzeitig zu erkennen und beheben, wird heute als ein wichtiger Bestandteil sowohl des Software Forward- als auch des Reengineering gesehen. tesiehe Wikipedia *https://de.wikipedia.org/w/index.php?title=Smell_(Programmierung)&oldid=149680565* und steht für ein Symptom einer Software, welches auf ein (bspw. architekturelles) Problem schließen lässt. Bad Smells können als Qualitätsmerkmal einer Software verstanden werden. So zeigt die Erfahrung und Untersuchungen beispielsweise, dass sich sehr lange Methoden oder Klassen mit vielen Methoden nachteilig auf das Programmverstehen auswirken können. Fowler (2003) beschrieb bestimmte Muster, die Programmen zwar häufig vorkommen, sich aber eigentlich negativ auf die Qualität der Software oder deren Entwicklungs- oder Wartungsprozess auswirken. Diese Muster frühzeitig zu erkennen und beheben, wird heute als ein wichtiger Bestandteil sowohl des Software Forward- als auch des Re-Engineering gesehen.

Wir werden in dieser Arbeit weder tiefer auf die Berechnung von Metrik eingehen[66] noch versuchen, eine Liste[67] von Bad Smells zu erstellen. Die konkrete Berechnung bestimmter Metrik wie

[61] bspw. für Vermessung der Ablaufzeit bestimmter Programmteile
[62] bspw. die Anzahl der Abhängigkeiten bestimmter Art von einer anderen Programmeinheit
[63] Sozio-organisatorische Metriken; hier wären bspw. die Anzahl der Stunden zu nennen, die ein Mitarbeiter an einem Programmcode verbracht hat.
[64] Interessanterweise wird im Glossar der IEEE unter dem Begriff *metric* lediglich die objektive Metrik (und eine Qualitätsmetrik, basierend auf dieser) beschrieben. Subjektive, qualitative Metriken scheinen für die IEEE keine Bedeutung zu haben. (siehe »IEEE Standard Glossary of Software Engineering Terminology« (1990))
[65] siehe Wikipedia *https://de.wikipedia.org/w/index.php?title=Smell_(Programmierung)&oldid=149680565*
[66] siehe hierzu u. a. Grechenig (2010)
[67] Natürlich ist es nicht bei den Code Smells von Fowler und Beck (1999) geblieben. Inzwischen gibt es auch eine

Lines-of-Code ist in den einschlägigen Fachbüchern zum Software Engineering oder Qualitätsmanagement in der Softwareentwicklung beschrieben; sie ist eventuell ein wenig kniffelig, aber in der Regel keine Zauberei.

Für die Überprüfung unserer Hypothese greifen wir auf das Werkzeug Bauhaus Toolkit zurück, welches uns für einen gegebenen Programmquellcode einen Graphen, verknüpft mit bestimmten metrischen Informationen, liefern kann.[68] Aus diesen Informationen könnte man durch Abstraktion und Vergleich mit den üblichen Werten Bad Smells extrahieren und die verursachenden Programmelement identifizieren.

ganz Reihe architektonischer Smells, die ebenfalls bei der Konstruktion von Software berücksichtigt werden sollen, siehe hierzu bspw. Starke (2014)

[68] Details hierzu siehe Kapitel 2.3.6

Software-Metriken und Fahrzeug-Metriken lassen sich zur Erläuterung grundsätzlicher Eigenschaften von Metriken in Analogie bringen. Denn auch für motorgetriebene Fahrzeuge aller Art gibt es eine Reihe von Messwerten, die in unserem Zeitalter ein jeder kennt: PS, KW, Höchstgeschwindigkeit, und – für einen kleineren Teil der Gesellschaft – Anhängelast, Zuglast, Energieeffizienzklasse.
Wir lernen bereits als Kind durch das Spielen von *Auto-Quartett*[69], dass Fahrzeuge mit einer hohen PS-Zahl nicht unbedingt die schnellsten sind. Und wir lernen, dass sich manche Metriken durch Rechnung direkt und umkehrbar in andere überführen lassen. Wir entdecken, dass es – wenn wir beispielsweise Flugzeug- und Automobil-Quartett spielen, in unterschiedlichen Fahrzeug-Quartetts zwar identische Metrik-Bezeichnungen (bspw. PS) gibt, deren mittlere Werte aber stark voneinander abweichen.
In dieser sozial-spielerischen Auseinandersetzen über das Quartett lernen wir auch, dass es offenbar weitere mögliche Werte gibt, die indes nicht aufgeschrieben wurden. Durch Aussagen wie „Sieht der cool aus!“ stellen wir fest, dass beispielsweise auch die „Schönheit“ der Fahrzeuge unterschiedlich bewertet wird. Allerdings stellen wir durch andere Aussagen („Nee, der Heckspoiler ist viel zu klein, diese Karre ist viel cooler!“) auch fest, dass diese Bewertung offenbar subjektiv gefärbt ist. Die anderen Werte wie PS-Zahl und Energieeffizienzklasse können direkt und ingenieurmäßig (somit auch wiederholbar auf einem definierten Verfahren basierend) ermittelt werden. Die Schönheit des Fahrzeugs ließe sich durch Methoden der empirisch-statistische Methoden der Sozialforschung ermitteln, ließe aber viele Freiheitsgrade und wäre daher stets zu hinterfragen.
Die bereits genannte *Energieeffizienzklasse* ist auch aus anderen Bereichen des täglichen Lebens bekannt.[70] Sie klassifiziert den Energieverbrauch eines Gerätes auf einer Ordinal-Skala[71] und soll dem Käufer die Kaufentscheidung bzgl. des Energieverbrauchs erleichtern. Auf das Thema Metriken bezogen macht sie auf eine andere Schwierigkeit im Umgang mit ihnen aufmerksam: Gegebenenfalls hat mein Kleinwagen eine bessere Energieeffizienzklasse als mein neuer Toaster, oder – um bei Fahrzeugen zu bleiben – der fette, tonnenschwere SUV eine bessere Energieeffizienzklasse als mein Kleinwagen, obwohl er mich beim Schnellstart an der Ampel regelmäßig alt aussehen lässt? Dies liegt daran, dass die Berechnung der Metrikwerte auf weiteren Werten beruhen kann – im Falle der Energieeffizienzklasse auf einer vordefinierten Geräteklasse.

Beispiele 2.2 Beispiel für den Einsatz von Metriken im Alltag
hier: Quartett-Kartenspiel und Energieeffizienzklassen

Eine Softwarequalitätmetrik ist eine Funktion, die eine Software-Einheit in eine Zahlenwert abbildet. Dieser Wert ist interpretierbar als der Erfüllungsgrad einer Qualitätseigenschaft der Software-Einheit.

Definition 2.10 Definition Softwarequalitätsmetrik nach in der Bearbeitung von Grechenig (2010)

Name	Art	Beschreibung
Lines of Code (LOC)	objektiv	Anzahl der Zeilen einer Programmdatei
Source Lines of Code(SLOC)	objektiv	Anzahl der Programm-Zeilen einer Programmdatei
Comment Lines of Code(CLOC)	objektiv	Anzahl der Kommentar-Zeilen einer Programmdatei
Non-Comment Lines of Code(NCLOC)	objektiv	Anzahl der Nicht-Programm-Zeilen einer Programmdatei
Logical Lines of Code(LLOC) / Number of Statements (NOS)	objektiv	
Halstaedt	objektiv	Anzahl der Operanden und Operatoren eines Programmcodes
McCabe	objektiv	
Koschke		

Tabelle 2.4 Ausgewählte Metriken
Die angegebenen Arten beziehen sich auf Tabelle 2.3

Software-Metriken sind nach Müller (1997) interessant, wenn es um Fragen des Programmverstehens oder der Erhöhung der Programmqualität in der Wartungsphase geht. Doch wird Software nicht losgelöst entwickelt, sondern steht viel mehr in wirtschaftlichen oder sozialen Wettbewerbszusammenhängen – auch Open-Source-Software und Freeware. In vielen Software-Häusern, die moderne Projektmanagementparadigmen wie *Xtreme Programming* folgen, werden inzwischen Metriken zur Erfolgskontrolle der einzelnen Projektphasen (und Programmierer[72]) eingesetzt, auch wenn – wie Lichter (2007) anführt – im Software-Bereich (noch) nicht mit Metriken zu Qualitätsbeurteilung gerechnet wird, vermutlich, weil es sich bei Software um ein immaterielles Gut handele. Fenton und Pfleeger (1998) heben bzgl. der Ziele des Metrikeinsatzes auf eine Verbesserung des Programmverstehens, eine Erhöhung der Qualität, eine einfachere und bessere Erfolgskontrolle, eine höhere Vergleichbarkeit sowie eine validere Kostenprognose ab.

„You cannot control what you cannot measure, and you can't control what you don't measure." schrieb DeMarco (1982) und traf damit offenbar ins Herz der Software-Entwicklung: Sie können nicht steuern, was sie nicht messen können. Dieser vielzitierte Satz stammt aus einer Zeit, in der das *Was* und *Wie* der Vermessung von Software und insbesondere der Leistung der Programmierer und des Projekts noch nicht entschieden war. Die Auswahl einer/mehrerer aussagekräftigen Metrik/en steht heute im Fokus der Projektsteuerung, gleichwohl der Einsatz von Metriken mit

[72] Insofern suggeriert *Software-Visualisierung* eigentlich etwas falsches: Zwar wird die Software vermessen, bemessen wird mit den Metriken aber anderes.

bedacht geschehen sollte.[73]

Auch wenn der Wunsch nach *einer* einfachen Metrik (quasi nach *der* Metrik) immer im Hinterkopf residiert,[74] so sieht die Praxis doch anders aus: Die Masse möglicher Analysen und deren diese Zahl noch bei Weitem übersteigende Zahl möglicher Korrelationen erster, zweiter und n-ter Ordnung erschweren eher die Wahrnehmung von Zusammenhängen und damit einem möglicherweise besseren Programmverstehens.

Aber selbst, wenn sich Softwareentwickler oder Projektmanager lediglich mit einer klassischen Lines of Code-Metrik die Entwicklung eines System verdeutlichen wollen, werden sie bei modernen Systemen allein schon von der Menge der Bestandteile auf der Software-Seite[75] erschlagen: Beispielsweise wartet das Paket .NET 3.5 (SP1) von Microsoft™ mit 112 Assemblies, 935 Namespaces, 40 513 Typen und Klassen, 386 790 Methoden und 246 795 Feldern[76] auf, was es für den Anfänger sehr schwer macht, sich in das System einzuarbeiten und selbst erfahrene, langjährige Programmierer immer wieder an die Grenzen Ihres Wissens bringt. Sollen solche Systeme während des Produktionsprozesses oder nach Inbetriebsnahme[77] analysiert werden, stellt sich nicht nur die Frage nach der geeigneten Metrik und deren geeignete Interpretation, sondern auch einer geeigneten Repräsentation der metrischen Informationen.

Textuelle Tabellenform ist die klassische Darstellungsvariante für metrischen Informationen. Dies wird einerseits pragmatische Gründe gehabt haben, da andere Darstellungsformen nicht vorhanden oder aufwändig zu erstellen waren und sich somit für die sich schnell verändernden Daten auszahlten. Andererseits werden solche Datentabellen – seien sie auch sortier-, filter- und durchsuchbar – schnell unübersichtlich und zerstören den Bezug zum betrachteten Objekt; hier: der Software.

[73] siehe dazu auch Kan (2002). In der Tat hat sogar Tom DeMarco mehrfach sein obiges Zitat grundsätzlich relativiert, bspw. in DeMarco, Märtin und Märtin (1997). In DeMarco (2009) fokussiert er bei seiner Kritik auf einen anderen Aspekt als Kan: Tom DeMarco sieht den kreativen Anteil der Software-Entwicklung durch die strenge Orientierung an Metriken in Gefahr und deutet darauf hin, wie viele interessante Projekte sich erst durch das Weglassen von Metriken entwickeln konnten, siehe dazu auch *https://maximeboninblog.wordpress.com/2017/08/09/tom-demarco-about-the-dark-side-of-metrics/*

[74] Nach Barnard und Price (1994) sind die folgenden neun Metrik ausreichend, um mittels der Goal Question Metrik Methode zu Ergebnisse zu gelangen: KLOC (Kilo-Lines of Code ohne Kommentare), Durchschnittliche Anzahl inspizierter Codezeilen, durchschnittliche Vorbereitungsrate, durchschnittliche Inspektionsrate, durchschnittlicher Aufwand pro KLOC, durchschnittlicher Aufwand für einen gefundenen Fehler, durchschnittlich gefundene Fehler je KLOC, Prozent der Re-Inspektionen und Fehlerentfernungseffizienz

[75] häufig ist Software – gerade auch in industriellem Kontext in ein Hardware- und Organisationssystem eingebettet

[76] Daten von *http://codebetter.com/patricksmacchia/2008/08/13/net-3-5-sp1-changes-overview/*, ermitteln mit dem Tool NDepend; auch wenn konkreten Werte auf *https://stackoverflow.com/questions/1406996/number-of-classes-in-net* angezweifelt werden (was bei vermessenen Werten schon für sich alleine interessant ist), so wird mindestens die Größenordnung stimmen

[77] siehe zu den unterschiedlichen Definitionen des Software-Reengineering Kapitel 2.3.1

Metrik-Visualisierung

> Es gibt also gute Gründe, Metriken einzuführen, und ebenso gute Gründe, dabei behutsam und systematisch vorzugehen und die Ziele nicht zu hoch zu stecken, sondern nur wenige wichtige Metriken konsequent zu erheben und zu verwenden.
> —Lichter (2007, S. 279)

In diesem Abschnitt schlagen wir nun den Bogen ins Kapitel 2.2, denn für die Darstellung großer, komplexer oder versteckter Informationen eignen sich – wie dort angedeutet – Visualisierungen. Wir haben bereits dargestellt, dass Softwarevisualisierung seit jeher für die Visualisierung komplexer Zusammenhänge verwendet werden, auch wenn die verwendete Vielfalt in den meisten Fachbüchern nicht erwähnt und nur die Unified Modeling Language als standardisierte Visualisierung beschrieben wird. So nützlich sie auch für eine Standardisierung der verschiedensten Modellierungsansätze war: Die Unified Modeling Language hat auch Nachteile, dass ist jedem Software-Modellierer bekannt.[78]

Grundsätzlich lassen sich metrische Informationen mit beliebigen, nummerischen Parametern einer Metapher verbinden. Wir nennen dieses Verfahren *Mapping*.[79] Bei den in in dieser Arbeit angeführten historischen Beispielen handelt sich nahezu ausschließlich um Metrik-Visualisierungen.

2.3.4 Abhängigkeiten [MOR]

> Question: How do you make two systems loosely coupled?
> Answer: don't connect them.
> — Orchard (2004)

Neben der Vermessung der Software ist ein weiteres großes Thema des Software-Reengineering die Arbeit mit Abhängigkeiten zwischen den Elementen der Software. Hierzu gehören Refactoring-Aufgaben wie das Verschieben von Code-Blöcken, Erkennen der Nutzung eines Members/eine Operation/einer Klasse/eines Packages auch über mehrere Zwischenschritte hinweg, Erkennen von ausgelösten Events und Dispatcheraufrufen aber auch Aufgaben des Programmverstehens

[78] u. a. ist die Einarbeitung in das Metamodell ist nicht nicht einfach, es existieren nur wenige ausgereifte Cartridges für den direkten Einsatz in der modellgetriebenen Softwareentwicklung und ohne Werkzeugunterstützung ist die Nutzung schwierig, es sind jedoch nur wenige ausgereifte Unified Modeling Language-Werkzeuge verfügbar; weitere bei Fettke (2005)
[79] siehe Kapitel 3.1.1

oder der Dokumentationsnachführungen könnten Bedarf an der Sichtung von Abhängigkeiten haben.

Die Sichtung von Abhängigkeiten bringt bisherige 2D-System wie die UML schnell an Grenzen. Vion-Dury, Santana und Bull (1994) konnten zeigen, dass es signifikante Unterschiede zwischen der Betrachtung von 2D- und 3D-Systemen gerade in Bezug auf Abhängigkeiten zwischen den dargestellten Elementen gibt. Hiernach lohnt sich der Einsatz von 3D für die Visualisierung von Abhängigkeiten.

2.3.5 Unreal Engine 4 als Reengineering-Projekt

> Wer das bisherige Framework, die bisherige Programmiersprache weiterverwendet, sieht vor sich selbst und anderen wenigstens halbwegs kompetent aus. Auf einem neuen Gebiet wäre man plötzlich wieder Anfänger.
> – Passig und Jander (2013, S. XVI)

Wir hatten in unserer Arbeit zum Ziel, Aussagen über die Einsatzfähigkeit von Head Mounted Display oder Desktop-Systemen für typische Aufgaben des Software-Reengineering zu treffen. Durch Konkretisierung wurde diese zunächst einfach erscheinende Aufgabenstellung doch sehr viel komplexer. Insbesondere da wir gerne eine Arbeit verfassen wollten, die für den Bereich Software-Reengineering von Vorteil wäre, stellte sich uns die Frage, welches denn typische Aufgaben des Software-Reengineerings sind und welche von diesen weiterhin sinnvoll mit Visualisierungen gelöst werden könnten. Leider gibt es hierzu in der Literatur nur wenig Anhaltspunkte oder wissenschaftliche Untersuchungen, auf die wir uns stützen konnten – das konkrete Vorgehen im Rahmen des Software-Reengineering scheint auf individuellem oder institutionellem Erfahrungswissen zu basieren.

Während unserer Implementierungsphase stießen wir auf einige Schwierigkeiten mit der Unreal Engine. Uns wurde dabei klar, dass wir unsere selbst gesteckten Ziele auf dem zunächst modellierten Wege nicht erreichen würden, verwarfen wir unser ursprüngliches In-Game-Konzept und fokussierten stattdessen auf eine In-Editor-Lösung. Unsere Schwierigkeiten rührten u. a.

- von unseren fehlenden Vorkenntnissen in diesem Bereich der Programmierung der Unreal Engine (Massendatenverwaltung, Grafik-Optimierung),
- von der fehlenden, veralteten oder unzureichenden Dokumentation der Klassenstruktur, Funktionalitäten und Zusammenhängen,
- von der schwierigen Zuordnung von Forumsbeiträge zu der von uns eingesetzten Unreal Engine-Version,

- vom umständliche, zeitaufwändige und z. T. sogar hinderliche Arbeit mit der vorgesehenen Programmierumgebung Microsoft Visual Studio, da für die Kompilierung unseres Projekts jeweils die komplette Unreal Engine als Referenz herangezogen werden musste,
- von der unübersehbaren Größe des Frameworks, das sich auch gegen grafische Analyse mit den in Visual Studio eingebauten Tools sperrte sowie externen UML-Tools aufgrund der komplexen Makrostruktur nur schwer zugänglich war
- von dem proprietären Makro-System der Unreal Engine, welches dem Programmierer einerseits viel Arbeit abnimmt und C++ sogar um Elemente wie Garbage Collection und Laufzeit-Typ-Informationen erweitert, die bislang in der Sprache nicht vorgesehen sind, andererseits allerdings auch gewissen Hürden und Fesseln auferlegt, die erst einmal durchschaut werden wollen, sowie
- von einem Game-Editor, der sehr umfang- und feature-reich ist, seine Geheimnisse aber nicht sofort Preis gibt.

Im Laufe des Projekts wurde uns – leider viel zu spät, sonst hätten wir rechtzeitig ein Meta-Projekt anstoßen können – klar, dass wir gerade selbst in einem Reengineering-Projekt stecken, denn was sind die aufgeführten Punkte anderes als typische Aspekte und Probleme von Reengineering-Projekten.

Mehr zu diesem Aspekt unserer Arbeit in dem Kapitel 3.2.

2.3.6 GXL – Graph eXchange Language [MOR]

Der XML-Dialekt GXL[80] wurde als gemeinsames Standard-Austauschformat für Software-Reengineering-Werkzeuge entwickelt. GXL basiert auf einer XML-DTD Document Type Definition und kann wie in Abb durch ein UML-Klassendiagramm repräsentiert werden.

Durch die Verwendung der GXL soll ein Software Reengineer in der Lage sein, einzelne, unabhängige Werkzeuge zu einer Toolbox zusammen zu schließen. Das konzeptionelle Modell der Sprache ist ein getypter, attributierter und gerichteter Graph, erweitert um Konzepte für Hypergraphen und hierarchische Graphen. Damit ist die Sprache so allgemein gehalten, dass sehr viele verschiedene Anwendungsfälle des Software-Reengineering abgehandelt werden können.

Die GXL entstand durch den Zusammenschluss verschiedener universitärer Vorläuferformate.[81]

Anhand des GXL-Graph-Modells aus Abb. 2.17 ist zu erkennen, dass ein GXL-Graph aus beliebig vielen Graphen bestehen kann. Jeder dieser Graphen kann wiederum sogenannten Graph-

[80] Graph eXchange Language, www.gupro.de/GXL/

[81] GraX (GRAph eXchange format, University of Koblenz, DE), TA (Tuple Attribute Language, University of Waterloo, CA), PROGRES graph rewriting system (University Bw München, DE); unter Verwendung von Ideen von: RPA (Relation Partition Algebra, Philips Research Eindhoven, NL), RSF (Rigi Standard Format, University of Victoria, CA); beeinflusst von vielen Formaten, die Graph-Zeichenwerkzeugen verwendet wurden, u.a. daVinci, GML, Graphlet, GraphXML

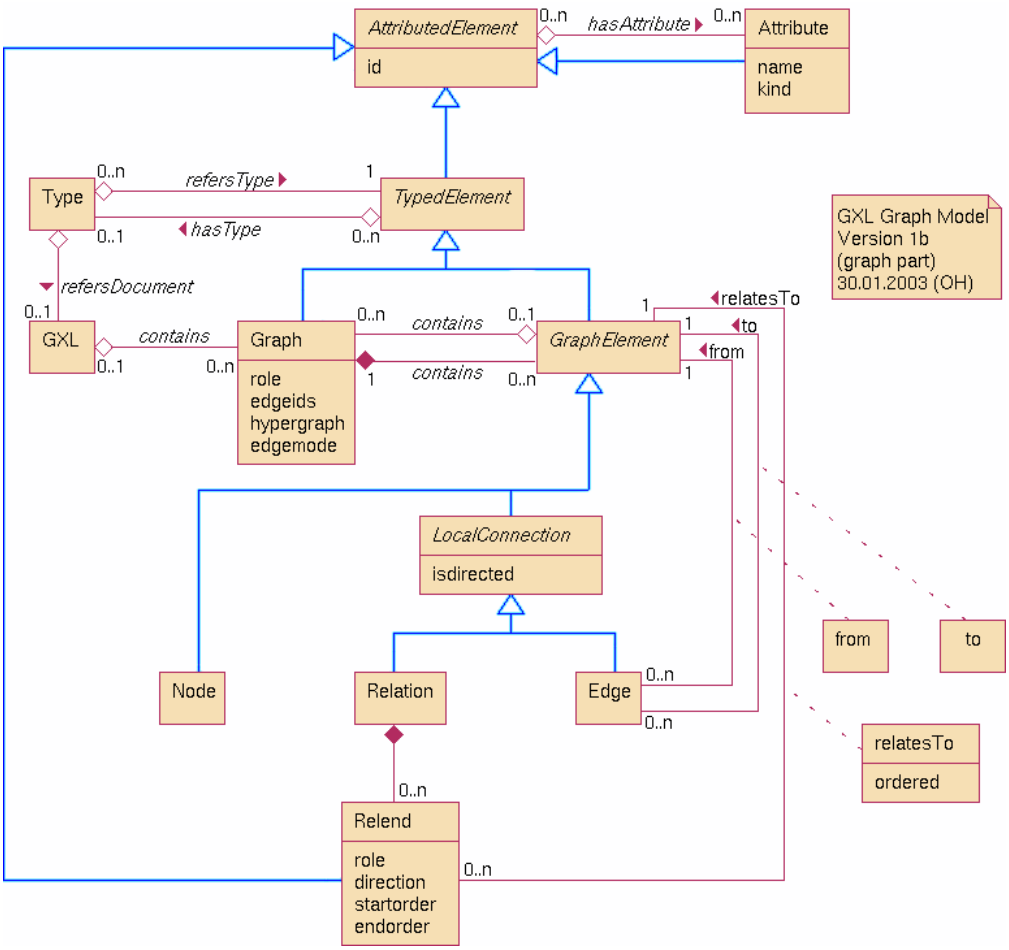


Abbildung 2.17 GXL-Graph-Model (ohne Attribute)
Quelle: *http://www.gupro.de/GXL/MetaSchema/graphpart.png*

Elemente enthalten, als da wären Knoten, Kanten, Beziehungen[82] und weiteren Graphen[83]. Jede Kante hat in GXL zwei Verbindungen zu anderen Graph-Elementen, bezeichnet mit den Rollen *from* und *to*, um die Richtung des Graphen festlegen zu können. Alle Elemente haben einen in der GXL definierten Type und können beliebig viele Attribute enthalten, von denen jedes durch einen Namen und eine Art bestimmt wird.

GXL-Beispieldokument [MOR]

Wir wollen an dieser Stelle ein durchgehendes Java-Beispiel einführen (siehe Listing 2.1 bis 2.4), an dem wir sowohl kurz die Umsetzung in GXL als auch in späteren Kapiteln die Visualisierung in unserem System demonstrieren. Bei dem Beispiel handelt es sich um eine einfacher Vererbungsstruktur mit drei Klassen sowie einer Verwaltungsklasse.[84] In Listing 2.5 wird das gleiche Java-Programm im GXL-Format gezeigt, um den Zusammenhang zwischen den beiden Repräsentationen zu verdeutlich. Wir gehen nicht weiter auf Details des GXL-Formats ein, die zwar programmiertechnisch von Belang waren, jedoch nicht für das weitere Verständnis relevant. Das vollständige GXL-Listing ist im Anhang A.1 zu finden.

Listing 2.1 Demoprojekt: Klasse A

```
1  package myPackage.package_A;
2
3  public class Class_A {
4
5     public String toString(){
6        return "Class_A";
7     }
8
9  }
```

Das Bauhaus Toolkit verwendet den UML-Dialekt GXL, setzt aber nicht die vorgesehene Strukturierung der GXL ein: Die GXL sieht zwar vor, dass ein Graph aus beliebig vielen Untergraphen bestehen kann, doch haben wir es im Falle von Softwaresystemen mit mehreren, sich überschneidenden Gruppierungen zu tun (bspw. Klassenhierarchie, Aufrufabfolge und Zugriffsabhängigkeit). Da die GXL keine Links auf Knoten zulässt, ließen sich diese multiplen Gruppierungen nur durch getrennte GXL-Dateien je Gruppierung bewerkstelligen, wobei jede Datei jeweils die vollständigen Knoteninformationen mitführen würde.[85]

[82] Verbindungen zu beliebig vielen Graph-Elementen, um Hypergraphen zu unterstützen
[83] um hierarische Graphen zu unterstützen
[84] Wir beschreiben das Beispiel nicht im Detail, da die konkrete Funktionalität für unser Vorhaben nicht von Belang ist.
[85] Für eine Zuordnungsfähigkeit zwischen den GXL-Dateien müsste dann ggfs. mittels Knoten-IDs gesorgt werden.

Listing 2.2 Demoprojekt: Klasse A1

```
1  package myPackage.package_A.package_A1;
2
3  import myPackage.package_A.Class_A;
4
5  public class Class_A1 extends Class_A{
6
7     public int plusY (int y) {
8        return x + y;
9     }
10
11    private int x = 0;
12
13    public int squareX(int x){
14       return x * x;
15    }
16
17    public void setX(int x){
18       this.x = x;
19    }
20
21    public String toString(){
22       return super.toString() + "; " + "Class_A1 (" + x + ")";
23    }
24
25 }
```

Listing 2.3 Demoprojekt: Klasse A2

```
1  package myPackage.package_A;
2
3  public class Class_A2 extends Class_A {
4
5     private int x = 0;
6
7     public int modXY(int x, int y){
8        return x % y;
9     }
10
11    public void setX(int x){
12       this.x = x;
13    }
14
15    public String toString(){
16       return super.toString() + "; " + "Class_A2 (" + x + ")";
17    }
18 }
```

Listing 2.4 Demoprojekt: Klasse B

```
package myPackage.package_B;

import myPackage.package_A.Class_A;

public class Class_B {

Class_A[] Class_As = new Class_A[10];

   public String toString(Class_A elem, int pos){
      if (pos >= 0 && pos < 10) {
         return Class_As[pos].toString();
      } else return "";
   }

}
```

Listing 2.5 Beispielprojekt in GXL

```
<?xml version="1.0" encoding="UTF-8"?>
   <gxl xmlns:xlink="http://www.w3.org/1999/xlink">
      <graph id="Code Facts" edgeids="true">
      <node id="N1">
      <type xlink:href="File"/>
      <attr name="Source.Region.Length"><int>15</int></attr>
      <attr name="Source.Name"><string>Class_B.java</string></attr>
      </node>
      <node id="N10">
      <type xlink:href="Method"/>
      <attr name="Source.Name"><string>plusY</string></attr>
      <attr name="Source.Region.Length"><int>3</int></attr>
      </node>

   \ldots

      <edge id="E1" from="N2" to="N1"><type xlink:href="Enclosing"/></edge>
      <edge id="E10" from="N13" to="N9"></type> xlink:href="Enclosing"/></edge↩
          >
   </graph>
</gxl>
```

Das Bauhaus Toolkit verzichtet daher auf die in der GXL vorgesehene Verschachtelung von Graphen und definiert die Strukturierung über Assoziationen, also Kanten. Bedingung dafür, dass ein Kantentyp als gliedernder Kantentyp taugt, ist, dass es sich bei dem durch diesen Kantentypen aufgespannten Graphen um einen Wurzelbaum handelt. In einem solchen sind alle Kanten gerichtet, er ist kreisfrei und es gibt einen eindeutigen Wurzelknoten. Es existiert zwischen zwei Knoten dieses Graphen nur ein Weg. Ein solcher Wurzelbaum mit n Knoten besitzt $n - 1$ Kanten.

Ein Graph ist dann kreisfrei, wenn für jeden seiner Knoten gilt, dass dieser entweder keine abgehenden (Nachfolger- oder Kind-)Kanten besitzt oder alle Nachfolger-Knoten kreisfrei sind.

Die die GXL-Datei des Bauhaus Toolkits muss also die Kreisfreiheit und die Wurzeleigenschaft geprüft werden. Dabei kann es je nach Struktur der geprüften Software sein, dass ein bestimmter Kantentyp im einen Fall einen Wurzelbaum darstellt, im anderen aber nicht. Lediglich bestimmte Kantentypen sind in einigen Programmiersprachen per definitionem eindeutig strukturierend, wie bspw. Klassenhierarchien.

2.4 Natural User Interface (NUI) [JG]

> Wer als Werkzeug nur einen Hammer hat, sieht in jedem Problem einen Nagel.
> — Paul Watzlawick

Wie Watzlawick so schön formuliert hat, machen die Werkzeuge, mit denen wir umgehen, aus, wie wir Probleme wahrnehmen. Bei der Einarbeitung in ein neues Tätigkeitsfeld lässt sich dies oft beobachten: Erst wenn die Werkzeuge bekannt sind, können die Probleme differenziert betrachtet werden und ein angemessener Lösungsansatz gewählt werden.

Doch was wäre, wenn man bereits alle Werkzeuge kennen würde und man nicht wieder neue lernen müsste? Mit dieser Frage beschäftigen sich die Designer von Interfaces schon seit Jahren. Ein Ansatz ist dieselbe Steuerung zu übernehmen, die ein anderes Produkt schon vorgelebt hat, welches eine breite Masse an Benutzern sich bereits antrainiert hat. Nun muss nur noch umdefiniert werden, was mit der Steuerung in der eigenen Anwendung erreicht werden kann. Weiter gedacht bedeutet dies, auf den bereits erworbenen Fähigkeiten des Nutzers, egal woher diese stammen aufzubauen. Mit anderen Worten: sich am alltäglichen Leben zu orientieren.

Stone, Earnshaw, Gigante und H. (1993) beschreiben dies sehr passend:

> An intuitive interface between man and machine is one which requires little training and proffers a working style most like that used by the human being to interact with environments and objects in his day-to-day life. In other words, the human interacts with elements of this task by looking, holding, manipulating, speaking, listening, and moving, using as many of his natural skills as are appropriate, or can reasonable be expected to be applied to a task.

Wenn der natürliche Lösungsansatz, der in den Sinn kommt, der richtige ist und dies im Zusammenhang mit einem Interface der Fall ist, spricht man von einem NUI. Für ein gutes Design müssen die Probleme so dargestellt werden, dass der Nutzer auch auf einen natürlichen Lösungsansatz kommen kann. Dass heißt beispielsweise, dass Dokumente in den Papierkorb geworfen werden können, aus welchem sie sich wieder herstellen lassen, bevor er geleert wurde. Dies ist eine typische Büro-Metapher übertragen auf die Nutzung von Computern.

Beim Design von Interaktionen für eine neue Anwendung bietet es sich an den Gedanken des NUI zu verfolgen. Insbesondere im Hinblick auf neue Technologien und Evaluationen, in denen Probanden nur begrenzt Zeit haben, ist eine möglichst schnell zu erlernende Steuerung ideal.

Im Falle unserer Arbeit handelt es sich bei der neuen Technologie um die der heutigen Head Mounted Displays. Die Interaktionen, welche gewährleistet werden müssen, sind Bewegung und Selektion. Es soll den Nutzern möglich sein, in dem Virtual Environment der Software City sich dreidimensional zu bewegen und einzelne Elemente der Software City-Metapher, also Häuser, Straßen und Verbindungen, anzuwählen um Informationen über diese zu erlangen. Im Kapitel 3.1 gehen wir auf unseren Interaktionsansatz ein.

2.4.1 Navigation und Orientierung

Dass die Interaktion für die Orientierung und insgesamt für die Wahrnehmung wichtig ist, zeigen Vion-Dury, Santana und Bull (1994). Sie fanden heraus, dass es möglich ist, durch Interaktionsmöglichkeiten in einer Visualisierung die Kontextwahrnehmung zu steigern. Dies bedeutet jedoch nicht, dass, wenn einmal eine geeignete Eingabeart gefunden scheint, diese universell gelten würde. Denn nach Rohr (2012) deutet vieles darauf hin, dass durch Interaktionsmöglichkeiten die Qualität und Effizienz einer Arbeit in (3D-)Visualisierungen gesteigert werden kann. Gleichzeitig merkt Rohr an, dass es wichtig ist, ein für die Dimensionalität geeignetes Eingabegerät zu haben (bspw. Maus für 2D).

Chance, Gaunet, Beall und Loomis (1998) fanden heraus, dass eine Bewegung, die ähnlich dem Gehen ist, bei der die Kopfbewegung die Blickrichtung kontrolliert, besonders effektiv in Hinsicht auf die Vermeidung von Motion- bzw. Cybersickness ist und eine bessere räumliche Orientierung bietet, als eine Steuerung mit Joystick oder einer hybriden Steuerung.

Auf diese Arbeit bauten Bowman, Davis, Hodges und Badre (1999) auf und fanden heraus, dass eine freie Navigation im dreidimensionalen Raum, die auch als Fliegen beschrieben werden kann, eine bessere räumliche Orientierung bietet, als eine vorgefertigte oder selbst geplante Strecke, die dann automatisch abgeflogen wird. Außerdem fanden sie heraus, dass nur Probanden die eine Strategie zur Orientierung besaßen, von einer freien Bewegung Vorteile hatten.

In die gleiche Richtung orientierung sich Ware, Hui und Franck (1993), welche anhand eines Pfadverfolgungproblems zeigen, dass sich die Anzahl der Fehlentscheidungen bei einer 3D-Aufgabe durch Interaktionsmöglichkeiten im Vergleich zu statischen Visualisierung ohne Interaktion halbiert.

2.4.1.0.1 Bewegungsform

In Mine (1995) wird für die Bewegung innerhalb von Virtual Environments zwischen Bewegungsformen mit konstanter Geschwindigkeit, konstanter Beschleunigung und über mit physischen oder virtuellen kontrollierter Geschwindigkeit unterschieden.

Konstante Geschwindigkeit und konstante Beschleunigung haben den Vorteil, dass sie leicht zu verstehen sind. Allerdings ist eine konstante Beschleunigung auch immer mit einem Bremsweg verbunden, der genauso lang wie der Beschleunigungsweg sein sollte.[86]

2.4.2 Auswahl

Für Auswahl-Aktionen bietet es sich ebenfalls an die Idee der NUI umzusetzen. Guan und Zheng (2008) erklären, dass die Zeigegeste mit dem Zeigefinger eine sehr natürliche Geste darstellt. Diese setzt jedoch voraus, dass die realen Hände vom System erkannt werden. Eine Alternative, die aus Präsentationen bekannt ist, ist die Verlängerung der Hand durch einen Laserpointer oder Stock.

Laserpointer als Anzeige einer Selektion oder zum Markieren wichtiger Punkte zu nutzen ist eine alltägliche Idee, wie sich auch daran zeigt, dass sie stets versucht wird weiter zu entwickeln.[87] Ein Problem der Zeigegeste ist, dass für Anfänger die Zeigerichtung in Hinsicht auf die eigene Orientierung[88] diese Geste zu Verwirrung führen kann.[89] Alternative Auswahloptionen nach Mine (1995) sind Fadenkreuze, welche sich auf zweidimensionalen Anzeigen anbieten, allerdings in Stereodisplays häufig wegen der Parallaxe der Augen schwer zu verorten sind.

[86] was ein genaues Stehenbleiben schwer macht, wenn der Nutzer die Beschleunigungsrichtung entscheiden und diese evtl. noch ein- oder ausschalten kann
[87] siehe u. a. Myers, Bhatnagar, Nichols, Peck, Kong, Miller und Long (2002) und Oh und Stuerzlinger (2002)
[88] siehe 2.1.3
[89] siehe Mine (1995)

Die echte Bewegung im Raum kann durch dynamische Skalierung für beliebig große virtuelle Räume verwendet werden. Auch kann die Blickrichtung und das damit verbundene Zentrum der Anzeige verwendet werde, was jedoch zur Folge hat, dass kein Umsehen ohne Auswahl möglich ist oder die Auswahl verzögert auftritt. Auch physikalische Kontrollen wie Joysticks, Gamepads, HTC Controller oder virtuelle Kontrollen (denen es dann jedoch (bislang) an Haptik mangelt). Auch Greifgesten und Berührungen sind zur Auswahl denkbar, erfordern jedoch, dass die auszuwählenden Objekte in lokaler Nähe sind.

2.4.3 Interaktion, Presence und Performance

Die Performance einer Anwendung hängt mit den Interaktionen und dem Wahrgenommenen zusammen. Im Kapitel 2.1.4 beschreiben wir die Komponente Place Illusion für Presence, die es dem Menschen durch ein natürliches Umgucken das Erfassen seiner Umwelt ermöglicht. Chance, Gaunet, Beall und Loomis (1998) zeigten, dass eine solche Art das Virtual Environment wahrzunehmen förderlich für räumliche Orientierung ist. Im Gegensatz dazu stehen die Aussagen aus Slater und Wilbur (1997) und B. Welch, Blackmon, Liu, Mellers und W. Stark (1996): Die weisen darauf hin, dass Presence nicht als Indikator für eine höhere Performance gilt, denn diese hängt in erster Linie von den Interaktionen und anderen immersiven Rahmenbedienungen ab, die zwar Presence fördern können, jedoch in einem viel direkteren Zusammenhang mit der Performance stehen.

2.5 3D-Entwicklungsumgebungen [MOR]

> You know, people are such snobs, with this 'oh, it's not about graphics' thing. That's such nonsense. It's totally about graphics.
> — Mark Rein, Epic Games Vizepräsident, Interview mit (GamesIndustry.biz 2005)

In den letzten Jahren erobern – forciert durch die steigende Leistung und raffinierteren Grafik-Algorithmen moderner Grafikkarten – immer wirklichkeitsnähere Spiele mit beeindruckenden 3D-Effekten dem Markt. Verbesserungen im Produktionsprozess und neue Techniken ermöglichen darüber hinaus Bildschirme in nie gekannten Größen zu erschwinglichen Preisen.

Zeitgleich mit der technischen Leistungssteigerung wurden von den Unternehmen der Game Industrys sog. Game Frameworks und Game Engines entwickelt, die die Erzeugung von Spielen

und deren Wartung vereinfachen soll. Viele dieser Game Engines wurden aufgrund Ihrer innewohnenden Marktvorteile als Unternehmensgeheimnisse eingestuft. In den letzten Jahren wurden als Pendant dazu frei verfügbare, z. T. sogar OpenSource Game Engine-Projekte veröffentlicht, die zwar in Ihrer Leistungsfähigkeit wahrscheinlich[90] nicht an die Profivarianten heranreichen, aber dennoch sehr beachtliche Ergebnisse erzeugen. Einen guten und relativ aktuellen Überblick über den Markt der Game Engines gibt [Wikipedia - Game Engines]. Diese Übersicht kann eine Hilfe bei der Auswahl einer Engine sein.

Game Engines sind aufgrund der Breite ihres Leistungsspektrums sehr umfangreiche Software-Bibliotheken, denen häufig auch eine eigene Logik zugrunde liegt. Je nach Game Engine gibt es auch eine Reihe von PlugIns, mit denen sich eine Game Engine um neue Funktionen, bspw. für neuartige Controller erweitern lässt. Der Umfang der Funktionsbibliotheken macht für Programmierer einen Umstieg von einer Engine auf eine andere schwer, so dass ein Umstieg genau bedacht werden muss.

Der essentielle Vorteil der Game Engines liegt darin, dass sie – z. T. sogar systemübergreifend – von den konkreten Eigenschaften der Hardware und der 3D-Programmierbibliotheken abstrahieren. Gerade die Erzeugung von komplexen Spielwelten mit vielen Mechanics und Dynamics Hunicke, Leblanc und Zubek 2004 wird dadurch entscheidend vereinfacht.

Unreal Engine
: wurde ab 1998 von Epic Games basierend auf C++ entwickelt; die Engine ist geeignet für Ego-Shooter-, Strategie- und Rennspiele, findet Verwendung u.a. von Electronic Arts und Ubisoft. Die Engine ist frei verfügbar bis zu einem gewissen Maximalumsatz beim Verkauf der erzeugten Spiele. Eingesetzt bspw. für *Batman: Arkham Asylum*, *BioShock*, *Gears of War*, *Medal of Honor: Airborne* und *Mortal Kombat vs. DC Universe*.

CryENGINE
: entwickelt von Crytec, besticht durch ihre Grafikfeature, wurde bspw. verwendet für: *Far Cry*, *Crysis* und *Crysis 2*.

Frostbite Engine
: eignet sich für First-Person-Shooter, wurde von Digital Illusions CE in erster Linie für die Battlefield-Reihe entwickelt, spielte aber in beinahe allen EA-Spielen eine Rolle. Unter anderem folgende Spiele basieren auf dieser Engine: Die *Battlefield*-Videospielreihe, *Star Wars: Battlefront*, *Need for Speed: The Run*, *Need for Speed: Rivals* und *Medal of Honor*.

Anvil Engine
: von Ubisoft fokussiert auf künstliche Intelligenz, wurde ursprünglich für den firmeninternen Gebrauch entwickelt. Die Videospielreihe *Assassin's Creed*, *Prince of Persia* und *Tom Clancy's Rainbow 6: Patriots* wurden mit der Anvil Engine entwickelt.

[90] Da es nur wenig verlässliche Information zum Stand der Technik in den Unternehmen der Game Industry gibt, können hier nur Vermutungen angestellt werden

Unity 3D Game Engine
war eine der ersten Game-Engines, die installationsfreie Spiele über Browser-Games ermöglichte und daher im Mobil-Game-Markt sehr stark verbreitet ist. Zu den bekanntesten Spielen, die mit dieser Engine entwickelt wurden, zählen *Battlestar Galactica Online*, *Gone Home*, *Hearthstone* sowie verschiedene Spiele für Mobilgeräte.

Source Game Engine
Source wurde von der Valve Corporation programmiert. Genutzt wurde sie u.a. für *Counter Strike: Source*, die *Portal*-Computerspielreihe, *Left 4 Dead*, *Left 4 Dead 2* und *Half-Life 2*

Quake Engine
, von id Software 1995 entwickelt, wurde bekannt durch das gleichnamige Spiel und steht heute im Verdacht, den Anstoß für alle heute auf dem Markt befindlichen Game-Engine gegeben zu haben. 3D-Gaming wurde erstmals mit der Quake Engine ermöglicht, daraus hervor gingen u.a. *Quake*, *Quake Arena* und *Half Life*.

RAGE
, entwickelt von Rockstar Games für die PC- und PlayStation-3-, Wii- und Xbox-360-Spiele, sollte die Herstellung von Spielen erleichtern. RAGE wurde u.a. in folgenden Spiele zum Einsatz: *Midnight Club: Los Angeles*, *Grand Theft Auto IV* und *Red Dead Redemption*.

HeroEngine
von Simutronics setzt ihren Schwerpunkt in MMO-Games und erhielt mehrere Auszeichnungen. Lizenz dieser Engine wurden von mehreren Firmen gekauft, um sie in der eigenen Engine einzusetzen. *Hero's Journey* und *Star Wars: The Old Republic* sind bekannte Spiele, die mit dieser Engine erzeugt wurden.

2.5.1 Auswahl der Entwicklungsumgebung [MOR]

Wir verfügen beide aus unserem Bachelorprojekt über Erfahrungen – auch in der Programmierung – mit der Unreal Engine 4.12 in Kombination mit dem LEAP Controller, der Oculus Rift und der HTC Vive, daher lag es für uns aufgrund der flachen Lernkurve dieser komplexen System nahe, die bekannten Systeme in unserer Arbeit weiter zu nutzen. Wir erarbeiteten unsere Software auf Grundlage der Unreal Engine 4.15.1 und der HTC Vive.

EPIC Games (und Digital Extremes), Hersteller des bekannten und umstrittenen Computer-Spiels „Unreal Tournament“, gab im Jahre 2009 erstmals die für bis dahin unter Verschluss bzw. Lizenz gehaltene Game Engine *Unreal Engine 3* für die Nutzung frei. Davor waren die *UnrealEd* Versionen, Editoren die bereits die UnrealSkript Sprache verwendeten, die einzige Möglichkeit für Modder Content für ihre Lieblingsspiele zu produzieren.

2.5.2 Unreal Engine [JG]

Die Unreal Engine 4 ist als Game Engine auf einen Workflow ausgelegt, der Spieleentwicklung fördern soll. Viele Funktionen der Engine waren für unsere Entwicklung nicht von Interesse, einige der grundlegenden Konzepte der Unreal Engine sind jedoch für das weitere Verständnis unserer Arbeit wichtig, dass wir sie im Folgenden kurz erläutern wollen.

In der Unreal Engine wird eine Welt, ein Karte, kurzum eine Virtual Environment als `Level` bezeichnet. Die grundlegende Instanz in der Unreal Engine ist ein `Object`. `Level` werden von `Actors` bevölkert. Diese sind in der Lage zur Laufzeit ein Verhalten aufzuweisen und eine *Transformation* besitzen: eine Position im Raum, eine Rotation und eine Skalierung. Genau genommen sind dies allerdings keine Fähigkeiten des `Actors`, sondern seiner `Components`. Es gibt zwei grundlegende `Components`-Arten: Zum einen die `ActorComponent`, die ein veränderliches Verhalten über die Zeit hinweg erlaubt und zum anderen die `SceneComponent`, welche ein `Transform` besitzt. Die `SceneComponent` welche die `RootComponent` eines `Actors` darstellt, ist für sein `Transform` zuständig. Eine Erweiterung der `SceneComponent` stellt die `PrimitiveComponent` da. Sie bringt die zur Anzeige nötigen Fähigkeiten mit sich. Eine `PrimitiveComponent` wird gerendert, kann über Kollisions- und Schattenwurfseinstellungen verfügen. Kurzum: Eine `PrimitiveComponent` stellt die Grundlage eines virtuellen Objektes dar, das gesehen werden kann und mit dem sichtbar interagiert werden könnte.

Die `Actors` und `Components` des `Levels` sind das Ergebnis der `ActorFactorys` und der `ComponentFactorys`, die aus den zugehörigen `Assets` erstellt werden.

Ein `Assets` ist eine Beschreibung für ein potentielles Spielobjekt welches im sogenannten `ContentBrowser` liegt. In ihm ist beschrieben wie ein entsprechendes Spielobjekt konstruiert werden soll und welches Verhalten, falls es denn eins hat, es haben soll. Ein `Asset` ist entweder `Blueprint`- oder C++-basiert. Wobei `Blueprint`-basierte `Assets` von C++-`Assets` erben können – andersherum geht dies nicht.

`Blueprints` sind eine graphische Programmiersprache, die innerhalb des Editors der Unreal Engine benutzt werden kann. Mit ihr können nur Spielobjekte programmiert werden. Sie sind speziell für Designer gedacht, die für Feineinstellungen und Gestaltung der `Level` zuständig sind.

Spielobjekte, die von Spielern oder dem Computer gesteuert werden sollen, heißen `Pawn`. Ein `Pawn` ist ein spezieller `Actor`, der den Spieler oder die künstliche Intelligenz, die ihn steuert, in der Spielwelt repräsentiert. Die Programmierung des `Pawns` entscheidet wie der Spieler seine Spielfigur steuern kann.

Neue `Assets` werden mit `Factorys` erzeugt. Diese werden auch beim Importieren angestoßen, wenn z.B. eine Textur in ein `TextureAsset` gewandelt wird, welches dann in Materialien verwendet werden kann.

In unserer Anwendung wurden vor allem GXL-Dateien importiert und visualisiert, wofür wir die nötigen `Factorys` erstellen und nutzen mussten.

Plugins

Wir haben uns für eine Game Engine entschieden, damit wir das Rad nicht neu erfinden müssen. Diesen Gedanken verfolgen auch die Unreal Engine-Plugins. Diese Erweiterungen enthalten entweder Einstellungsmöglichkeiten auf Editor- oder Package-Ebene, Optimierungen einzelner Features oder Spielobjekte und Funktionsbüchereien, um den Spieleentwicklern das Leben leichter zu machen. Auch wir haben uns dieser Plugin-Bibliotheken bedient.[91]

Performance

Ein weiterer wichtiger Punkt bei dem Umgang mit der Unreal Engine war in unserer Arbeit die Optimierung, um bei der Nutzung mit einem Head Mounted Display möglichst geringe Problem mit Motion- bzw. Cybersickness zu bekommen.[92] Daher wollen wir hier kurz erklären, wodurch die Performance bei der Unreal Engine begrenzt wird.

Unter Performance verstehen wir die Dauer, die die Berechnung eines neuen Bildes braucht: die Bildwiederholungsrate. Diese hängt immer vom langsamsten der drei Folgenden Aspekte ab: Berechnung des Spielverhaltens, Berechnungen der sichtbaren Objekte auf der CPU (sog. Draw-Calls) und Berechnung der Darstellung auf der GPU.

Eine Software City umfasst viele einzelne Elementen und diese sollen einzeln angesprochen werden können, um Informationen zum einzelnen Element zu gewinnen. Selbst bei verhältnismäßig kleinen Systemen, wie bspw. dem commons-launcher Modul aus org.apache, welches in gerade einmal 17 Dateien mit gerade einmal 1544 Lines-Of-Source-Code besitzt, produziert einen Software-Graphen der 82 Klassen und 355 Methoden umfasst.[93]. Wir wollen kurz auf die wichtigsten Kernproblem eingehen, auf die wir während unserer Arbeit gestoßen sind.

CPU

Um in der Unreal Engine in der Anzeige diese kleine Stadt anzuzeigen und jedes Element einzeln ansprechen zu können, müssen, wenn Methoden als Häusern gesetzt werden, allein für die Straßen und Häuser fast 500 Spielobjekte vom Typ `PrimitiveComponent` erzeugt werden. Dazu kommen dann noch die Verbindungen zwischen den Methoden (nur jene die von Haus zu Haus gehen)

[91] siehe Kapitel 3.2.4
[92] siehe Kapitel 2.1.7
[93] siehe Tabelle auf Seite 164

von denen es über 500 gibt und die sich, da sie gebündelt sein sollen, nicht naiv als ein einzelnes Spielobjekt realisieren lassen. Somit hat eine kleine Software City bereits über 1000 Spielobjekte, die alle gleichzeitig sichtbar sein sollen, damit der Nutzer sich einen Überblick verschaffen kann.

Für jedes dieser Objekte müssen einzelne Draw-Calls angestoßen werden, diese bestehen aus *culling* (wann ein Objekt nicht mehr angezeigt werden soll), *material setup* (welche Farbe jeder Pixel des Objekts haben soll), *lighting setup* (wie Licht behandelt und welche Schatten geworfen werden sollen), *collision* (wo es zu Überschneidungen kommt) und noch einigen weiteren. Zusätzlich müssen die Daten vorreitet, die Grafikpipeline freigeschaltet werden und die GPU für verschiedene Buffer vorbereitet werden.

Da die CPU i. d. R leistungsmäßig schlechter als die GPU ist und die GPU direkten Zugriff auf die Elemente Grafikkarte hat, ist eine Kernidee, die Draw-Call zu minimieren und die Arbeit an die GPU zu übertragen – soweit es geht. Der einzige Weg, die Draw-Calls zu senken, ist die Anzahl der `PrimitiveComponentss`, genauer: der Meshes, zu senken.

GPU

Relevante Gründe bei unserer Arbeit für hohe Kosten auf der GPU waren vor allem die Anzahl der Knoten aller Meshes und bei Experimenten mit manchen Materialien die Kosten des Texture-Mappings.

Die Anzahl der Knoten der Meshes lässt sich durch Vereinfachung der selbigen senken. Meshes, die zu wenige Knoten haben, lassen sich allerdings nicht gut deformieren, wie es für SplineMeshComponents nötig ist.[94]

Static vs. Dynamic Lighting

Ein wichtiger Aspekt bei der Orientierung ist die Wahrnehmung der Umgebung und die damit verbundene Einschätzung der Position wahrgenommener Objekte. Hierfür nutzt der Mensch verschiedene Indikatoren; einer davon ist der Schatten, den die Objekte werfen. Daher war es uns wichtig, diesen zu erhalten, auch wenn eine Stadt ohne Schatten wesentlich günstiger in der Berechnung ist.

Bei der Lichtberechnung in der Unreal Engine muss insbesondere zwischen *Static Lighting* und *Dynamic Lighting* unterschieden werden: Erstes es gedacht um für alle statischen Objekte eines *Levels* einen Schatten zu berechnen, der die ganze Spielzeit erhalten bleibt. Letzteres ist für veränderliche Objekte gedacht. Die statische Lichtberechnung ist deutlich 'günstiger' als die dynamische, dafür kann dynamische Lichtberechnung auch bewegliche Objekte und neu erschaffene Objekte mit einem angemessenen Schattenwurf versehen. Dies hat zur Folge, dass alle statischen

[94] mehr dazu in Kapitel 2.5.2

Objekte, die in einem Level angezeigt werden sollen, schon vor Spielbeginn existieren müssen, damit sie das *Static Lighting* nutzen können. Werden sie erst nach Spielbeginn erstellt, haben sie entweder keinen Schatten oder müssen auf *Dynamic Lighting* zurückgreifen.

Grafische Objekte für die Visualisierung von Kanten [MOR]

Da wir uns aus oben beschriebenen Gründen für den Einsatz einer Game Engine entschieden haben, haben wir für die Visualisierung von Kanten nicht die gleichen Freiheiten wie andere Autoren, die mit hardwarenahen Softwarebibliotheken wie OpenGL arbeiten: Wir müssen nutzen, was uns die Game Engine anbietet.

In Unreal Engine existieren nach unserer Recherche drei verschiedene grafische Objekte, die sich für die Visualisierung von Kanten eignen:

Cable Actor
: Ein Cable Actor ist eine kabelartige, grafische Verbindung zwischen zwei anderen Objekten oder Punkten im Raum. Der Cable Actor reagiert auf Gravitation, übt aber keine Kraft auf die Objekte aus, an die er ggfs. gebunden ist. Er kann mit anderen Objekten kollidieren. Seine Länge wird nicht absolut angegeben, sondern drückt seine Flexibilität aus. Ein Cable Actor besteht an sich aus einem kabelartigen Mesh, welches über die Länge des Kabels extrapoliert wird. Der Pfad des Cable Actors wird durch verschiedene Zwischenpunkte bestimmt, die zwar auslesbar jedoch nicht durch Programmierung manipulierbar sind.

Spline Component und Spline Mesh Component
: Eine Spline Component ist eine nicht-visuelle Komponente der Unreal Engine, die als Ausrichtung für Objekte wie bspw. Straßen, Seile, Ketten, Eisenbahnschienen, Zäune oder Felsketten dienen kann[95]. Auch können Spline Components als Pfad für Actors genutzt werden, beispielsweise für Flugbahnen von Vögeln oder Flugzeugen, Laufbahnen von Ameisen oder Flussbahnen eines Wasserlaufs. Der Pfad der Spline Component ist eine Catmull-Rom-Spline; ihr konkreter Verlauf wird durch Zwischenpunkte und Tagenten an diesen bestimmt.

Partikelsysteme mit Pfadverlauf
: Partikelsysteme dienen der Animation vielerei Effekte wie Feuer, Lauf, Regen, Schnee, Schweißbögen, Laserstrahlen und weiteres mehr. Unter einem Partikel wird dabei ein kleines Teilchen verstanden, das je nach Aufbau dies Systems auch durch Meshes oder Sprites ersetzt werden kann. In Unreal Engine kann sich die Bahn eines Partikel auch an einer Spline Component orientieren. Partikelsysteme werden in VR-Umgebungen genutzt, um bspw. Feuer, Laserstrahlen, Nebel oder Explosionen zu modellieren. Die Berechnung der Position der Partikel wird hauptsächlich von der Grafikkarte übernommen. Jedes Partikelchen wird einzeln berechnet; für eine vollständige Verbindung brauchen wir Hunderttausende solcher

[95] siehe Abbildung 2.18

Abbildung 2.18 Spline Component
Ein beliebiges Mesh kann entlang der Catmull-Rom-Spline verbogen und skaliert werden

Partikel: Die (sehr gute) Grafikkarte[96] stieß bei unseren Tests sehr schnell an Ihre Grenzen.

Heutige Hardware ist leistungsstark, die Game Engines optimieren die Darstellung, dennoch dürfen wir nicht vergessen, um welche Datenmenge es sich einerseits in einer typischen GXL-Datei handelt[97] und andererseits, um wie viel sich die Datenmenge durch den Einsatz von Meshes noch vergrößert. In Unreal Engine werden die meisten grafischen Darstellungen zur Unterstützung älterer Grafikkarten noch auf der CPU berechnet und die heute wesentlich leistungsstärkeren GPUs der Grafikkarte nur selten komplett ausgereizt; auch die Nutzung von Mehrprozessortechnik steckt noch in den Kinderschuhen.[98]

[96] siehe Tabelle 4.5
[97] siehe Tabelle auf Seite 164 für einen Einblick in die Verhältnis mäßig kleinen Systeme die wir dargestellt haben
[98] siehe Kapitel 2.5.2 für genauere Informationen zur Performance

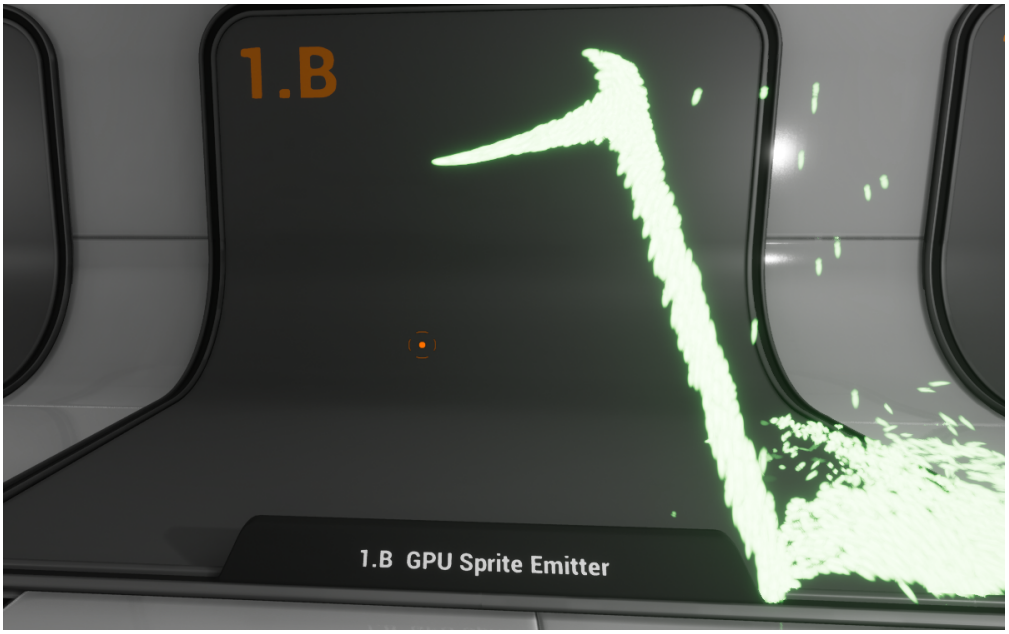


Abbildung 2.19 Partikelsystem mit Pfadverlauf

2.6 Evaluation [JG]

Zu unserer Aufgabenstellung gehörte eine Evaluation, für die wir unsere Software-City-Visualisierung – einmal am Desktop und zu anderen mit Hilfe eines Head Mounted Display dargestellt – vergleichen. Wir wollen im Folgenden kurz erläutern, was unter dem Begriff *Evaluation* verstanden wird, verschiedene grundlegende Typen der Evaluation und deren Einsatzgebiete vorstellen, sowie auf Besonderheiten der Evaluation im Zusammenhang mit Head Mounted Displays eingehen.

2.6.1 Begriffbestimmung

> Der Begriff Evaluation stammt von dem lateinischen Wort „valor“ (’Wert’) und der Vorsilbe e/ex, d.h. aus. Zusammen bedeutet dies “einen Wert aus etwas ziehen“, also eine Bewertung vornehmen. (Stockmann (2004))

Der Unterschied zwischen einer einfachen Bewertung und einer Evaluation im wissenschaftlichen Sinne ist, das hinter einer Evaluation ein Ziel steht. Es sollen eine oder mehrere Thesen überprüft und somit bestätigt oder widerlegt werden.

2.6.2 Evaluationstypen [JG]

Um eine Unterscheidung von Evaluationstypen vornehmen zu können, müssen wir uns klar machen, was alles Teil einer Evaluation ist. Wir haben als ersten den zu untersuchenden Gegenstand. Verändert sich an diesem etwas (wie in unserer Anwendung das Visualisierungssystem), handelt es sich um eine Vergleichsstudie.

Studien lassen sich auch anhand der erhobenen Daten unterscheiden. Sind es Einordnungen in Kategorien mit nicht klar definierbaren Abständen, wie z. B. in die Kategorien sehr gut, gut, weder noch, schlecht und sehr schlecht, handelt es sich um eine qualitative Studie. Sind die Daten jedoch eindeutig messbar, dann lassen sie sich quantifizieren, also ihrer Menge nach betrachten. Ein Beispiel für quantifizierbare Daten sind Zeitmessungen.

Es kann zwischen internen und externen Evaluationen unterschieden werden. Interne Evaluationen gehören der selben Organisation, die das Produkt entworfen hat, an. Das bedeutet, dass sie unter Umständen nicht über die ausreichenden methodischen Kompetenzen verfügen oder nicht ausreichend distanziert vom Produkt sind, um Alternativen zu bedenken. Externe hingegen gehören nicht der selben Organisation an, bringen dadurch Vor- und Nachteile mit sich. Sie sollten über profunde Kenntnisse in Methodik und Fachbereich verfügen und können als unabhängige Dritte eine geeignete Meinung für Verhandlungen bieten. Allerdings bringen sie auch Kosten mit sich und können weitere Auf- sowie Nachbereitungen der Evaluation erschweren, da immer der Umweg über die Externen nötig ist. Ein weiteres Manko kann sein, dass externe (durch ihre Fremdheit) die Probanden in Angst versetzen können.[99]

Auch der Zeitpunkt, zu dem eine Evaluation vollführt wird, ist entscheidend. Zum Beispiel kann zwischen Vorevaluationen und der eigentlichen Evaluation unterschieden werden. Die Vorevaluation hat als Zielsetzung nicht die Thesen zu untersuchen, sondern herauszufinden mit welchem Aufbau, die Thesen am besten untersucht werden können.

Als letzten Unterscheidungspunkt möchten wir die Probanden erwähnen. Bei ihnen lässt sich zwischen den direkt am Projektbeteiligten, Endnutzern und Experten unterscheiden. Ersteres führt zu einem Selbsttest, im zweiten Fall handelt es sich um eine Nutzerstudie und im dritten einen Expertentest, dass heißt Experten des Fachbereichs, die nicht selbst Nutzer sind.

2.6.3 Einsatzgebiete [JG]

Typische Einsatzgebiete qualitativer Studien sind die Verhaltensforschung oder die Marktforschung. In beiden ist oft der Aspekt menschlicher Wahrnehmung ein entscheidender Faktor und dieser lässt sich bis heute nicht quantifizieren. Die Lösung ist es also, die Probanden selbst um

[99] vgl. Stockmann (2004)

ihre Einschätzung zu bitten. Damit jedoch keine freien Antworten miteinander in Beziehung gesetzt werden müssen, werden Antwortmöglichkeiten vorgegeben. Wie bereits in Kapitel 2.6.2 erwähnt, handelt es sich dort meist um eine Kategorisierung. Ein weiteres Gebiet in dem keine quantitative Studie möglich ist, stellt die User Experience dar. In ihr interessiert das Erlebnis des Nutzers, welches höchst individuell ist.

Quantitative Studien haben ihr Anwendungsgebiet in direkter oder indirekter Konkurrenz zu anderen Produkten. Bei direkter Konkurrenz ist der Gegner bekannt, bei indirekter werden Daten erhoben, damit diese später einmal mit einem beliebigen Konkurrenten verglichen werden können. Ein Beispiel für solche indirekte Konkurrenzen sind Hardwareteile. Bei ihnen werden Daten erhoben und als Kennwerte zum Produkt hinzu geschrieben, die dann vom Kunden verglichen werden können. Damit dieser das am besten geeignete Produkt finden kann. Bei direkter Konkurrenz kann auch an Olympia gedacht werden, wo die Zeiten von Sportlern verglichen werden oder um ein Beispiel in der Informationstechnik zu nennen im der Laufzeit von Algorithmen. Im Zuge unserer Arbeit werden die Zeiten unserer Probanden gegeneinander gestellt.

Wir erheben in unserer Arbeit sowohl qualitative als auch quantitative Daten. Diesen Ansatz bezeichnet die Literatur als *Mixed Methods*[100]. Hierbei muss über den Zeitpunkt der verschiedenen Datenerhebungen und die Gewichtung nachgedacht werden (Creswell (2009)). Die qualitativen Daten sollen Aufschluss gegenüber prozessbezogenen Fragen bieten und die quantitativen bezüglich der Zielerreichung.[101]. Das Anwendungsgebiet dieses Ansatzes ist in der Überschneidung der oben genannten. Also wenn eine Konkurrenz in einer menschlichen Umgebung stattfindet, bei der nicht nur das Ergebnis zählt, sondern auch der Weg dorthin, insbesondere aus der Perspektive, dass nicht alle Probanden die selben Vorkenntnisse besitzen.

2.6.4 Evaluation mit Head Mounted Displays [JG]

Die Nutzung der eher unkonventionellen Darstellungsmethode via Head Mounted Display, auch wenn sie seit einigen Jahren nun der Öffentlichkeit zugänglich sind, mangelt es an Verbreitung, bringt einige Herausforderungen mit sich. Auf diese gehen wir im folgenden ein.

Die Nutzung von Head Mounted Displays hat in der Vergangenheit, bis heute ein Problem mit Cyber Sickness. Außerdem kann es Epilepsie-Anfälle auslösen. Deswegen ist es notwendig die Probanden über Gefahren der Nutzung eines Head Mounted Display aufzuklären und sie eine Einverständniserklärung unterschreiben zu lassen. Es kann passieren, dass Probanden nicht mehr gewertet werden können, weil sie die gestellten Aufgaben nicht bewerkstelligen konnten. Daher muss über den Zeitpunkt des Einsatzes von Head Mounted Displays nachgedacht werden.

[100] zu deutsch: Multimethodenansatz
[101] siehe Stockmann (2004)

Ein anderer Aspekt ist, dass die herkömmliche Steuerung mit Maus und Tastatur von den meisten als gewohnt betrachtet werden kann. In Bezug auf eine Bewegung im dreidimensionalen Raum ist dank vieler Programme, die eine solche Bewegung erfordern, eine bekannte Steuerungsmodalität erarbeitet, doch längst nicht jeder beherrscht diese. Dennoch bietet eine solche Etablierung die Möglichkeit diese zu nutzen. Im Bezug auf eine Steuerung mit Hilfe von Controllern, die nur zu einem einzigen Head Mounted Display Produkt gehören, gibt es solche Etablierungen nicht. Es muss also jedem Probanden eine ungewohnte Interaktion beigebracht werden. Selbst Probanden mit einiger Erfahrung in der Nutzung des Produkts muss die spezielle Steuerung neu erklärt werden, da in diesem Bereich zu jeder Anwendung wieder eine neue Steuerung gehört. Allerdings kann davon ausgegangen werden, dass die Steuerungen wenn sie nach dem Konzept der NUI entworfen wurden, leichter zu erlernen sind, als wenn die Navigation im dreidimensionalen Raum mit Maus und Tastatur erlernt werden muss.

Während der Proband bei einer Desktop Anwendung zum Versuchsleiter gucken kann und auch nonverbale Kommunikation stattfinden kann, fällt dies bei der Nutzung eines Head Mounted Display weg. Die Sicht auf die reale Welt ist soweit abgeschottet, dass es dem Probanden unmöglich ist nonverbale Kommunikation zu führen, außer durch direkte Berührungen. Welche jedoch eher vom Versuchsleiter, der den Probanden wenigstens sehen kann, ausgehen. Auch muss darauf geachtet werden, dass der Proband nicht mit Dingen aus der realen Welt in Berührung kommt, die den Versuch beeinträchtigen könnten oder den Probanden verletzten würden.

In vielen Versuchen mit Head Mounted Display spielt eine möglichst realistische Reaktion des Probanden auf die dargestellten Ereignisse und Gegenstände eine Rolle. In diesen Fällen muss ein besonderes Augenmerk darauf liegen ob der Proband Presence erfahren hat. Also ob er sich auf die dargebotene Illusion im Sinne der Plausibilität eingelassen hat und die validen Erkundungsaktionen ausgeführt hat. In einigen Fällen, insbesondere wenn nur die räumliche Vorstellung des Probanden eine Rolle spielt, lässt sich dies jedoch vereinfachen und es muss nicht extra getestet werden, ob der Proband ein Presence erfahren hat.

2.6.5 Evaluation von Softwarevisualisierungen [MOR]

In seinem Buch verwendet Diehl (2007) ein eigenes Kapitel auf das Thema quantitative und qualitative *Evaluation von Softwarevisualisierung*. Interessanterweise geht er vergleichsweise kurz auf quantitative Auswertungsmethoden ein. Diehl führt an, das qualitative Methode für die Bewertung von Softwarevisualisierungen von besonderer Bedeutung seien und bezieht sich dabei auf McConathy (1993), insbesondere auch, da solche Tests weniger Versuchspersonen benötigen und mehr Aspekte des System abdecken würden.

Diehl unterscheidet bezüglich der qualitativen Evaluation zwischen drei unterschiedlichen Foki:

Evaluationen basierend auf der Gestaltung[102], aufgabenbasierte sowie solche, die die kognitiven Dimensionen abfragen. Erstere prüft eine Visualisierung auf Konformität mit den akzeptierten gestalterischen Ansprüchen, zweitere fokussiert auf die Interpretation einer Aufgabenerledigung während letztere die Visualisierung bezüglich verschiedener kognitiver Anforderungen überprüft.[103]

Diehl (2007, S. 149) hebt als großes Problem bisheriger Evaluationen von Software-Visualisierung die Nutzung von generierten, nicht-realen Zufallsdaten. In solchen Datensätze werden z. T. Verbindungen erzeugt, die bei realen Daten nicht vorhanden wären. Dies ist einer der Gründe, weshalb wir für unsere Evaluation mit realen Daten arbeiten werden. Ein anderer ist eher persönlicher Natur: Wir wollen reale Software in unserem System sehen.

2.6.6 Eingesetzte Evaluationssysteme [MOR]

In dieser Arbeit setzen wir zwei unabhängige Fragebogensysteme zur Ermittlung der Usability unseres Systems ein. Wir wollen allerdings nicht tatsächlich die Usability ermitteln, da wir kein System auf Marktreife testen wollen, sondern wollen über die Fragenbogensysteme lediglich Antworten auf die in der Literatur[104] geäußerte Vermutung erhalten, dass die Kenntnis der Interaktionsmechanismen einen Einfluss auf die Bewertung eines Systems haben wird.

Wir arbeiten in dieser Arbeit mit dem meCue- sowie dem Software Usablity Scale-Fragebogen. Wir haben mit beiden Fragebögen bereits gute Erfahrungen gemacht.

2.6.7 meCue [MOR]

Der Name *meCue* ist eine Abkürzung für den englischen Titel „Modular Evaluation of key Components of User Experience„. meCue basiert auf dem analytischen Komponentenmodell des Nutzungserlebens nach , ist relativ jung, laut Internetseite[105] in einer Reihe von Studien getestet und für gut befunden worden.

führten das CUE-Modell ein, welches durch eine Kombination von Produktqualitäten, Emotionen und Nutzerkonsequenzen die Usability eines Systems bewertetet. gehen davon aus, dass diese drei Faktoren sich gegenseitig beeinflussen und damit auch das Gesamturteil des Nutzers entscheidend bestimmen. meCue ist die praktische Umsetzung des CUE-Modells in Form eines anpassbaren Fragenkatalogs.

[102] siehe Kapitel 2.2.2
[103] Diehl führt hier das *cognitive dimension framework* nach Green und Petre (1996) als Beispiel an. Diese definiert sechs grundlegende Aktionen und prüft deren kognitiven Dimensionen
[104] siehe Kapitel2.4
[105] *www.meCue.de*

Als Ergebnis gibt meCue Antwort auf die drei Untersuchungsbereiche Produktwahrnehmung, Emotionen zum Produkt und die Nutzerkonsequenzen sowie ein abschließendes Gesamturteil. Die Produktwahrnehmung teilt sich wiederum in aufgabenbezogenen Produktwahrnehmungen sowie nicht-aufgabenbezogene Produktwahrnehmungen. Im Fragebogen werden Fragen zu den Themenbereichen Nützlichkeit, Benutzbarkeit, visuelle Ästhetik, Status, Bindung, positive und negative Emotionen, Nutzungsintention und Produktloyalität gestellt. Auf Basis dieser Fragen lässt sich nach das Nutzererleben quantifizieren.

2.6.8 SUS [MOR]

Der Software Usablity Scale feierte gerade seinen dreißigsten Geburtstag und somit ein schon lange erprobter und genutzter Fragebogen. Brooke veröffentlichte 1996 ein Buch zu einem Usablity-Fragenbogen, welchen er bereits Jahre zuvor entwickelte. Dieser Fragebogen ist nach Brooke (2013) bewährt um eine schnelle Einschätzung der Usability eines System jeglicher Art zu erhalten. Dür SUS-Bewertungen werden mindestens 50 Probanden empfohlen, wir werden den Test dennoch anwenden.

Kapitel 3

Modellierung und Implementierung [MOR]

> I often describe the life of a software architect as a long and rapid succession of suboptimal design decisions taken partly in the dark.
> — Kruchten (2001)

In diesem Kapitel wollen wir unsere Modellierungs- und Implementierungsansätze vorstellen. Erst gehen wir auf unsere Modellierungen und die damit verbundenen Entscheidungen ein, danach im Kapitel Implementierung gehen wir abstrakt auf die Probleme und auf einige Designentscheidungen ein, die nicht direkt aus der Modellierung hervorgehen.

3.1 Modellierung [MOR]

Im Bereich der Modellierung gehen wir insbesondere auf die Datenstruktur ein, die wir zur Repräsentation der GXL-Daten nutzen, verdeutlichen unseren Ansatz zur Berechnung der Software-City-Struktur und erläutern die Strategie der Kantenbündelung, für die wir uns in dieser Arbeit entschieden haben.

3.1.1 GXL [MOR]

Im von der Bauhaus Toolkit erstellten GXL-Dateiformat werden nicht alle Optionen, die das Format bietet ausgeschöpft. Im Grund enthält eine solche Datei eine Liste von Knoten (Pakete, Klassen, Methoden und andere Einheiten eines Software-Sourcecodes) sowie Kanten (Assoziation verschiedenster Art zwischen Einheiten des Sourcecodes). Die Knoten und Kanten können jeweils

noch über Attribute verfügen, welche u. a. Namen, Positionen und Metrikwerte beinhalten. Wir haben daraus die in Abbildung 3.1 dargestellte Struktur in Unified Modeling Language-Notation modelliert.

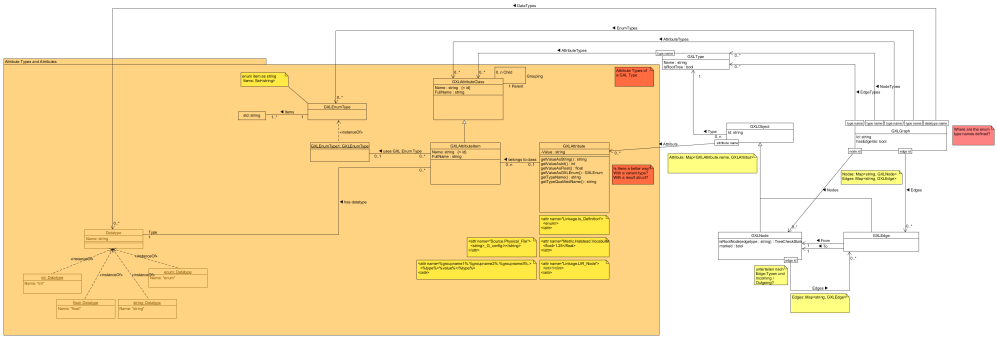


Abbildung 3.1 Das GXL-Format des Bauhaus Toolkit als UML-Diagramm
Das Diagramm ist unvollständig bzgl. der Methoden.

Im Laufe der weiteren Konzeption stellten wir fest, dass diese Datenstruktur eigentlich zu unflexibel für Abfragenvielfalt ist, die wir auf ihr ausführen wollen. Wir nahmen uns daher des Themas *Graph-Datenbanken* an und untersuchten insbesondere das Datenbanksystem *MongoDB*[1] intensiver, welches im Internet als System gepriesen wird, mit dem Graphen einfach zu verarbeiten wären, stellten aber fest, dass sich dieses *NoSQL*-System ebenso wenig für unsere Abfragen eignet wie jedes relationale Datenbanksystem.

Aus dem gleichen Grund beschäftigten wir uns mit XSLT, in der Hoffnung, dass wir die GXL-Datei direkt als Datei-Datenbank nutzen könnten und einfach direkt auf Ihr mittels XSLT Abfragen ausführen könnten. Zwar ging dies vom Grunde her, doch war der zeitliche Auswand für die korrekte Formulierung der Abfragen enorm, so dass wir von dieser Lösung Abstand nahmen. Letztendlich realisierten wir die GXL-Datenstruktur doch als In-Memory Pointer-Monster.

Unser Konzept aus diesem Daten eine Visualisierung folgt der Pipeline von Santos und Brodlie (2004). Wir reichern diese Daten in einer Graphtransformation um die nötigen Informationen für die Darstellung an, ordnen den Elementen eine Graphische Repräsentation zu und platzieren, rotieren und skalieren diese dann in unserer Virtual Environment in der sie angezeigt werden.

3.1.2 Knoten und Kanten [MOR]

Wir im Kapitel 2.5.2 beschrieben bietet die Unreal Engine verschiedene Elemente an, die sich für die Darstellung von Verbindungen grundsätzlich eignen würde. Durch langwierige Tests fanden wir heraus, dass wir nur mit der statischen SplineMeshComponent-Variante noch eine gewisse

[1] siehe *https://www.mongodb.com*

Anzahl Verbindungen bei guter Bildwiederholrate erhalten können. Die dynamischen Varianten verbrauchen offenbar soviel Kapazität, dass die Bildwiederholrate gegen 0 tendiert. Nachdem wir lange versucht haben durch Optimierung der SplineMeshComponents-Einstellungen die Performance zu verbessern, sind wir schließlich dazu übergegangen, die Meshes selbst zu erzeugen. Hierdurch haben wir zwar starre, unveränderliche Strukturen generiert, was unserem anfänglichen Ansatz widersprach, doch konnten wir so auch etwas größere Software-Städte mit Verbindungen generieren.

Die graphische Modellierung der Knoten viel dahingegen leicht. Für diese bietet sich das `EditorCube`-Mesh an.

Interessanter war die Modellierung der internen Informationen. Unsere Modellierung sieht vor, dass ein direkter Zusammenhang zwischen der Visualisierung und den GXL-Daten herrscht um diese Möglichst schnell im Spiel erreichen zu können, wenn diese abgefragt werden. Ob Verbindung oder Knoten, wir wollten eine klare Trennung zwischen Daten und der graphischen Anzeige. Dies ganze erweiterten wir für die Knoten noch einmal um eine Ebene, die die Position in der Stadt berechnet, bevor ohne diese direkt in die graphisch sichtbare Komponente einzutragen. Somit gibt es dort eine Ebene für die GXL-Daten, eine für die Daten unserer Berechnung der Anordnung und eine für die Anzeige.

Bezogen auf die Kanten haben wir es mit dem Klassischen Model-View-Controler Ansatz zu tun, bei dem der Pawn des Nutzers der Controller ist und bei den Knoten noch eine Ebene mehr um die Berechnung der Anordnung unabhängig von der gewählten Anzeige zu gestalten und die Daten trotzdem zu behalten. Um dieses Modell in der Unreal Engine umzusetzen, wollten wir alle Layout betreffenden Entscheidungen in `ActorComponents` auslagern, da diese ausschließlich ein Verhalten zur Erstellung der Ansicht im In-Editor Modus gedacht sind. Die Knoten und Kannten sollten als einzige vom Typ `PrimitiveComponent` sein, damit sie angezeigt werden.[2]

3.1.3 Software-City [JG]

Zu Beginn dieser Arbeit untersuchten wir die Domäne der Software Cities oder allgemeiner noch der 3D-Graphen-Visualisierung. Insbesondere in Hinsicht auf die Nutzung eines Head Mounted Display wollten wir uns nicht auf unsere Intuition verlassen, um einen sinnvollen Designentwurf zu machen.

Um uns zwischen den verschiedenen Ansätzen für Software Cities zu entscheiden machten wir einige Vorversuche.

Die erste Idee, die wir hatten ist in Abbildung 3.2 dokumentiert. Unsere Idee war es eine feste Grundfläche mithilfe einer hierarchischen Struktur, wie z.B. einer Paketstruktur aufzuteilen. Die

[2] siehe Kapitel 2.5.2

Klassen oder Methoden sollten, gemessen an 3 Metriken skaliert werden und an einer vierten sollte die Position auf der Grundfläche orientiert werden. In der Abbildung an den beiden Blöcken direkt auf dem gelben Grund zu sehen. Dieser Ansatz war angelehnt an die „Information Pyramid“ von Andrews[3].

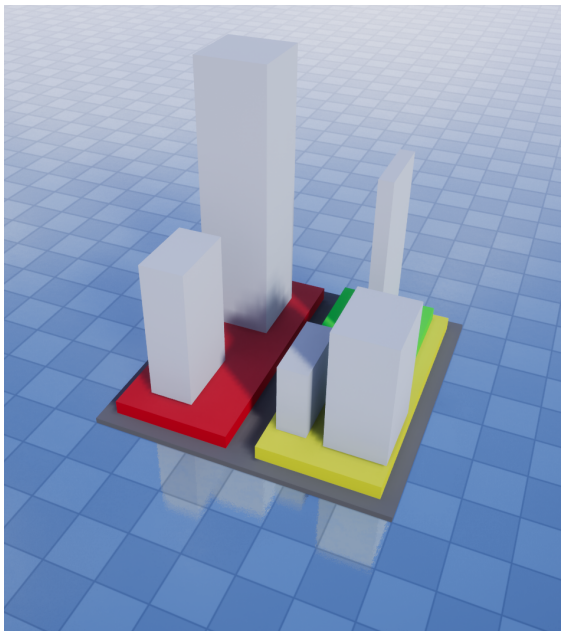
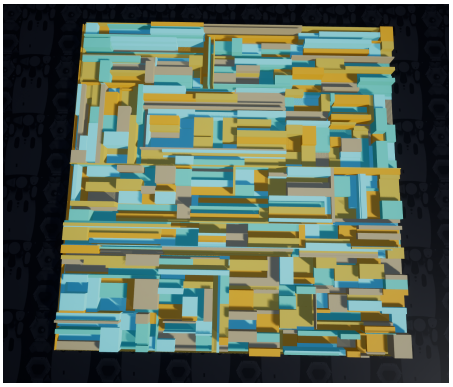


Abbildung 3.2 Veranschaulichung der ersten Anordnungsidee

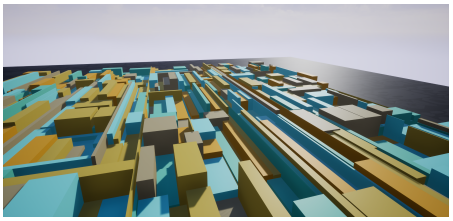
Unser Ansatz benutzte eine feste Grundfläche und würde diese allerdings nur begrenzt ausnutzen, da die Gebäude um vergleichbar zu sein in ihren Größen an die kleinsten angepasst werden müssten. Die Flächen unter den Gebäuden wollten wir wie beim *Slice&Dice*[4]-Algorithmus bei TreeMaps aufteilen. Dies führte uns zusammen mit den Aussagen von Steinbrückner[5] zu der Idee selbst ein TreeMaps-Visualisierung umzusetzen.

Unser zweiter Ansatz war eine 3D-Slice-And-Dice-TreeMap umzusetzen, siehe dazu Abbildung 3.3. Bei diesem Ansatz stießen wir jedoch auf mehrere Beobachtungen. Erstens ist das Virtual Environment so groß wie wir es wollen. Zweitens wird der TreeMap ein fester Platz ausgenutzt um auf diesem möglichst einen guten Überblick zu bieten, was für Desktop-Anwendungen bedeutet, die Bildschirmgröße möglichst ausnutzen zu können. Bei 3 Dimensionalen TreeMaps lässt sich dieses Ziel durch die möglichen Verdeckungen jedoch schon nicht mehr so gut erreichen. Es muss gedreht und in die Ansicht hinein manövriert werden. Wenn jedoch eine Navigation notwendig ist, warum dann nicht auch einen größeren Platz ausnutzen? Ein weiteres Manko aus unserer Perspektive war die schlechte Vergleichbarkeit der Höhen in unseren TreeMaps, durch die

[3] siehe Kaptiel 2.2.5
[4] siehe Shneiderman (1993, S. 314)
[5] siehe Kapitel 2.2.5



(a) Draufsicht



(b) Seitliche Perspektive

Abbildung 3.3 Eingefärbte TreeMap

vielen Verdeckungen war es schwer zu entscheiden ob zwei flachere Flächen die von zwei höheren getrennt wurden gleich hoch waren.

In unserem dritten Ansatz einer Software City entschlossen wir uns daher unsere Stadt räumlich stärker auszubreiten und somit den Platz des Virtual Environment voll auszunutzen. Auch war es uns wichtig, dass Lücken zwischen den Gebäuden entstehen um die Übersichtlichkeit zu erhöhen.

In der Literatur war der erste Ansatz, der Bottom-Up seinen Platzverbrauch kalkuliert, also mit anderen Worten wächst je mehr Knoten es gibt, war der EvoStreets Ansatz. Dieser erfüllte unsere Anforderungen, da sowohl Platz an den Straßen zwischen Häusern leicht einzubauen war, als auch frei Flächen zwischen den parallelen Straßen eingeplant waren.

Anordnung

Wir entschlossen uns eine eigene Anordnungsfunktion zu entwerfen die am EvoStreets Ansatz orientiert ist. Bei der Umsetzung dieses Algorithmus gingen wir nach folgendem Prinzip vor[6]:

1. Alles ist ein Block.
2. Blattknoten der Ansicht sind ein Block, dessen Breit und Tiefe durch Metriken bestimmt werden.
3. Ein Nicht-Blattknoten (Straßen- oder Viertelblock) ist so breit wie die Blöcke die beiden tiefsten Blöcke die links und rechts der Straße stehen sind zuzüglich der Straßenbreite und einem Abstand auf beiden Seiten. Die Tiefe bestimmt sich Anhand der Breiten der Blöcke der insgesamt längeren Straßenseite plus einem Abstand zwischen zu den Enden Straße und jeweils zwischen den Blöcken.

[6] welche in Abbildung 3.4 veranschaulicht wird

4. Kindblöcke werden erst auf der linken Straßenseite hinzugefügt und dann auf der kürzeren Straßenseite.
5. Kindblöcke werden um 90 Grad mit oder gegen den Uhrzeigersinn gedreht, sodass die Breite des Kindblocks zur Fassadenbreite wird und Kindblöcke die Straßen sind an ihrer Elternstraße mit ihren eigenen Kindblöcken beginnen.

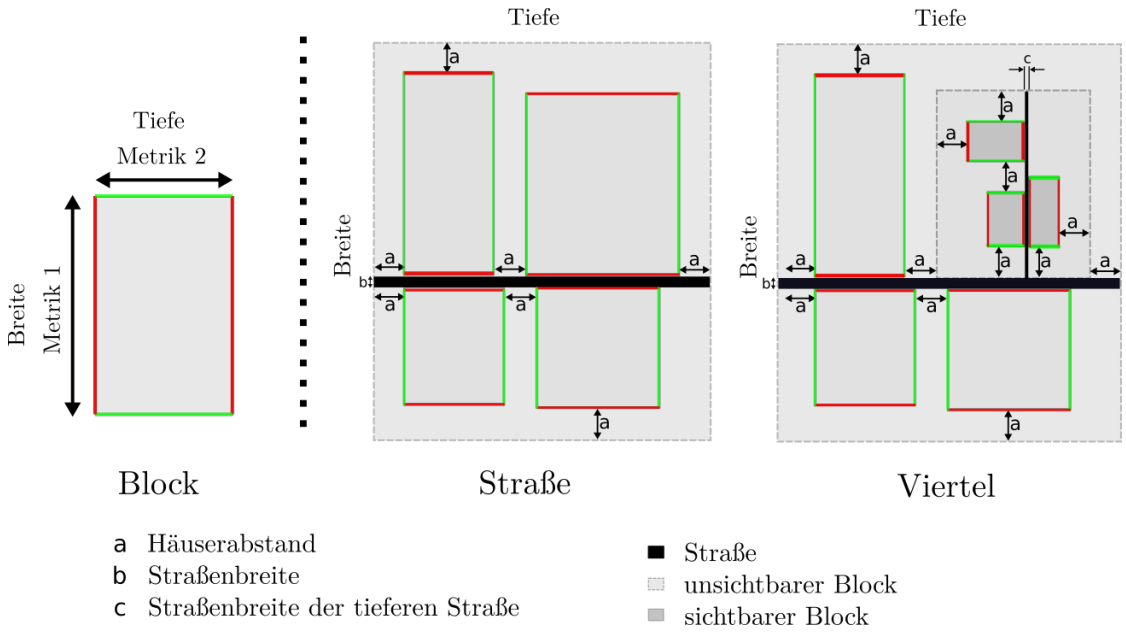


Abbildung 3.4 Veranschaulichung des Anordnungs-Algorithmus

Wir haben uns dafür entschieden die Metriken mit Straßen mit zu drehen, damit auf einen ganzen Straßenzug nur einmal eine mentale Rotation gemacht werden muss und nicht die Gebäude Ausmaße auf die globalen Richtungen zurückgedreht werden müssen. Somit lassen sich Metriken immer anhand der Straße interpretieren, an der das Haus steht. Dies eignet sich für lokale Perspektiven besser, macht aber eine globale Einschätzung beinahe unmöglich. Wir Argumentieren jedoch dafür, dass für eine globale Perspektive, die Breite und Tiefe eines Hauses ungeeignet sind, da zu viel Verdeckung statt findet. Dies relativiert sich erst bei einer Vogelperspektive, bei der man jedoch keine dreidimensionale Stadt benötigen würde. Eine 2D Stadt würde für diesen Zweck voll und ganz ausreichen.

In Abbildung 3.5 is zu sehen, wie der Algorithmus angewandt auf unser Demoprojekt aussehen soll.

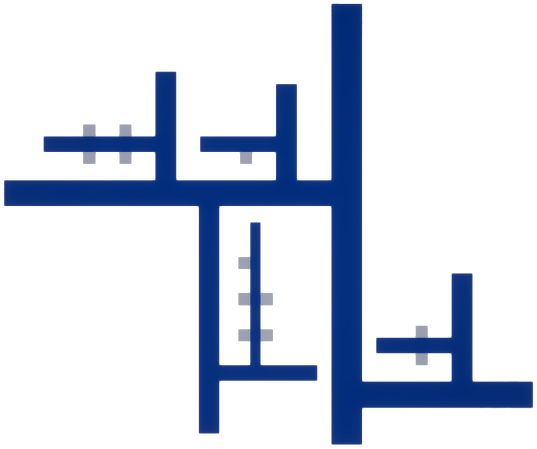


Abbildung 3.5 Anwendung des Algorithmus auf unser Demoprojekt aus Listing 2.5

3.1.4 Kantenbündelung [MOR]

Wir haben uns für die Realisierung einer hierarchischen Kantenbündelung entschieden, da uns nach dem Studium des Stands der Forschung[7] dies als der plausibelste Ansatz erschien. Durch die hierarchische Strukturierung – so unser Gedankengang, ließe sich auch in den Kanten die hierarchische Zuordnung der verbundenen Knoten nachvollziehen.

Konkret führen in unserer Modellierung alle Kanten – beginnend von beiden Seiten an den assoziierten Elementen – über den Mittelpunkt der Straße, an der die jeweiligen Element liegen und dann rekursiv aufsteigend über die Mittelpunkte der übergeordneten Straßen bis zu der Straße, die beide Elemente gemeinsam haben. Die vertikale Positionierung wird in jedem Rekursionsschritt höher gesetzt. Das Ergebnis auf Basis des Demoprojekts ist in Abbildung 3.6 zusehen.

Wir hatten darüber nachgedacht, die Kanten nicht alle genau in einem Punkt zu bündeln, sondern leicht zu versetzen, haben uns jedoch dagegen entschieden, da die Kanten eine gewisse Stärke nicht unterschreiten sollten um noch gut sichtbar zu sein. Wir stellten kleineren Vortests fest, dass Kanten entsprechender Stärke nebeneinander durch den Schnittpunkt zu führen, Platz benötigt und das Modell unübersichtlicher macht.

3.1.5 Realisierung in der Unreal Engine [MOR]

Wir haben uns dafür entschieden eine beleuchtete Szene mit `Skylight` und einer statischen Lichtquelle zu nutzen um einen statischen Schatten für die Wahrnehmung zu erhalten. Schatten sind

[7] siehe Kapitel 2.2.6

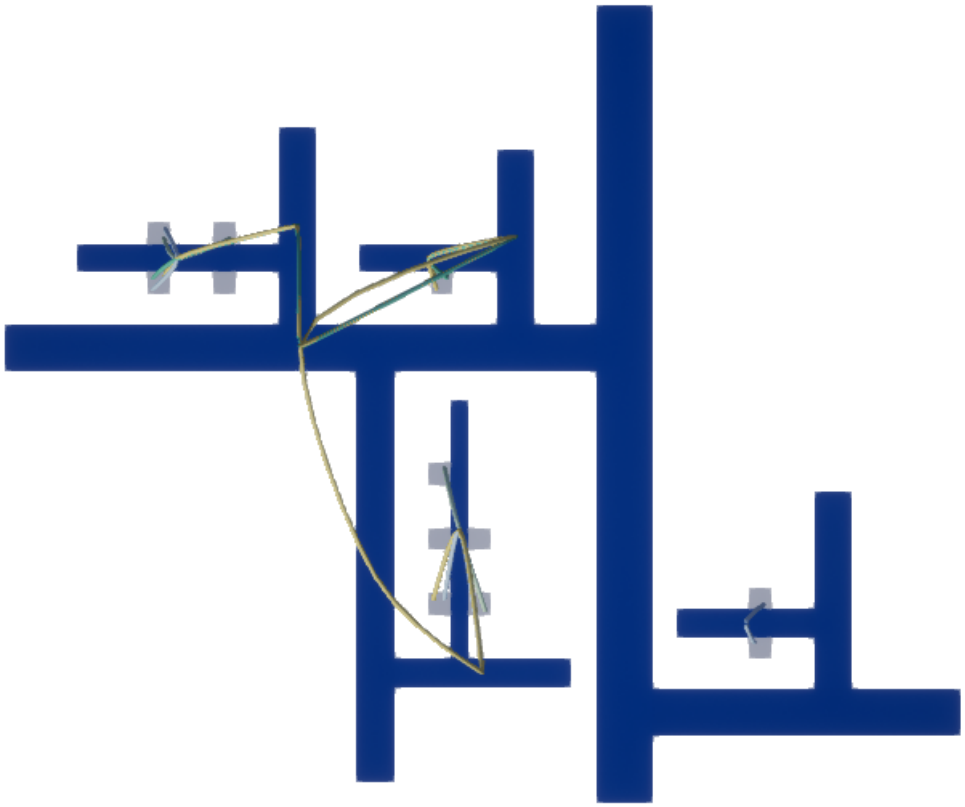


Abbildung 3.6 Gebündelte Kanten im Demoprojekt

wie in Kapitel 2.1.3 bereits erwähnt ein Indikator für die Tiefenwahrnehmung. Diese hilft dabei auch auf große Distanzen, auf denen stereoskopisches Sehen nicht mehr funktioniert, welche bei Software City vorkommen, die Distanzen einzuschätzen und damit eine genauere räumliche Vorstellung zu bekommen. Das `Skylight` haben wir hineingebracht um die Schatten etwas aufzuweichen, damit eine farbliche Zuordnung der Kanten von der Schattenseite möglich ist. Mit einer Lichtquelle sind die Schatten sehr hart. Obwohl unbeleuchtete Materialien günstiger bei der Berechnung sind, haben wir uns daher gegen sie entschieden.

Um der ersten Komponente der räumlichen Vorstellung[8] der räumlichen Wahrnehmung die nötigen Informationen zu bieten, entschlossenen wir uns dafür den `Atmospharic Fog` zu verwenden. Denn dieser gibt dem Virtual Environment einen Horizont.

Um bei der Nutzung eines Head Mounted Display eine möglichst hohe Immersion zu bieten, ist es notwendig die möglichen Erkundungsaktionen so gut wie möglich zu gestalten. Durch das Tracking der HTC Vive ist auf sensorischer Ebene, eine gute Grundlage für Erkundungsaktionen gegeben. Seitwärts Bewegungen, drehen auf und ab, werden meist richtig und flüssig erkannt. Auf der bildgebenden Seite ist es etwas schwieriger. Aus der Wahrnehmungsforschung ist bekannt, dass bei ca. 30 Bildern pro Sekunde die menschliche Wahrnehmung ein flüssiges Bild erzeugt. Dies gilt allerdings nur für statische Displays. Bei einer schnellen Kopfdrehung nimmt der Mensch auf einem Display, dass nur 30 Bilder pro Sekunde anzeigt schwarze Flecken am Rand wahr. Diese zerstören die Place Illusion und sind ein Auslöser für Cyber Sickness, denn in der Realität passiert dies im gesunden Zustand nicht. Daher ist es uns wichtig eine möglichst hohe Framerate zu halten. Eine so hohe Performance jedoch zu gewährleisten, stellt bei großen Softwaresystemen eine erhebliche Schwierigkeit dar. Nicht umsonst werden in Spielen die Levelabschnitte möglichst klein gehalten, durch die kleineren Bereiche kann zu jedem Zeitpunkt des aktiven Spielens ein flüssiges Spielerlebniss gewährleistet werden. Da bei der Software City allerdings ein Überblick über das gesamte System möglich sein soll, konnten wir diesen Weg nicht einschlagen.

3.1.6 Modellierung Interaktion [JG]

Wie bereits beschrieben, war Teil unserer Aufgabe, ein herkömmliches Maus-Tastatur-Bildschirm-System gegen ein System mit Head Mounted Display zu evaluieren. Neben der unterschiedlichen Darstellung und den damit verbundenen Effekten[9] stellt sich auch die Frage nach der Interaktion. Wie bereits kurz dargestellt, sind die Freiheitsgrade der Interaktion mit einem Head Mounted Display – auch durch neuartige Controller – weitaus höher, wohingegen die Maus-Tastatur-Variante den Vorteil der längeren Gewöhnung genießt. Der Interaktion mit einem Head Mounted Display wird nachgesagt, sie sei direkter und natürlicher[10], während sowohl Maus als

[8] siehe Kapitel 2.1.3
[9] siehe Kapitel 2.1
[10] siehe Kapitel 2.4

auch Tastatur nur indirekte Interaktion ermöglicht. Gleichwohl sind die Haupteinsatzgebiete der beiden Technologien bislang unterschiedlich, so werden Head Mounted Display in der Visualisierung und Forschung verwendet wohin gegen Maus und Tastatur auch in Arbeitszusammenhängen speziell im Büro eingesetzt werden; eine Übertragung in andere Anwendungsbereiche wird diese polarisierenden Aussagen relativieren.

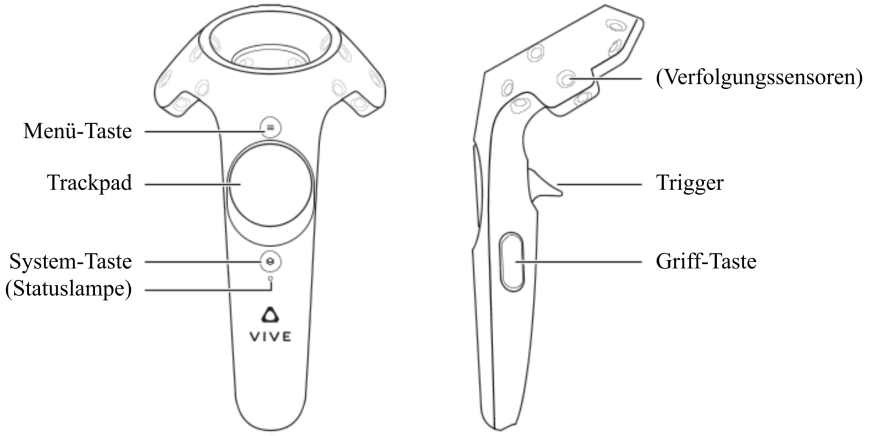


Abbildung 3.7 Interaktionsmöglichkeiten eines Controllers der HTC Vive (Quelle: nach HTC (*Über die Vive Controller*))

Um die Ergebnisse unserer Versuche vergleichbar zu halten, wollten wir im Vorfeld die möglichen Interaktionen mit unserem System festlegen. Um die Aufgabenstellung erledigen zu können sind folgende Interaktion mit einer dreidimensionalen Darstellung notwendig: Umsehen, Zoomen (Heranholen, Vergrößern), Auswählen von Objekten, Darstellung von Informationen, Verfolgen von Verbindungen. Wobei wir Zoomen und das Verfolgen von Verbindungen zu Navigation im dreidimensionalen Raum verbinden.

Maus und Tastatur

Bei der Maus und Tastatursteuerung entschlossen wir uns aus der Perspektive der schnelleren Erlernbarkeit, an etablierten Steuerungen in (pseudo-)dreidimensionalen Virtual Environments anzuknüpfen. Wir bedienten uns bei Steuerungen aus 3D Bearbeitungsprogrammen wie Blender, Maya und nicht zuletzt der Unreal Engine selbst. Da diese Steuerung zum größten Teil selbst von den Steuerungen der FPS Spiele übernommen wurde, erlaubt dies uns den Trainingseffekt der Nutzer die mit solchen Spielen Erfahrungen haben ebenfalls abzugreifen.

Somit kamen wir bei der Steuerung der Bewegung mit WASD erweitert um Q und E, wie in der 3D Bearbeitungsprogrammen üblich an. Das Umgucken wurde der Mausbewegung zugeordnet, mit der Einschränkung, das der Blickwinkel sich nur anpasst, wenn entweder die rechte oder

linke Maustaste gedrückt ist. Die Selektion legten wir auf das gedrückt halten mit der linken Maustaste.

Head Mounted Display

Für das Umgucken mit dem Head Mounted Display war für uns klar, dass die Kopfbewegung genutzt werden soll. Chance, Gaunet, Beall und Loomis[11] bestätigte uns in dieser Wahl.

Für die anderen Interaktionen ist eine Steuerung mit den Händen nötig. Aus unserer Vorerfahrung mit einer Handgestensteuerung auf dem aktuellen Stand der Technik wissen wir, dass es dort oft Schwierigkeiten in der Genauigkeit bei der Erkennung gibt. Daher entschieden wir uns für die konservativere Lösung mit Controllern die restlichen Interaktionen umzusetzen. Da die Controller der HTC Vive ebenfalls von den Kameras erfasst werden und in das Virtual Environment übertragen werden, schienen sie uns als geeignete Lösung. Auch erlauben sie eine unabhängige Bewegung der linken und rechten Hand.

Die Bewegung innerhalb einer Software City konnten wir uns am ehesten Fliegend vorstellen.[12] In Diskussionen kam auch die Idee sich durch die Stadt zu Teleportieren auf, da jedoch vom Prüfer gefordert wurde auch Übersichtsperspektiven wie eine Draufsicht einnehmen zu können. Für die Orientierung wäre es nach Literaturlage noch besser, wenn die Bewegung in dem Virtual Environment durch tatsächliches Gehen in der Realität verursacht würde, jedoch haben wir uns wegen unseres kleinen Bereichs in dem sich die Probanden bewegen können, dagegen entschieden. Eine Begründung die sich auch in der Literatur als Problem angesprochen wurde. Für die Kontrolle der Flugrichtung orientierten wir uns an Mine[13] in dem wir diese mit einem der Controller angeben lassen. Es wird dahin geflogen wohin der Controller zeigt, das erlaubt eine unabhängige Kopfbewegung von der eigentlichen Bewegung was uns sehr wichtig war. Über eine Taste wollten wir die Bewegung dann entweder Vorwärts in diese Richtung oder Rückwärts auf einer anderen aktivieren lassen.

Für die Auswahl haben wir uns damit wir einen geeigneten Vergleich zur Maus und Tastatur haben für eine die auf Entfernung funktioniert. Griff oder Berührungsmetaphern zur Auswahl erfordern eine Wahrnehmung über etwas das Überlappen kann, dafür hätten wir der Maus und Tastatur Steuerung einen Avatar geben müssen der First-Person gesteuert werden kann. Dies erscheint uns als eine unsinnige Interaktionsmethode in Hinsicht auf Software bezogene Aufgaben. Stattdessen haben wir uns entschlossen eine Auswahl zu nehmen die wie bei dem anklicken dessen was sich unter dem Cursor befindet, auf ein gewisse Reichweite funktioniert, nämlich das Deuten aufs Zielobjekt. Zum Deuten wird wie bei einem Zeigestab oder Laserpointer der Controller

[11] siehe Kapitel 2.4.1
[12] wie als Kind über der Bauklotzstadt oder auf einem fliegenden Teppich
[13] und Superman

verwendet, da wir dies eine natürliche Aktion für viele Menschen ist.[14]

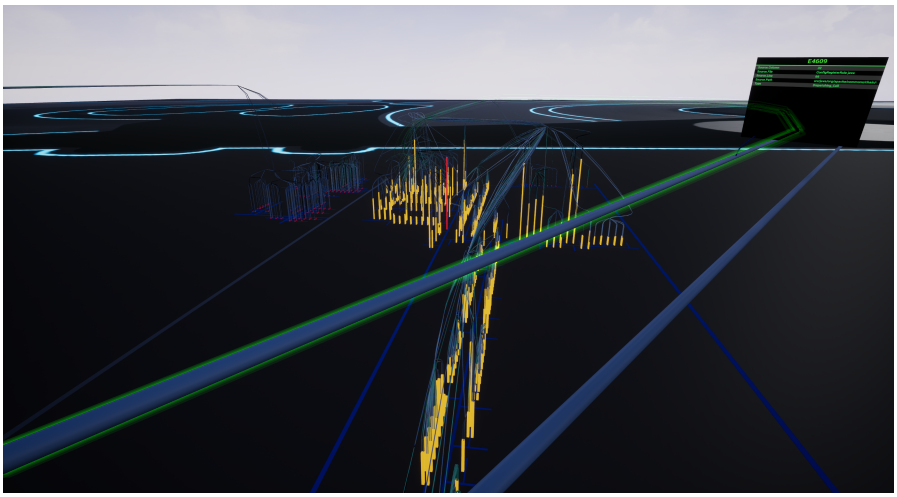


Abbildung 3.8 Eine selektierte Verbindung

Reichweite bei der Auswahl

Um die Selektion im Sinne der Eindeutigkeit zu vereinfachen entschlossen wir uns eine maximale Reichweite einzuführen, auf die eine Auswahl noch möglich ist. Dies sollte auch dem Zwecke dienen einen Ausgleich für die niedrigere Auflösung mit der HTC Vive zu schaffen, da sonst mit der Maus und Tastatur eine deutlich genauerer Auswahl möglich wäre. Ein weiterer positiver Aspekt der eingeschränkten Reichweite ist es, dass die Probanden dazu gezwungen werden sich zu bewegen um einige Elemente auswählen zu können. Dadurch wird sicher gestellte, dass sie automatisch ihre räumliche Vorstellung anpassen[15] und somit die Gelegenheit bekommen die Kernaufgabe zu bewältigen.[16]

Es war uns wichtig den Nutzern ein Feedback zu ihren Handlungen zu geben. Abbildung 3.8 zeigt wie eine markierte Verbindung aussehen könnte, wohin gegen die selbe Markierungsform auf einem Knoten in Abbildung 3.9 zu sehen ist. Wir haben mit verschiedenen Markierungsformen experimentiert, eine war die Farbe des markierten Elements auf eine sonst in der Visualisierung nicht vorkommende zu ändern, andere war die außen Kanten nachzumalen und durchleuchtend zu machen. Der Ansatz in den Abbildungen ist eine Erweiterung des letzten bei dem die Markierung sich in der Ferne auffächert, sodass sie immer dieselbe Fläche bedeckt, jedoch in der Ferne schwerer zu erkennen wird. Dies hatte für unsere Aufgabenstellung den Effekt, dass die

[14] siehe Kapitel 2.4.2; auch das Deuten mit der Fernbedienung auf den Fernseher geht in die selbe Richtung
[15] siehe Kapitel 2.1.3
[16] siehe Kaptiel 4.1

Verbindung leichter wieder zu finden war, jedoch nicht bis in große Ferne allein mit den Augen verfolgt werden konnte.

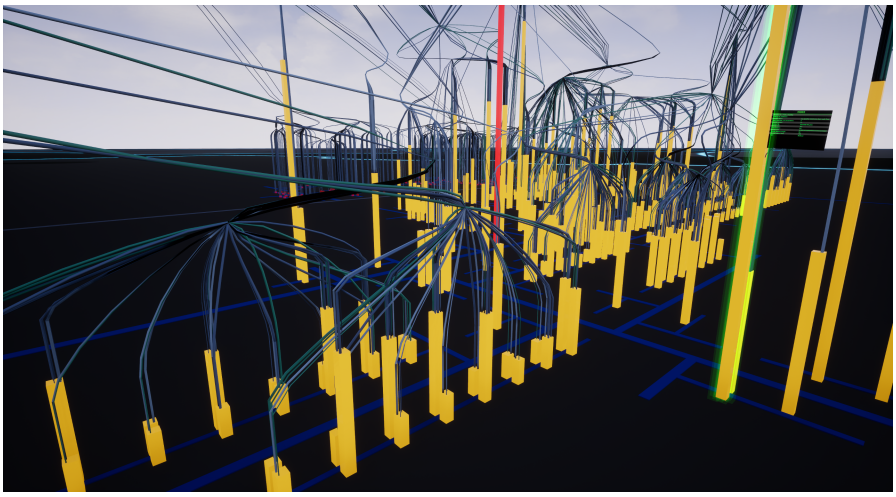


Abbildung 3.9 Ein selektierter Knoten

Geschwindigkeit

Bei der Entscheidung des Flugmodus orientierten wir uns an Mine, der auf verschiedene Bewegunsmodi eingeht.[17] Um eine möglichst einfache Steuerung zu erhalten beließen wir es bei einer konstanten Geschwindigkeit für den Flugmodus. Aus persönlicher Erfahrung mit einigen SteamVR Spielen in den z.B. konstante Beschleunigung auf Knopfdruck als Bewegungsmodus verwendet wurde, konnten wir sagen, dass solch eine Steuerung lange in der Erlernung braucht. Eine über ein physikalisches oder virtuelles Gerät gesteuert (abgesehen von an und aus) hielten wir ebenfalls für zu Aufwendig in der Anlernphase.

Resultat unserer Entscheidung ist, eine etwas unpräzise Steuerung, daher haben wir bei dem Design der Aufgaben darauf geachtet keine Präzise Bewegung zu fordern, sondern stattdessen die Positionierung in einem gewissen Rahmen offen zu lassen.

[17] Siehe Kapitel 2.4.1.0.1

3.2 Implementierung [MOR]

> Der Mensch hat dreierlei Wege klug zu handeln:
> Erstens durch Nachdenken, das ist der edelste,
> zweitens durch Nachahmen, das ist der leichteste,
> und drittens durch Erfahrung, das ist der bitterste.
> —Konfuzius (551 v. Chr. – 479 v. Chr.)

Üblicherweise – so steht in Lehrbüchern zum Vorgehen in Softwareentwicklung geschrieben – erhebt man zunächst die Anforderungen, konzipiert dann das zukünftige System und implementiert am Ende nur noch schnell die Konzepte. Auch wenn diese verkürzte Darstellung im Grunde dem heute als veraltet angesehene Wasserfallmodell entspricht, so wird es dennoch immer noch als Grundlage idealer Softwareentwicklung verstanden. Konzipieren mit dem Ziel der Implementierung kann man nur, wenn das System, in dem implementiert werden soll, bekannt ist. Doch für ein unbekanntes System lassen sich nur schwer Details konzipieren, die über allgemeingültige Informatikerweisheiten hinaus gehen.

In unserem Fall war das Implementierungssystem – hier: die Unreal Engine – unbekannt, oder genauer: Wir hatten beide in einem anderen Projekt rd. 4 Monate Kenntnisse in der Programmierung der Unreal Engine erworben, von denen wir nicht ahnten, wie oberflächlich sie letztlich doch waren und vielleicht sogar immer noch sind. Dies war auch einer der Gründe weshalb die anfängliche hehre Idee, in Ruhe Konzepte für die Stadtvisualisierung sowie die Bündelung der Verbindungen, dann für das Gesamtsystem zu konzipieren und schließlich zu implementieren einer leichten Panik wich, nachdem wir feststellten das C++ und Unreal Engine-C++[18] außer dem Namen und einige grundlegenden Konzepten nicht viel gemeinsam haben: Unreal Engine-C++ ist quasi (C++)++; die Unreal Engine implementiert bereits seit Jahren einige der Features die für C++ erst für das nächste Release 2018 vorgesehen sind, in anderen Programmiersprache seit Jahrzehnten schon Realität sind. Hierzu gehört eine Garbage-Collection, Modulstrukturen, ein Runtime-Type-System und Meta-Klassen.

Diese Vorteile erkauft man sich in der Unreal Engine durch ein Programmierparadigmen, Makro-Strukturen und Precompiler, die den eigentlichen Programmcode zunächst in einer *intermediate*-Version übersetzen, in denen die gesamte 'Magie' vonstatten geht. Damit das alles wie gedacht funktioniert, ist der Programmierer angehalten, sich sehr genau an die vorgegebenen Makro-Positionen und -Verwendungen, spezielle Benennungen von speziellen Elementen sowie die Verwendungen von Makroparametern, die bspw. Sichtbarkeit, Wertbeschränkungen oder Persistenz der zugehörigen C++-Objekte bestimmen, zu halten.[19]

[18] Unreal Engine-C++ ist keine offizielle Bezeichnung.
[19] Von außen betrachtet wäre es einfacher gewesen, auf C++ zu verzichten und statt dessen eine dem allgemeinen Standard besser entsprechende Programmiersprache zu nutzen. Wir denken, dass die Verwendung von C++ als

Insofern gingen wir den bitteren Weg des Konfuzius. Unsere Arbeit bedient sich vieler Domänen, die wir uns in vielen kleinen Schritten, unter zu Hilfenahme vieler Testprojekte, erschließen mussten. Leider gibt es weder online noch in Buchform eine aus unserer Sicht wirklich gute Anleitung zur den Features der Unreal Engine-API. Online ist zwar eine mehr oder weniger ausführliche Beschreibung von Modulen, Klassen, Attributen und Operationen zu finden, doch wie diese Elemente zusammenspielen und welche Klassen man für welchen Zwecke mit welchen anderen zu kombinieren hat, ist nur für sehr grundlegende Dinge beschrieben. Letztendlich haben wir viele der Implementierungen, die wir in unser System eingebaut haben, aus dem Sourcecode der Unreal Engine, der frei zum Download verfügbar ist, und Erweiterungen anderer Programmierer sowie Demoprojekten, die über den Marktplatz der Unreal Engine zu erhalten sind, 'herausoperiert' und für unsere Zwecke umgebaut. Leider sind wir an vielen Stellen weit von einem Verständnis dessen entfernt, was die Unreal Engine konkret macht und ob wir überhaupt alles berücksichtigt haben – von validen Testfällen ganz zu schweigen.

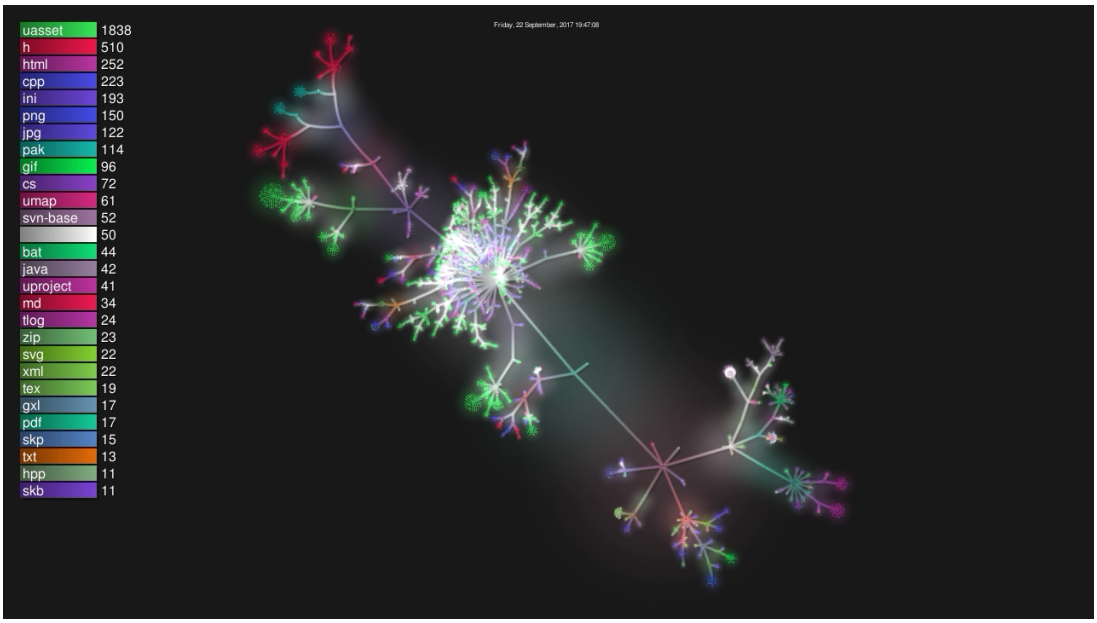


Abbildung 3.10 Projektübersicht Modellierung & Implementierung; Stand: 13.10.2017

Basissprache einerseits historische Gründe in der Entwicklung der Unreal Engine hat, andererseits sind viele Grafikbibliotheken in C oder C++ gehalten.

3.2.1 Knoten und Kanten [JG]

> Ein schlechter Programmierer ist man vor allem, weil man schlecht programmiert.
> — Passig und Jander (2013, S. XV)

Die Implementierung der Knoten verlief weitestgehend nach unserer Modellierung[20]. Es wurden Einstellungsmöglichkeiten in die GUI übertragen in denen das Layout für die Knoten angepasst werden kann. Auf diese Weise erlauben wir eine dynamische Zuordnung der grafischen Repräsentation im Zuge der in Kapitel 3.1.1 beschriebenen Pipeline. Auch kann ausgewählt werden, welche Knotenart als Blattknoten verwendet werden kann, wenn nicht eine andere Art die hierarchisch Tiefer liegt auch angezeigt werden soll. In Abbildung 3.11 ist ein benutzerdefiniertes Layout zusehen, dass sich deutlich vom Standard abhebt.

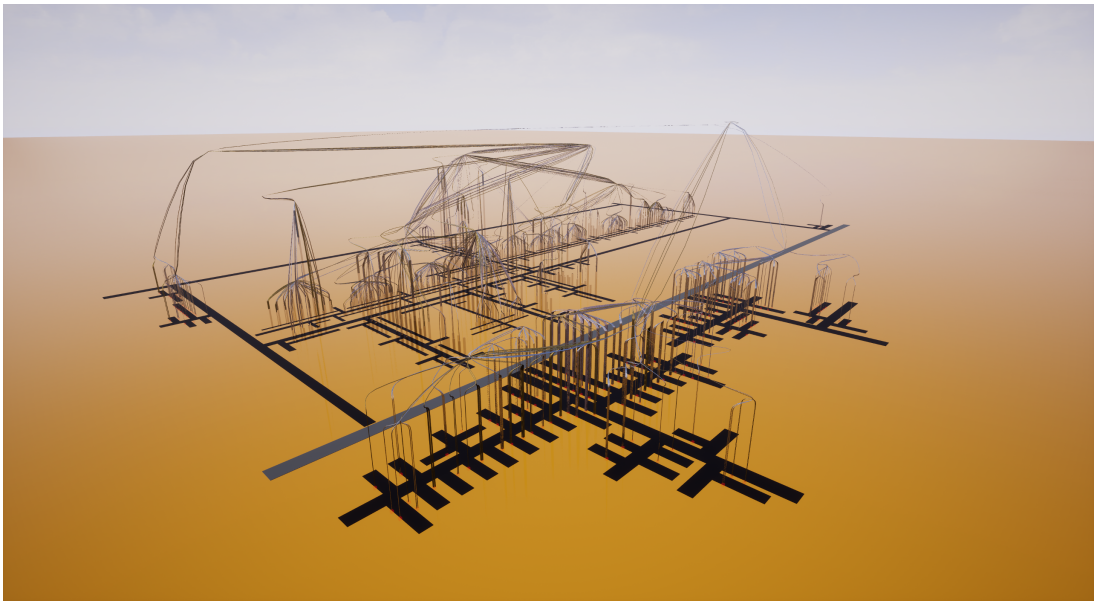


Abbildung 3.11 Gläserne Stadt: Methoden aus Glas; Dateien in Rot; Bibliotheken in durchscheinend Rot; Verbindungen aus Chrom, Kupfer, Stahl und Gold

Die Kanten ließen sich jedoch nicht so einfach umsetzen. Wie im Kapitel 2.5.2 erwähnt, sorgt ein naiver Ansatz hier zu enormen Kosten auf der CPU. Daher mussten wir eine Lösung für diesen Überschuss an Draw-Calls finden. Nachdem wir verschiedene Ansätze erfolglos ausprobiert hatten, stießen wir auf eine Editor-Funktion für Developer die MergeActors heißt. Diese ließ sich zwar nicht auf eine erzeugte Stadt anwenden, da diese zu groß für die Unreal Engine waren,

[20] siehe Kapitel 3.1.2

allerdings konnten wir durch Reengineeringwerkzeuge den aufgerufenen Quellcode tief in der Engine lokalisieren und den Dialog aus dem Editor in C++ nachbauen und endlich die einzelnen Komponenten einer Verbindung zu einem Mesh zusammen führen. Vorher waren für jede Strecke auf den einzelnen Punkten einer Spline Component ein eigens Mesh notwendig gewesen. Durch diesen Kniff konnten wir zwar die Draw-Calls reduzieren, jedoch zu kosten des Speichers. Für die Modelle unserer Evaluation mussten über 1000 Meshes gespeichert werden, die alle hinfällig sind sobald eine Metrik ausgetauscht wird und die Stadt neu berechnet wird. Außerdem war selbst mit dieser Methode keine Kombination der vereinfachten `Collider` möglich.[21]

Für die Verbindungen haben wir das Layout in Hinblick auf die Farbe und die Arten der Verbindungen die angezeigt werden können, modular gehalten.

Zusammenfassend haben wir uns also weitestgehend an unsere Modellierung gehalten. Wir haben eine Zusatzebene bei Knoten eingebaut um zwischen dem Algorithmus-Block und der angezeigten Straße unterscheiden zu können. [22]

3.2.2 XML-Reader [MOR]

Nachdem wir uns durch die alternativen durchgearbeitet haben, sind wir schlussendlich doch beim eingebauten XML-Reader geblieben.[23] Dieser hat zwar Schwierigkeiten mit dem DOCTYPE in den Dokumenten, aber wenn diese Zeile entfernt wird läuft er einwandfrei.

3.2.3 Software-City [JG]

Bei der Implementierung des Anordnungsalgorithmus gab es nach den Vortest der Modellierung keine Schwierigkeiten mehr. Die Anwendung auf echten Daten macht es zwar nötig mehrfach über den ganzen Graphen zu gehen, aber dies ist im Verhältnis zur Berechnung der Verbindung sehr günstig. Insofern haben wir es nicht für nötig befunden den Anordnungsalgorithmus weiter zu optimieren.

Wir haben des weiteren dafür gesorgt, dass die Anordnung nach den EvoStreets nur eine mögliche Implementation der abstrakten Layout Klasse darstellt. Die Berechnung des Algorithmus ist um MVC zu erfüllen in eine eigene Komponente ausgelagert worden. Diese Generalisierung in unserem Plugin soll es für die Verwendung in weiteren (Software-)Graph-Visualisierungen leicht Verwendbar gestalten. Diese haben wir dadurch gewährleistet, dass im In-Editor Modus eine

[21] mehr hierzu in Kaptitel 3.2.4
[22] siehe Kapitel 3.1.2
[23] siehe Kapitel 3.2.2

sogenannte `GeneratorComponent` ausgewählt werden muss, die entscheidet was für eine Visualisierung aufgebaut werden soll.

Die Straßenbreite, den Abstand zwischen den Häusern und die Höhe der Straßen haben wir modular gehalten und an die GUI weiter getragen, damit es leichter ist sinnvolle Werte für die visualisierte Software zu finden.

In der Abbildung 3.12 zeigt sich wie unsere Implementierung mit dem Demoprojekt umgeht.

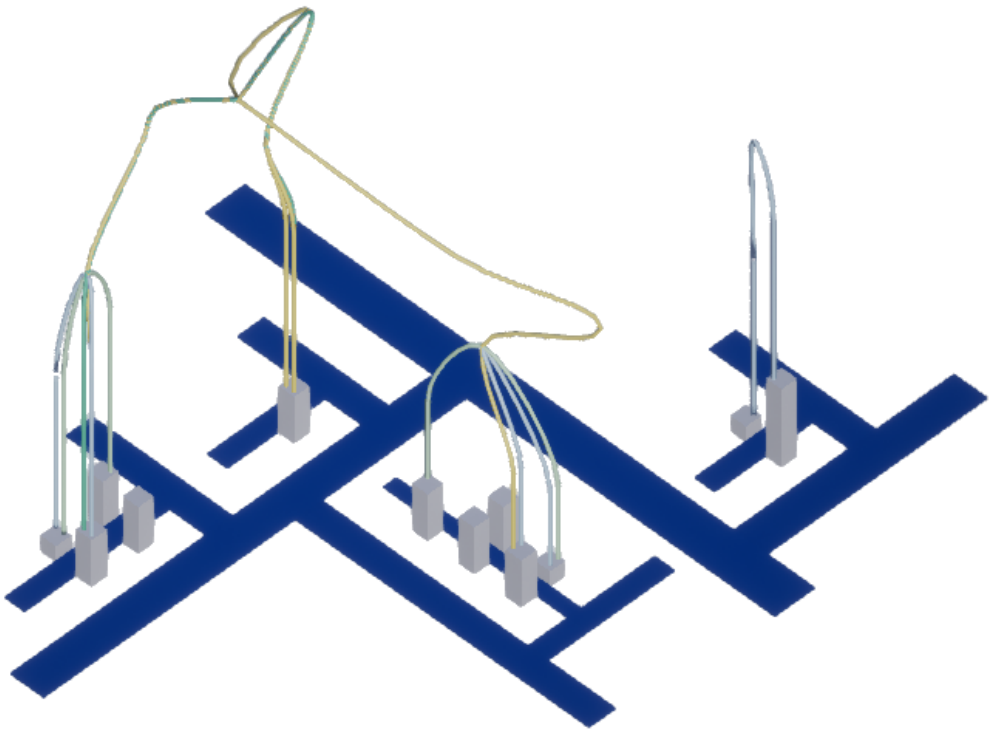


Abbildung 3.12 Demoprojekt orthografisch von der Seite

3.2.4 Implementierung der Interaktion [MOR]

Um uns die Umsetzung unserer Modellierung(siehe Kapitel 3.1.6) zu erleichtern haben wir von dem Pluginansatz der Unreal Engine Gebrauch gemacht[24]. Wir stießen auf das RunebergVR

[24] siehe Kapitel 2.5.2

Plugin[25]. Dieses Plugin stellt einen `Pawn` zur Verfügung von dem eigene Erben können, der Grundlegende Funktionen wie verschiedene Bewegungsmodalitäten, Griff Auswahl und minimale Gestensteuerung anbietet. Davon nutzen wir den Flugmodus für die Bewegung.

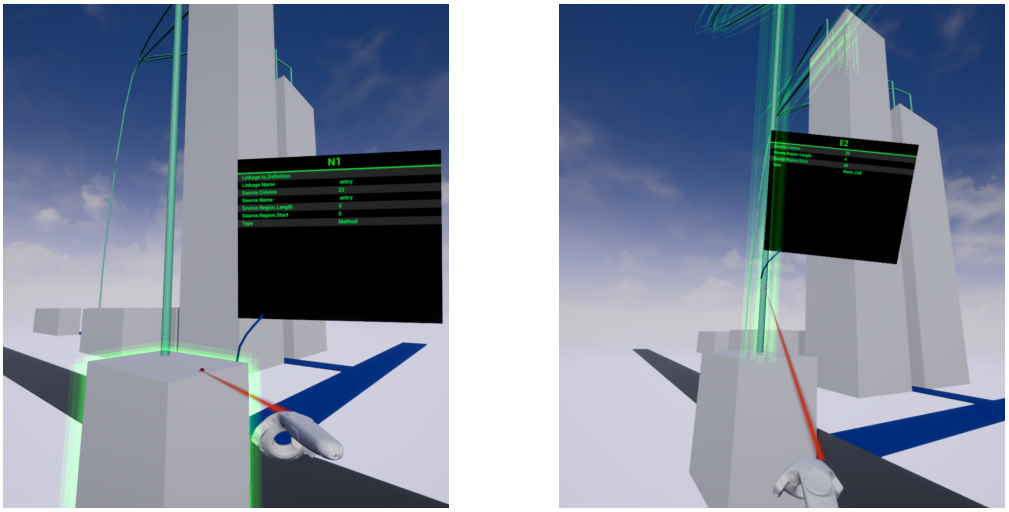
Die von uns angedachte Selektion mussten wir selbst mit einem `Raycast` umsetzen. Anzumerken ist hier, dass wir Aufgrund der zusammengefassten Kanten-Meshes den komplexen `Raycast` nehmen mussten, da die vereinfachten `Collider` beim zusammenführen Fehlerhaft sind. Da der Berechnungsaufwand jedoch zu keinem Zeitpunkt der Taktgeber unserer Bildwiederholungsrate war, sahen wir darin kein Problem.

Da die Unreal Engine keine mehrfach Vererbung, auch nicht durch Interfaces erlaubt, mussten wir bei der Programmierung unserer `Pawns` eine Funktionsbücherei schreiben um die gemeinsamen Funktionalitäten nur einmal implementieren zu müssen.

Wie in Kaptiel 3.1.6 angesprochen, haben wir versucht, die Interaktionen für die Maus-/Tastatur-Variante an den Interaktionen in Computerspielen anzupassen, um das Training dieser Steuerungsmodalität für unsere Zwecke ausnutzen zu können.[26] Im Falle des Head Mounted Display haben wir bei Weitem nicht alle Möglichkeiten der eingesetzten Controller ausgereizt und auch keine besonderen, neu zu erlernenden Menüs eingesetzt, sondern auf einfache und schnelle Erlernbarkeit und Übertragbarkeit aus dem Bereich der Maus-/Tastatur-Interaktion geachtet. Die Interaktion mit dem Head Mounted Display nutzt beide Controller. Die Selektionen von 3D-Objekten erfolgt bei beiden Systemen etwas unterschiedlich: In der Maus-/Tastatur-Variante wird das Objekt unter der Spitze des Mauszeigers durch Klick mit der linken Maustaste selektiert; in der Head Mounted Display wir durch Ziehen des Triggers ein *selection beam* (siehe Abbildung 3.13) dargestellt, mit dem Objekte in einer festgelegten Maximalentfernung ausgewählt werden können. Die eigentliche Auswahl erfolgt durch loslassen des Triggers.

Wir haben die Interaktionsmöglichkeiten in einem kleinen Vorversuch getestet. Die Testkandidaten beschrieben die Interaktionen als einfach, gewohnt und intuitiv, obwohl sie z. T. nur wenig bis keine Erfahrung mit Head Mounted Display-Systemen jeglicher Art hatten.

[25] Quelle: *https://github.com/1runeberg/RunebergVRPlugin*, Abruf: 12.10.2017
[26] siehe Kapitel 2.4



(a) Elementauswahl **(b)** Verbindungsauswahl

Abbildung 3.13 Auswahl in der Head Mounted Display-Umgebung
Der Laser wird duxrch Ziehen des Triggers[27]

Schwierigkeiten bei der Umsetzung

> Von ungezählten Dingen weiss man, dass man sie nicht weiss. Die Anzahl der Dinge, von denen man weiss, dass man sie nicht weiss, soll mal als verwickeltes Beispiel dienen. Das ist nicht weiter schlimm, denn wenn man weiss, dass man etwas nicht weiss, fragt man einfach Google, fertig. Schwieriger wirds, wenn man nicht weiss, was man nicht weiss. Diese Sorte Nichtwissen ist leider eine offene Tür für heftige Überraschungen.
> — Schreiber (2005)

Wie bereits beschrieben, vermuteten wir, dass wir über grundlegende Kenntnisse im Umgang mit der Unreal Engine verfügten. Im Laufe unserer Bachelor-Arbeit mussten wir leider feststellen, dass unsere Kenntnisse bei weitem nicht für die unsere Aufgabenstellung genügten. Inbesondere durch Performance-Probleme wurden uns immer wieder Rückschläge versetzt, die uns unsere Konzepte neu Durchdenken und immer wieder neue Optimierungen ausprobieren ließen.

Geplantes Vorgehen:

Wie schon an anderer Stelle erwähnt war unser geplantes Vorgehen eine Konzeption der Softwarevisualisierung, eine Modellierung und schließlich die Umsetzung. Wir hatten geplant die

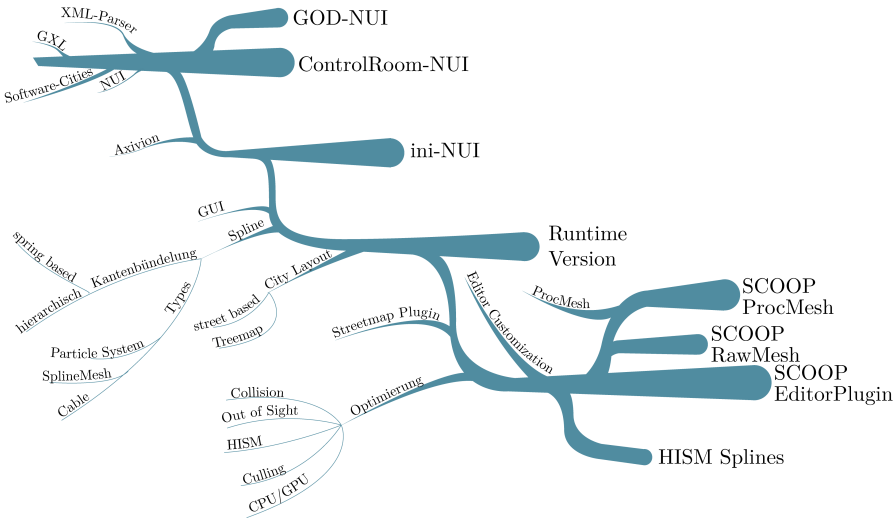


Abbildung 3.14 Projektverlauf

einzelnen Bereiche, namentlich GXL-Parser, Anordnung der Blöcke und Berechnung der Verbindungen, separat zu implementieren und bei zufriedenem Einzelergebnis zu kombinieren.

Das unser geplantes Vorgehen nicht umsetzbar war, wurde uns bei unseren ersten Vortests mit Zufalls generierten Daten klar, bei denen wir in der Größenordnung einer kleinen Software City waren. Die nötige Bildwiederholungsrate die für einen sinnvollen Umgang mit dem Head Mounted Display nötig sind, waren nicht erreichbar.[28] Es war also klar, dass optimiert werden müsste, die Suche nach einem Funktionstüchtigen Optimierungsansatz führte uns allerdings über einige Umwege wie in Abbildung 3.14 dargestellt ist sah unser tatsächliches Vorgehen etwas ausufernder aus.

Tatsächliches Vorgehen:

Nachdem unser naive Ansatz den Software Graphen dynamisch zu Beginn des In-Game Modus in eine Software City umzuwandeln gescheitert war, versuchten wir erst diesen zu optimieren. Doch auch dies scheiterte, denn dynamisch Erzeugte Meshes haben eine sehr teure Lichtberechnung. Es wurde also klar, dass wir im In-Editor Modus eine Lösung finden mussten.

Wir hatten allerdings keine Erfahrung damit, wie man in diesem Modus Dateien einlesen kann. Bei der Suche nach einer Lösung wurden wir wieder einmal von der unglaublich Lückenhaften Dokumentation der Unreal Engine überrascht. Mehr durch Zufall stießen wir schließlich auf ein Plugin, welches OpenStreetMap Karten die als xml-Datei vorliegen durch einen Import zu

[28] siehe Kapitel 2.1.7

einem Asset mit den nötigen Daten verwandelt. Durch Reengineering dieses Plugins konnten wir schlussendlich eine Lösung im In-Editor Modus finden.

Doch auch damit noch nicht genug. Die Kanten machten uns aufgrund ihrer schieren Menge Schwierigkeiten. Das hierarchische Bündeln der Kanten sorgte dafür, dass es zu viele Meshes gab, da jeder Abschnitt ein eigenes brauchte, um ein flüssiges Erlebnis zu gewährleisten. Wir versuchten durch verschiedene Ansätze diesem Problem Herr zu werden. Der **erste** Ansatz den wir in diesem Zusammenhang ausprobierten, war aus den berechneten Meshes, die in den SplineMeshComponents entstanden nachdem das Eingangsmesh entlang der Spline Component verdreht wurde, über die In-Editor Funktion `MergeStaticMesh` zusammen zuführen. Diese hatte allerdings gleich vier Nachteile. Erstens konnten im In-Editor Modus nur die gesamte Stadt verschmolzen werden, zweitens gingen Informationen der Verdrehung und Skalierung verloren. Drittens schon kleine Städte waren zu groß für den `FileWriter` der Unreal Engine welcher schon bei `2GB` aufgibt. Viertens, wenn die Stadt verschmolzen wird, können die einzelnen Elemente nicht mehr angesprochen werden und eine neue Berechnung die eine Zuordnung vornimmt müsste her. Unser **zweiter** Ansatz war, nur die Meshes zu verbinden die zu einer Kante gehören. Dafür mussten wir an die Rohdaten der verdrehten SplineMeshComponents ran. Dies erwies sich als äußerst schwierig, da auch hier wieder eine geeignete Dokumentation ausblieb und das Reengineering zu lange brauchte. Also mussten wir diesen Ansatz verwerfen. Der **dritte** Ansatz war aus den Rohdaten `ProceduraleMeshes` zu bauen, die teurer in der Berechnung sind als Static Meshes sich jedoch durch einen Knopfklick im Editor zu einem Static Mesh zusammenführen lassen. Die Funktion hinter diesem Knopf machten wir ausfindig und verwendet diese nach dem wir aus den Rohdaten die `ProceduralMeshes` der einzelnen Kanten gebaut hatten. Das Ergebnis nutzte allerdings wieder nicht die Verdrehung der SplineMeshComponents verloren. Immerhin die Skalierung bleib erhalten. Dies war allerdings immer noch kein Nennenswerter Erfolg. Der **vierte** Ansatz, war einfach wie in Spielen üblich alles auszublenden was zu weiter weg ist. Dies ging zwar entgegen dem Gedanken sich einen Überblick verschaffen zu können, aber zu diesem Zeitpunkt waren wir bereit dies hinzunehmen. Problem hierbei war, dass die Erkennung für diesen Vorgang (culling genannt) ein Ende der SplineMeshComponents nutzte und somit ein Grenzwert gefunden werden musste, bei dem trotzdem noch einer Teilverbindung gefolgt werden konnte ohne, dass diese verschwindet bevor man die nächste Teilverbindung sehen konnte. Da einige Verbindungen über die ganze Stadt gehen war es unmöglich einen Wert zu finden der die nötigen Vorteile brachte und gleichzeitig bei allen Verbindungen funktionierte. Auch löste es nicht, das Problem, dass in den Bereichen in denen mehrere hierarchische Ebenen sich nah bei einander bündelten die Bildwiederholungsrate einbrach. Daher wurde dieser Ansatz wieder verworfen.

Unser **fünfte** Ansatz war mit die Funktion `MergeActors` aus dem In-Editor Modus zu verwenden. Diese ist im Gegensatz zu `MergeStaticMesh` dazu gedacht, auch Verhalten und ähnliches zu kombinieren. Wenn diese allerdings auf alle SplineMeshComponents einer Spline Component

angewendet wird, entsteht ein Static Mesh, welches die richtige Form und Farbe hat. Lediglich die vereinfachten Kollision wird nicht unterstützt. Durch ein Developer-Tool aus dem In-Editor Modus konnten wir den Quellcode hinter dieser Funktion ausfindig machen und durch Reengineering die nötigen Information zur Ausfüllung des Dialogs in C++ finden. Dieser Lösungsansatz hat ein ausreichendes Ergebnis erzielt. So dass wir endlich die nötigen Bildwiederholungsraten hatten um an eine Evaluation mit einem Head Mounted Display zudenken.

Speicherbarkeit

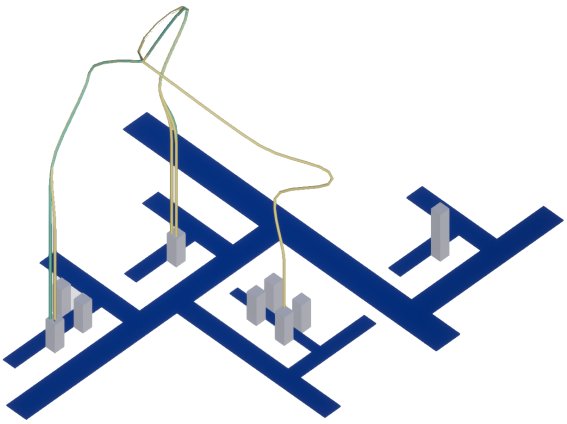
Ein ausdrücklicher Wunsch des Betreuers war es eine fertige Visualisierung speichern zu können, damit diese später erneut betreten werden kann. Dies stellte uns vor ein Problem mit der Serialisierung der Unreal Engine, wir hatten uns bemüht der Übersichtlichkeit halber in unseren Klassen und Strukturen möglichst nur dann auf die Unreal Engine Makros zurückzugreifen, wenn dies auch notwendig für die Berechnung bzw. Anzeige war. Ansonsten hatten wir reinen C++Code geschrieben. Alle Datenstrukturen die jedoch in C++ geschrieben sind und nicht mit dem Makro `UPROPERTY` ausgestattet wurden, werden nicht serialisiert. Außerdem gilt, dass alle Datenstrukturen die in einer `UPROPERTY` vorkommen dürfen aus der Unreal Engine stammen müssen. Dass heißt zu den primitiven Datentypen, den Klassen oder Strukturen der Unreal Engine, die mit dem Makro `UCLASS` oder `USTRUCT` und jeweils dem `GeneratedBody()` ausgestattet wurden, gehören. Dies hatte zur Folge das wir fast unsere komplette Datenstruktur überarbeiten und in Unreal Engine gerechte Formen umbauen mussten. Danach ging das Speichern allerdings endlich einwandfrei... Bis auf die seltenen Fälle in denen die Unreal Engine Referenzen auf geänderte oder gelöschte Assets nicht bereinigt hat. In diesen Fällen half oft nur ein neu erstellen. Dieses Problem trat häufig in Zusammenhang mit Levels auf.

3.2.5 Tests

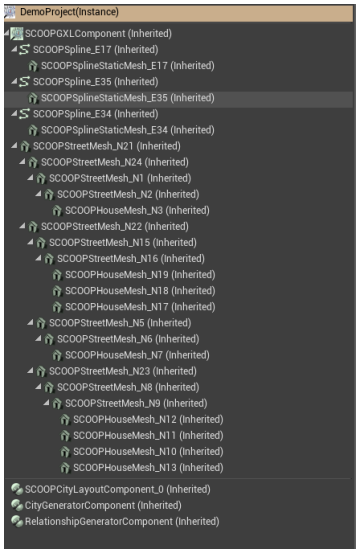
Da es in unserer Anwendung um eine Visuelle handelt, die mit großen Mengen an Werten arbeitet, viel es uns schwer Tests für diese Mengen zu formulieren. Stattdessen verließen wir uns auf Akzeptanztest und Tests mit kleinen Mengen um zu prüfen ob unsere Ansätze grundsätzlich richtig zu arbeiten scheinen.

Die Algorithmen wurden mit Zufallswerten geprüft. Im Falle der Anordnung der Häuser wurden die 5 Prinzipien nach denen die Stadt aufgebaut geprüft. Erst wurde ein einzelnes Haus geprüft, dann ein Straßenszenario und schließlich ein verschachteltes Viertelszenario. Schon bei einem Szenario dieser Größenordnung viel auf wie Aufwendig die Berechnung per Hand ist, sodass nach erfolgreichem absolvieren dieser Tests auf Akzeptanztests und Stichproben zurückgegriffen wur-

de. Zur Veranschaulichung ist in Abbildung 3.15a die Form eines Vierteltests auf kontrollierten Eingaben zusehen.



(a) Methodensicht des Demoprojekts



(b) Components der Methodensicht des Demoprojekts

Bei den Kanten wurden ebenfalls hauptsächlich Akzeptanz und Stichprobentests verwendet. Im Editor ist es möglich den genauen Verlauf einer Spline Component nachzuverfolgen, sodass diese Stichprobenhaft geprüft werden kann.

Das die richtigen Daten vollständig angezeigt werden wurde die Abzählen an den aufgebauten Städten geprüft. Den GXL-Dateien wurden die Anzahlen der Anzuzeigenden Häuser und Verbindungen entnommen und dies dann im Editor geprüft. In der Tabelle auf Seite 164 sind die Daten für diese Zählaufgaben notiert. In Abbildung 3.15b lässt sich für das Demoprojekt ein Zähltest nachvollziehen und sogar die GXL-Datei mit der Ausgabe abgleichen.

3.2.6 Dokumentation

Zur Dokumentation lassen sich dreierlei Dinge anmerken die erste, ist dass bedingt durch den großen Anteil an Fehlversuchen und neuem Wissen, welches wir im Zuge dieser Arbeit zum Umgang mit der Unreal Engine erwerben mussten, kein klarer Stil der Dokumentation zu finden war. In der Arbeitsgruppe galt unsere Arbeit unter anderem als Machbarkeitsstudie, weswegen der Fokus nicht auf Wartbarkeit, sondern auf Ergebnisse gesetzt wurde.

Zweitens ist es bei der Übersetzung der Unreal Engine von Werten die als `UPROPERTY` für den Editor gekennzeichnet werden, der erste Kommentar der über diesen Werten steht, als Info beim

Hovern mit der Maus über dem Eingabe Feld im Editor zusehen. Das hat zur Folge, dass wann immer die Bezeichner uns nicht sprechend genug erschienen wir diese Werte Dokumentiert haben.

Als drittes sei angemerkt, dass sich Blueprints durch Kommentarboxen auszeichnen mit denen im Gegensatz zum Quelltext klare Bereiche kommentiert werden können. Dies macht die Kommentare deutlich übersichtlicher. Eine Alternative zu diesen Kommentarboxen stellt dort allerdings auch die Zusammenfassung zu Makros oder Funktionen da, die dann ebenfalls Werte kapseln und die Übersichtlichkeit erhöhen. Ein Nachteil der Kommentarboxen, wie sie bei den Blueprints verwendet werden, ist dass lange ausführliche Beschreibungen sehr aufdringlich wirken und Aufgrund der Textgröße von den eigentlichen Berechnungen ablenken können.

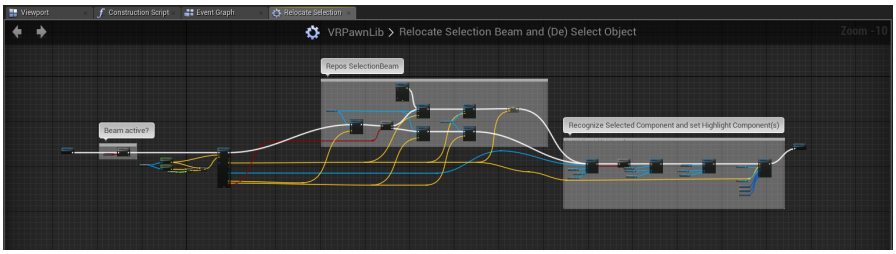


Abbildung 3.16 Kommentare in einer Blueprint

3.2.7 Systemstruktur

Unser endgültiges System ist als Plugin für die Unreal Engine realisiert. Bei der direkten Erzeugung der Knoten des Softwaregraphen (hier: die Häuser) und insbesondere der Kanten (hier: der Verbindungen) in einer live erzeugten Version hatten wir nicht genügend Performance, um ein ruckelfreies Bild zu erzeugen.[29] Durch die Realisierung als Plugin haben wir die Möglichkeit, die 'Landschaft' (hier: die Stadt mit Straßen, Häusern und Verbindungen) schon vor der eigentlichen Laufzeit des Systems zu kreieren. Andererseits haben wir dadurch während der Laufzeit nur eingeschränkte Möglichkeiten, auf Änderungen zu reagieren: Einblenden und Ausblenden von `Actors` wäre möglich sowie die Änderung der Materialien auf ihren Oberflächen. Veränderungen der Mesh-Struktur ist zwar grundsätzlich auch möglich, zieht aber aufwändige Licht-/Schattenberechnungen nach sich, die die Framerate rapide fallen lassen.

Gleiches gilt für Kollisionen mit den graphischen Objekten: Je ausgeprägter die Kollisionen sein sollen, desto mehr Leistung geht auf diese Berechnungen verloren. Wir haben uns daher dafür entschieden, die Kollisionen auf ein notwendiges Minimum zu reduzieren: Den Hit-Test zwischen einem Strahl und einem Objekt bzw. dem Mauszeiger und einem Objekt. Es gibt Beispiele im Internet, die beschreiben, dass sie alle X Wiederholungen die möglicherweise für Kollisionen in

[29] siehe Kapitel 3.2.4

Frage kommenden Objekte aktualisieren, um damit die Framerate wieder anzuheben. In einigen Tests haben wir allerdings bei der Anzahl von Objekten, die wir darstellen müssen, keinen signifikanten Unterschied erkennen können. Wir nehmen an, dass das Setzen der Kollisionen für einige Zehntausend Objekt ähnlich aufwändig wie die Berechnung möglicher Kollisionen ist und sich die Effekte gegenseitig aufheben. Bislang berücksichtigen wir daher die Entfernung der Objekte nicht in unserem System.[30]

Unser Plugin ist als erweiterbares Plugin aufgebaut. Grundlegende Idee war dabei, dass durch den Austausch der Visualisierungsklassen die Stadtvisualisierung entweder durch andere Visualisierungsformen ergänzt oder ersetzt werden können sollen. Die Systemstruktur ist in Abbildung 3.17 dargestellt.

Grundsätzlich ist das Plugin in zwei Module unterteilt: Das eine Modul ist für die Interaktion mit dem Unreal Engine-Editor zuständig, das andere ist das Laufzeitsystem.

Die zentrale, gemeinsame Klasse für die Datenhaltung ist die Klasse `SCOOP`. In ihr werden die aus der GXL-Datei extrahierten Daten abgelegt und sie wird an andere Klassen bei Bedarf weitergereicht, wenn sie Zugriff auf die Klassenstruktur benötigen.

Für den Aufbau der Datenstruktur werden unsere Klassen aus dem `SCOOP` Plugin eingesetzt, welche die Elemente der GXL-Datei in etwas anderer Form repräsentieren. Insbesondere die Kanten-Struktur der GXL-Datei, die für jede Kante einen `from`- und einen `to`-Knoten definieren, hängen wir direkt an die Knoten, die wir als zentrales Element der Betrachtung sehen. Die Kanten tauchen lediglich als Attribut der Verbindung zwischen zwei Knoten auf, wobei wir unterscheiden, ob es sich bei den Kanten um ein- oder ausgehende Kanten handelt.

SCOOPAsset

Wenn ein `SCOOPGXLFileAsset` in einen Level gezogen wird (Schritt 2) erzeugt das Plugin automatisch ein `SCOOPActor`, welches als zentrales Element für den Zugriff auf die GXL-Daten und die Generierung der Visualisierung eingesetzt wird. Das `SCOOPActor` erhält automatisch eine `SCOOPGXLComponent` als nicht-löschbare Standard-Komponente mitgeliefert. In dieser Komponente kann bei Bedarf ein anderes `SCOOPGXLFileAsset` als Daten-Grundlage ausgewählt werden.

In dieser Komponente kann eine Generatorkomponente und Generierungsoptionen ausgewählt sowie die Generierung angestoßen werden. Die Generatorkomponente kann ihrerseits wiederum Optionen anbieten, die speziell auf die in der Komponente vorgesehene Visualisierung abgestimmt sind.

[30] ein zweiter Grund war das `culling` nicht erfolgreich verlief; siehe Kapitel 3.2.4

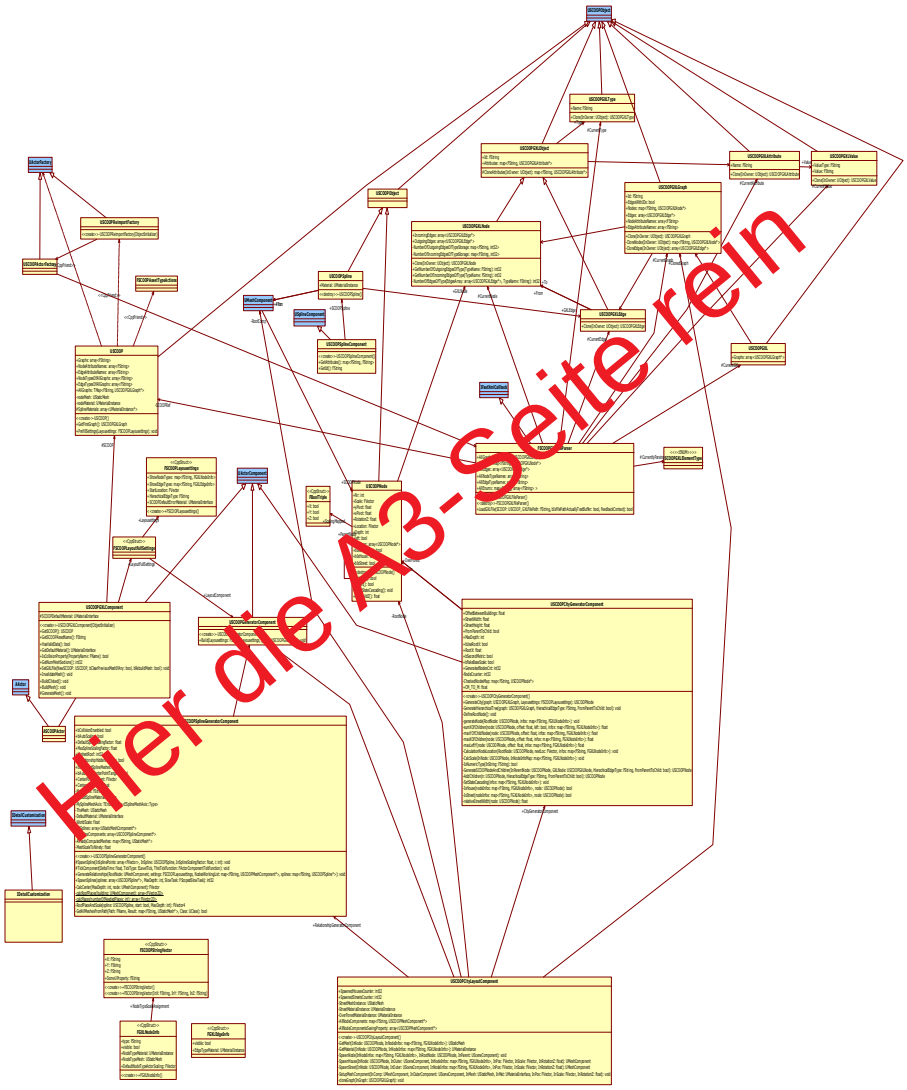


Abbildung 3.17 UML-Implementatierungsmodell unseres Systems

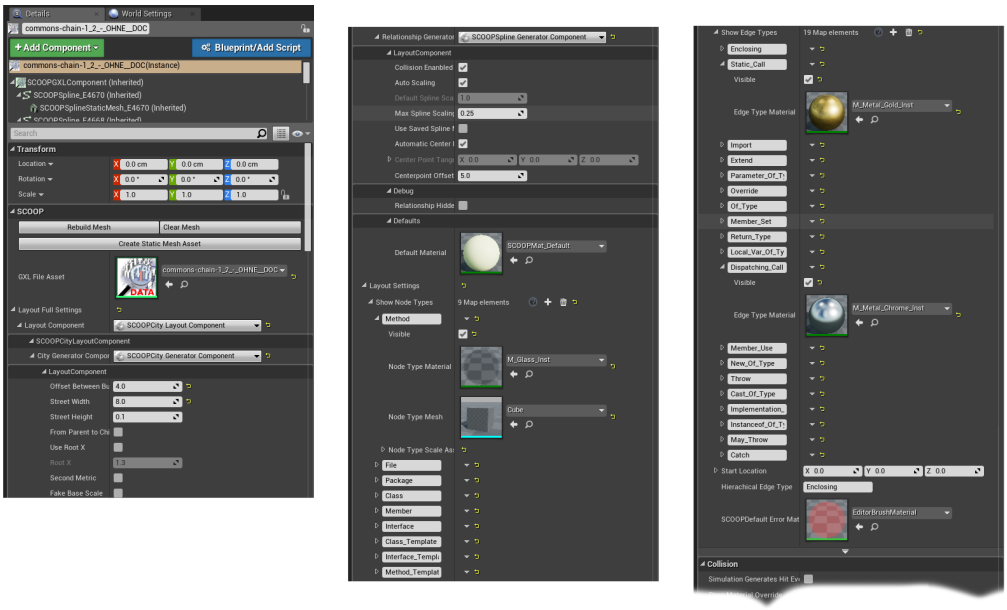
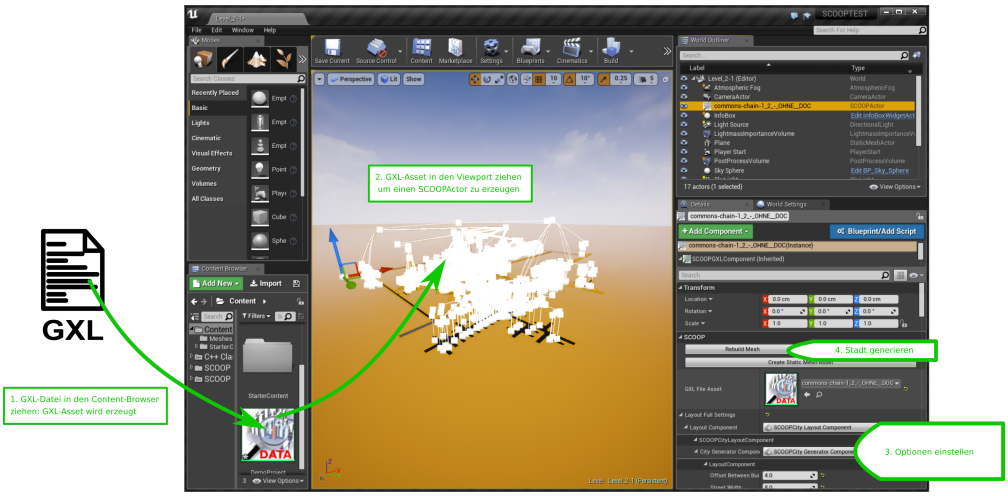


Abbildung 3.18 Übersicht der System-Optionen



GeneratorComponents

Im SCOOPPlugin sind zwei Generator-Komponenten enthalten, von denen eine ein Stadt-Modell erzeugen kann und die andere auf diesem Modell aufbauend hierarchisch gebündelete Verbindungen zwischen den Elementen der Stadt gemäß der Deklaration der zugrunde liegenden GXL-Datei erzeugt. Beide Komponenten erzeugen aus den Daten und den im Editor vor der Generierung gewählten Optionen unveränderliche Static Meshes als Kindkomponenten des SCOOPActors. So wäre es ein leichtes, auch mehrere Visualisierung parallel in einem Unreal Engine-Level zu erzeugen und bspw. vergleichend nebeneinander zu legen.

Je Komponente kann auf die in der GXL-Komponente definierten Optionen zugreifen sowie eigene Deklarieren, die im Editor angezeigt und vom Nutzer angepasst werden können. Die Options werden nach Unreal Engine-Schema strukturiert und sortiert. Für die Straßenvisualisierung lassen sich bspw. Materialen, das genutzte Mesh, Sichtbarkeiten und grundsätzliche Größen der Straßenzüge, für die Verbindungen ebenfalls bspw. Sichtbarkeiten, Mesh und Materialen. Auf diese Weise wäre es bspw. auch möglich einen Unreal Engine-Level zu erzeugen, in dem verschiedene Verbindungen unabhängig voneinander ein- oder ausgeblendet werden könnten. Durch Änderung der Meshes könnte man auch formenreichere Varianten erzeugen als wir sie für unsere Evaluation genutzt haben.

GXL-Dateien

Bei Import von GXL-Dateien werden die enthaltenen Daten interpretiert, teilweise umstrukturiert und in verschiedene speicherbasierte Arrays (bspw. AlleKnotentypen-Array) verteilt. Das Unreal Engine-Persistenzsystem sorgt dafür, dass die Daten auch nach dem Beenden und neustarten noch vorhanden sind. Die erzeugten Datenstrukturen sind inhaltlich noch sehr nahe an der originalen Datenstruktur der GXL-Dateien, Filterungen und Begrenzungen nehmen die Visualisierungskomponenten (genannt *Generatoren*) vor.

bekannte Einschränkungen

Folgende Einschränkungen sind uns aufgefallen:

- Das Plugin ist für die Unreal Engine 4.17.3 geschrieben. Es kann auch in neueren Versionen eingesetzt werden, muss dann aber konvertiert werden. Konvertierungsanleitungen finden sich im Internet.
- GXL-Datei mit einer Größe von mehr als 2 GB können bislang nicht eingelesen werden. Dies ist eine interne Beschränkung der Unreal Engine.

- Wir gehen beim Einlesen der GXL-Datei von einem bestimmten Format aus. Fehlerhafte Kanten werden momentan zwar großzügig ignoriert und dies protokolliert, doch unter bestimmten Umständen kann das Einlesen Probleme verursachen. Beispielsweise sind bislang alle verwendeten GXL-Dateien so aufgebaut gewesen, dass zunächst die Knoten und anschließend die Kanten enthalten waren. Unser Parser verlässt sich auf diese Reihenfolge.
- Blueprint-Unterstützung ist in diesem Plugin momentan nicht vorgesehen, der Zugriff über C++ aber problemlos.
- Das Plugin nutzt das RunebergVR-Plugin für das einfachere Handling von Interaktionen mit dem Head Mounted Display. Dies funktioniert auch im Großen und Ganzen gut, doch enthält unsere eingesetzte Version einen Fehler in der Beschleunigungsberechnung, der erst durch einen Vergleich zwischen Desktop- und Head Mounted Display-Anwendung offenkundig geworden ist. In neueren Versionen des Plugins (in neueren Unreal Engine-Versionen) scheint dieser Fehler behoben zu sein.

Kompatibilität

Das Plugin benötigt für die Kompilierung MS Visual Studio und ein Unreal Engine-C++-Projekt. Alternativ kann das Plugin auch im SCOOPTest-Projekt getestet werden. Das Plugin sollte auf allen Plattformen zu laufen, die die Unreal Engine untersützt. Wir haben allerdings nicht alle Platformen getestet (insbesondere nicht die mobilen).

Aktion	Maus/Tastatur/Desktop	Head Mounted Display
Umsehen (l/r/o/u)	Bewegen der Maus (l/r/v/z)	Bewegen des Kopfes (l/r/o/u)
Bewegen (v/r/l/r/o/u)	Tastatur (w/s/a/d/e/q)	TrackPad R (o/u + Controllerrichtung)
Auswählen von Objekten	Klick LMT	Trigger R loslassen (Selection Beam)
Information	Hover (2 sek)	Trigger R halten (2 sek) (Selection Beam)
Verfolgen	Klick LMT auf selektierte Verbindung	Triggerklick R auf selektierte Verbindung

Tabelle 3.1 Interaktion mit Maus, Tastatur und Controller
Legende:

o	oben
u	unten
v	vor/vorwärts
z	zurück
l	links
r	rechts
LMT	Linke Maustaste
Trigger R	Trigger am rechten Controller, siehe Abbildung 3.7
TrackPad R	TrackPad am rechten Controller, siehe Abbildung 3.7
Hover	Über einem Objekt mit dem Zeigewerkzeug (Mauszeiger oder Selection Beam) stehen bleiben
Selection Beam	siehe Abbildung 3.13

Kapitel 4

Evaluation [MOR]

Neben der Konzeption, Modellierung und Implementierung eines Tools zur Visualisierung von Software nach dem Software-City-Vorbild[1] hatten wir die Aufgabe, unter Verwendung unseres implementierten Visualisierungssystems eine kontrollierte Evaluation durchzuführen, die eine 3D-Stadt-Visualisierung auf einem Monitor der gleichen Visualisierung in Head Mounted Display gegenüberstellt. Die Software-City-Visualisierung wird für die Visualisierung verschiedener Aspekte genutzt, so u. a. von Steinbrückner (2013) für die Visualisierung der Evolution.

In Kapitel 2.1 haben wir bereits ausgeführt, dass aufgrund der höheren Präsenz vermutlich die räumliche Orientierung in Head Mounted Display-Systemen besser als in Desktop-Systemen ist. Diese Vermutung wollen wir in unserer Evaluation mit Hilfe einiger Probanden, unseres Systems, einer angeglichenen Desktop- und Head Mounted Display-Steuerung[2] sowie identischen Visualisierungen[3] untersuchen.

Um Aussagen über das räumliche Orientierungsvermögen einzelner Personen oder gar über eventuelle intrapersoneller Unterschiede zwischen der Desktop- und der Head Mounted Display-Umgebung treffen zu können, müssen wir *räumliche Orientierung* messen und vergleichen können. Die Messbarkeit des abstrakten Begriffs *räumliche Orientierung* setzt voraus, dass er quantifizierbar definiert wird[4] und mittels eines passenden Szenarios[5] überprüft werden kann.

Für den Begriff *räumliche Orientierung* werden in der Literatur verschiedene Ausrichtungen beschrieben. Wir wollen ihn in dieser Arbeit als das Wissen über die eigene Position im Raum definieren. Wir beschreiben im Folgenden zunächst das Szenario, welches als Grundlage für diese Definition und unsere Evaluation dient, um darauf aufbauend unsere Quantifizierung von *räumliche Orientierung* zu verdeutlichen.

[1] siehe Kapitel 2.2.5
[2] siehe Kapitel 3.2.4
[3] siehe Kapitel 4.6.4
[4] siehe Kapitel 4.2
[5] siehe Kapitel 4.1

4.1 Szenario [MOR]

Wir wollen in unserer Evaluation das räumliche Orientierungsvermögen mittels eines Szenarios ermitteln, welches wahrscheinlich jedem in der einen oder anderen Form bekannt ist. Das Szenario beschreibt das Finden des eines Rückwegs zu einem Ausgangspunkt.

> *Sie befinden sich an einem Ort, an dem sie noch nie zuvor waren, in einer Gegend, in der sie noch nie zuvor waren. Beginnend an diesem Ort steuern sie nacheinander einige weitere Orte an und wollen im Anschluss an diesen Ort zurückfinden, von dem aus sie gestartet sind.*[6]

Um dieses Szenario zu bewerkstelligen, nutzen Menschen unterschiedliche Vorgehensweisen. Wir haben für eine informelle Vorabbefragung das oben beschriebene Szenario verschiedenen Menschen[7] vorgelegt und sie gefragt, ob sie (a) ein solche Szenario bereits erlebt haben und (b) wie sie sich hierbei orientiert[8] haben. Grob kristallisierten sich dabei die in Tabelle 4.1 aufgeführten Orientierungstypen heraus. Wir wollen diese Ergebnisse dieser Vorabbefragung als grobe Kategorisierung unserer Probanden nutzen, auch wenn diese Kategorien weder Anspruch auf Vollständigkeit noch auf Wissenschaftlichkeit hegt. Während ihrer Orientierungsphase nutzten die befragten Personen nach eigener Auskunft die in Tabelle 4.2 aufgeführten Orientierungshilfen.

Orientierungstyp	Beschreibung
Breadcrump	Person sucht sich mehr oder weniger zufällig Merkmale der Umgebung aus, an denen sie sich orientieren; je unsicherer die Person ist, desto bewusste läuft dieser Prozess ab
Intuitiv	Person verlässt sich auf seine „Intuition“
Try&Error	Person geht bei der Suche nach dem Versuch-und-Irrtum-Vorgehen vor

Tabelle 4.1 Typen räumlicher Orientierung
aus einer informellen Befragungen, ohne Anspruch auf Vollständigkeit, Reihenfolge spiegelt nicht die Häufigkeit der Nennung wider

[6] Konkret könnte es sich beim Startort bspw. um ein Urlaubshotel oder den riesigen Parkplatz vor einem fremden Stadion handeln. Man ginge bspw. zunächst ins Theater, dann ein Restaurant und anschließend eine Tanzbar resp. zunächst in den Fanshop, dann ins Stadion und schließlich zu einem Fantreff. Am Ende des Tages will man wieder in das Urlaubshotel bzw. zu dem Auto zurück. Im demographischen Fragebogen haben wir diese Fragestellung angedeutet.
[7] nicht den Probanden
[8] also den Weg zurück zum Hotel resp. Auto gefunden

Orientierungsmuster	Beschreibung
Landmarken	Elemente der Umgebung, die auf dem Weg auffallen
Sonnenstand	Stand der Sonne, häufig verknüpft mit Entfernung
Hierarchien	hierarchische Struktur[9]
Reihenfolgen	Reihenfolge von Elementen der Umgebung
Entfernungen	Entfernung von Elementen der Umgebung
Navigationssystem	Nutzung von technischen Hilfsmitteln
Karte	Nutzung von Kartenmaterial
Intuition	ohne aktive, bewusste Orientierung

Tabelle 4.2 Muster räumlicher Orientierung
aus einer informellen Befragungen, ohne Anspruch auf Vollständigkeit; die Reihenfolge spiegelt nicht die Häufigkeit der Nennung wider

4.2 Effektivität und Effizienz räumlicher Orientierung [MOR]

Wir gehen in unserer Evaluation davon aus, dass sich *räumliche Orientierung* mittels zweier Kriterien messen lässt: *Effizienz* und *Effektivität*. Für beide Begriffe gibt es eine Vielzahl an Auslegungen, wir wollen uns für unsere Evaluation auf die Definitionen nach Drucker (1963) stützen. Drucker versuchte die Konfusion und Deutungsüberschneidung zwischen beiden Begriffen zu klären und schrieb bereits 1963: „It is fundamentally the confusion between effectiveness and efficiency that stands between doing the right things and doing things right.“[10] In den Definitionen 4.1 und 4.2 sind die üblichen[11] deutschen Interpretationen zu finden.

Effektivität: Die richtigen Dinge tun.

Definition 4.1 Definition *Effektivität* nach Drucker

Effizienz: Die Dinge richtig tun.

Definition 4.2 Definition *Effizienz* nach Drucker

Gabler Wirtschaftslexikon (2013) verdeutlicht die Drucker'sche Definitionen: Danach ist *Effektivität* „ein Beurteilungskriterium, mit dem sich beschreiben lässt, ob eine Maßnahme geeignet ist, ein vorgegebenes Ziel zu erreichen. Über die Art und Weise der Zielerreichung werden bei der Betrachtung unter Effektivitätsgesichtspunkten keine Aussagen getroffen“[12], während *Effizienz*

[10] Drucker (1963); in Drucker (1967) konkretisiert Drucker seine Definition am Beispiel des Verwaltungsbereichs
[11] lt. *https://de.wikipedia.org/w/index.php?title=Effektivit%C3%A4t&oldid=168981836*
[12] *Gabler Wirtschaftslexikon* 2013, Stichwort: Effektivität.

ein „Beurteilungskriterium [ist], mit dem sich beschreiben lässt, ob eine Maßnahme geeignet ist, ein vorgegebenes Ziel in einer bestimmten Art und Weise zu erreichen."[13]

Wir haben diese Definitionen auf das in Kapitel 4.1 beschriebene Szenario übertragen. In Anlehnung an *Gabler Wirtschaftslexikon* definieren wir *Effizienz* wie in Definition 4.3 als Beurteilungskriterium, ob die Orientierungsstrategie des Probanden geeignet war, zurück zum Ausgangspunkt zu finden. Hierbei bilden wir die Bewegung des Probanden auf eine Dimension ab. Jede Bewegung die in Richtung Ziel bringt wird also positiv jede die von Ziel wegführt negativ gewertet. Dadurch ergeben sich in Abhängigkeit zur Dauer Geschwindigkeiten zu jedem Zeitpunkt die gemittelt eine durchschnittliche Annäherungsgeschwindigkeit ans Ziel bieten. Dieses Maß ist unabhängig von den einzelnen Szenarien und erhöht die Vergleichbarkeit der Messergebnisse. *Effektivität* definieren wir wie in Definition 4.4 als Maßstab, wie gut die Orientierungsstrategie des Probanden geeignet war, um zum Ziel zurück zu finden.

Effizienz in unserer Evaluation: Beurteilungskriterium, wie gut die räumliche Orientierungsstrategie des Probanden geeignet war, zurück zum Ausgangspunkt zu finden, gemessen in [Heimweg in UEE/Dauer]. Wobei Unreal Engine-Einheiten, kurz UEE, eine einheitliche, aber nicht weiter definierte Maßeinheit für die Strecke zwischen beiden Punkten beschreibt.

Definition 4.3 Definition *Effektivität* unserer Evaluation

Effektivität in unserer Evaluation: Beurteilungskriterium, ob die räumliche Orientierungsstrategie des Probanden geeignet war, zurück zum Ausgangspunkt zu finden

Definition 4.4 Definition *Effizienz* unserer Evaluation

Die Effektivität bestimmen wir über die Zielerreichung[14]. Die Effizienz messen wir über die Rückfluggeschwindigkeit, die der Proband erreicht hat: Je kürzer die Rückflugdauer, d. h. je höher die Rückfluggeschwindigkeit, desto besser werten wir den Probanden.

$$Effizienz = \frac{UEE}{s}$$

Definition 4.5 Mathematische Definition *Effizienz* unserer Evaluation

Die Effektivität und die Effizienz stellen unsere abhängigen Variablen der Evaluation dar. Als unabhängige Faktoren legten wir das Modell (1 oder 2), die im Modell genutzte Metrik (A oder B) sowie die Reihenfolge der Nutzung der System (Head Mounted Display zuerst oder Desktop zuerst). Insofern haben wir einen 2x2x2-faktoriellen Versuchsaufbau gewählt. Kovariat zeichnen wir die im demographischen Fragebogen erhobenen Daten auf. Dazu gehören Alter, allgemeine Vorerfahrungen im PC-Bereich, Vorerfahrungen im Umgang mit Desktop-Computerspielen,

[13] *Gabler Wirtschaftslexikon* 2013, Stichwort: Effizienz.
[14] d. h. ob der Proband zurück zum Ausgangspunkt gefunden hat oder nicht.

Grafikprogrammen und VR-Systemen sowie eine Selbsteinschätzung zur Orientierungsfähigkeit in unbekannten Umgebungen. Weiterhin versuchen wir die eingeschlagenen Wege sowie die Anzahl der Fehlversuche zu protokollieren.

4.3 Einflussfaktoren [MOR]

Unser System greift für die Software-Visualisierung auf Ideen bekannter Visualisierungen zurück, die – jede für sich – inzwischen z. T. auch den Sprung aus der Wissenschaft in die Wirtschaft geschafft haben und in etablierten Systemen wie SonarQube (über Plugins)[15] genutzt werden können.[16]

Ein Teil unserer Experiments wird auf Head Mounted Display-Technik aufbauen. Mit dieser Technik wird aber nicht nur eine neue Darstellung eingeführt, sondern auch eine neue Art der Interaktion: Die sogenannte NUI-Interaktion der Virtual Reality-Systeme unterscheidet sich grundsätzlich von der klassischen Interaktion mit Desktop-Systemen. Die Entwicklung von NUI-System befindet sich noch am Anfang und ist nicht auf Virtual Reality-System beschränkt; es gibt zwar einige Ansätze eine Vereinheitlichung, es gibt bislang allerdings noch keinen *Common Sense*, einen allgemeingültigen Styleguide oder gar einen Wirtschaftsstandard. Dies liegt einerseits offenbar an der bisherigen Ausrichtung auf Spiele, in denen gerade durch ungewohnte oder neuartige Interaktionsschemata das Interesse am Spiel geweckt wird, andererseits aber auch daran, dass VR-Systeme erst in jüngster Zeit durch verbesserte Technik[17] für den Arbeitsmarkt interessant werden. Hieraus ergeben sich für unsere Experimente folgende Einflussfaktoren:

- Unsere Probanden kennen sich ggfs. nicht mit der eingesetzten Software oder Head Mounted Display-Systemen aus.
- Die Einarbeitungszeit der Probanden nimmt im Vergleich zum Kern der Evaluation viel Zeit ein.
- Die Interaktion in mit dem VR-System wird sich von der eines Desktop-Systems unterscheiden, sowohl
  - in den Interaktionsmechanismen (Controller statt Maus/Tastatur) als auch
  - in der Datenanzeige (Daten werden direkt in der persönlichen Umgebung angezeigt statt auf Fenstern auf dem Bildschirm) als auch
  - in der Darstellung, Anzeige und Interaktion mit Auswahlmögichkeiten (VR-Selektion statt Menü-, Symbol oder Multifunktionsleiste (Ribbonbar) wie bspw. in Windows)

Um die Unterschiede und den Lernbedarf zwischen dem Desktop- und dem Head Mounted Display-System möglichst gering zu halten, haben wir uns auf minimale Interaktionen beschränkt.

[15] Rinderle 2015–0006.
[16] Details zu den betrachteten Softwarevisualisierungsansätze siehe Kapitel 2.2.5
[17] Unser System siehe Tabelle 4.5

Für die Desktop-Variante haben wir die aus Desktop-Spielen bekannte `WASD`-Steuerung erweitert um die Tasten `E` und `Q` sowie die Maus gesetzt. Diese Tasten werden in einigen (3D-)Desktop-Grafikprogrammen ebenfalls eingesetzt, um den Sichtbereich zu verändern. Für die Head Mounted Display-Variante nutzten wir einen Controller als Richtungszeiger für die Flugbewegungen und den zweiten zum Markieren der Objekte. Details zur Interaktion sind in Tabelle 3.2.4 zu finden. Die Fluggeschwindigkeit sowie die Entfernung, in der Markierungen möglich sind, haben wir für beide Varianten vereinheitlicht.

Die HTC Vive ermöglicht freie Körperbewegungen innerhalb eines abgegrenzten Bereichs. Wir haben uns dazu entschlossen, den Probanden diese Freiheit zu lassen. Wir haben während des Versuchs darauf geachtet, dass sich die Verkabelung der HTC Vive nicht ablenkend um die Beine der Probanden wickelt.

4.3.1 Fragebögen [MOR]

Heutzutage bietet sich der Einsatz eines Fragebogensystems an, da solche Systeme helfen eine Reihe von sonst üblichen Fehlern zu vermeiden und die Ergebnisse der Fragebogen direkt für die Weiterverarbeitung in digitaler Form zur Verfügung stellen. Wir haben uns im Rahmen dieser Arbeit kurz mit der Auswahl eines geeigneten Werkzeugs beschäftigt und dazu die am Markt verfügen einander gegenüber gestellt. Als Bewertungskriterien waren und kostenfreier Zugang, Unterstützung alle angedacht oder in den verwendeten Standard-Fragebögen vorgesehenen Antwortkategorien, Steuerbarkeit der Frageanzeige, Exportierbarkeit, Datenschutz, einfache Trennung der Teilnehmerdaten und Anonymisierbarkeit besonders wichtig. Im zweiten Schritten interessierten uns Benutzbarkeit und Design. Wir haben nach eine Analyse freier Fragebogensystem uns für das System LimeSurvery[18] entschieden, welches allen Anforderungen genügte.

Um Einflüsse der Usablity auszuschließen, wenden wir zum einen den klassischen Software Usablity Scale- und zum anderen den jungen meCue-Fragebogen an.

4.4 These [MOR]

Wir gehen bei unserer Untersuchung von folgender These aus:

[18] *www.limesurvey.org*

Hypothese Im Virtual Environment einer Software City können sich Nutzer – unabhängig von ihrem grundsätzlichen Orientierungstyp – durch die Immersion und die daraus resultierende Place Illusion mit Head Mounted Display-Systemen effektiver und effizienter orientieren als mit vergleichbaren Desktop-Pseudo-3D-Systemen.

Definition 4.6 Hypothese unserer Arbeit

4.5 Planung [MOR]

4.5.1 Aufbau und Ausstattung der Evaluationsumgebung [MOR]

Die Evaluation findet auf einer Grundfläche von 2x3 Metern statt. Die Probanden haben sich vor Beginn des eigentlichen Versuchs bereits einige Minuten in diesem Raum aufgehalten. Sie werden in die Sicherheitsmechanismen der HTC Vive eingewiesen.

4.5.2 Aufgabenstellungen für die Probanden [MOR]

Im Rahmen unserer generellen Aufgabenstellung, die Orientierung zwischen Desktop- und Head Mounted Display-Systemen zu vergleichen, haben wir den Probanden die in Tabelle 4.3 aufgelisteten Aufgaben gestellt.

Nr	Aufgabe	Zeit[19]
1.	Navigieren Sie vom Startelement N13 über eine Verbindung E374 der Art „Call“ zum Element N150	1 - 2 min
2.	Navigieren Sie vom Element N150 über eine Verbindung E475 der Art „Dispatching Call“ zum Element N1350	1 - 2 min
3.	Navigieren Sie vom Element N1350 über eine Verbindung E13 der Art „Call“ zum Element N57	1 - 3 min
4.	Navigieren Sie von diesem Element auf dem schnellsten Wege zurück zum Startelement N13	0.5 - 5 min

Tabelle 4.3 Aufgabenstellung der Evaluation
Die Aufgaben werden je Proband jeweils mit dem Desktop- und mit dem VR-System für 2 verschiedene Softwarevisualisierungen durchgeführt, siehe Tabelle 4.4; die angegebenen Element- und Verbindungsnummer dienen hier nur als Beispiel

4.5.3 Versuchsplan [MOR]

In Vorversuchen haben wir eine Dauer von rund einer Stunde je Proband für die Bewältigung einer Einführung, der eigentlichen Aufgabenstellung sowie der statistischen Anteile ermittelt. In der folgenden Tabelle haben den grundsätzlichen Ablauf je Proband aufgeführt. Die Reihenfolge der eigentlichen Versuche wurde von uns bei jedem Probanden vertauscht.

Für den demographischen, den Usability- sowie den qualitativen Fragebogen haben wir das Survey-System Lime-Survey[20] eingerichtet. Diese papierlose Fragebogen-System vereinfachte uns die Arbeit mit den Daten.

Phase	Beschreibung	
Demographie	1. Datenschutzerklärung 2. Sicherheitshinweis 3. Verzichtserklärung 4. Demographischer Fragebogen	
Versuchseinführung	Beschreibung der zentralen Aufgabe und der Elemente des Versuchs	
	Gruppe 1	Gruppe 2
Übungsphase	Desktop	VR
Aufgabe 1 (Modell 1, Metrik A)	Desktop	VR
Aufgabe 2 (Modell 1, Metrik B)	Desktop	VR
Übungsphase	VR	Desktop
Aufgabe 3 (Modell 2, Metrik A)	VR	Desktop
Aufgabe 4 (Modell 2, Metrik B)	VR	Desktop
Statistik	1. meCue-Fragebogen 2. SUS-Fragebogen 3. zusätzliche qualitative Systembeurteilung	

Tabelle 4.4 Versuchsablauf je Proband
Die Reihenfolge der von VR und 3D wird je Proband vertauscht. Die Navigationsaufgabe ist in Tabelle 4.3 beschrieben. Weitere Details zum Versuch siehe Text.

Jeder Proband durchläuft nach der Übungsphase vier Versuche: zwei mit der VR-, zwei mit

[20] *http://limesurvey.com*

der Maus-Tastatur-Steuerung. Die beiden Versuche je Darstellungsvariante unterscheiden sich dadurch, dass sie unterschiedliche Zustände der gleichen Stadt darstellen. Genauer gesagt, unterscheiden sich die Städte lediglich durch die Höhe der Gebäude. In der Praxis könnten diese Höhenveränderungen bspw. durch die Veränderung des auf die Höhe aufgetragenen Attributs des analysierten Sourcecodes zurückzuführen sein.[21] Um dem Versuch nicht zu vielen Freiheitsgrade zu bieten, gehen wir davon aus, dass sich die Grundstruktur der Stadt nicht verändert: Die Anordnung der Straßen bleibt in unserer Versuchsreihe identisch.

Um Einflüsse durch die Reihenfolge der durchgeführten Versuche auszuschließen, wechseln wir je Proband die Reihenfolge von Head Mounted Display- und Desktop-System. Die Reihenfolge der Städte behalten wir bei.

4.5.4 Rollenverteilung [MOR]

Wir haben zwei Rollen entwickelt, die während des Versuchs eingenommen werden müssen:

Versuchsleiter:
Der Versuchsleiter fungiert während des Versuchs als Ansprechpartner für den Probanden. Er erläutert den Versuch, stellt die Aufgaben, achtet auf den Probanden und koordiniert insgesamt den Versuchsablauf. Inhaltliche haben wir für den Versuchsleiter ein Drehbuch entwickelt, in dem bestimmte Fragen und Formulierungen vorgegeben, um diesbezüglich die Versuche einigermaßen gleich zu halten.

Assistent:
Der Assistent reicht Dinge an, sorgt für eine funktionierende Technik und koordiniert den Versuch im Hintergrund.

Wir verteilten die Rollen im Vorfeld der Versuchsreihe zwischen uns und behielten sie bis zum Ende der gesamten Versuchsreihe bei.

4.6 Durchführung [MOR]

Die Durchführung erfolgte zwischen dem 05. und 07. Oktober 2017. Wir setzten je Proband eine Zeit von 60 Minuten an, was in den meisten Fälle sehr gut hinkam.

[21] In Ausnahmefällen könnten die Höhenveränderungen auch auf Umprogrammierung oder Optimierung einzelner Funktionen zustande gekommen sein. In der Regel würde sich dies aber auch in der Struktur der Stadt niederschlagen: Es würde neu Funktionen und Klassen dargestellt, andere verschwinden oder werden integriert.

4.6.1 Teilnehmer [MOR]

Als Probanden haben wir Studenten verschiedener Semester sowie wissenschaftliche Mitarbeiter (beide Gruppen dem Fachbereichs 3 – Informatik – der Universität Bremen angehörig) genutzt. Wir haben die Studenten und wissenschaftlichen Mitarbeiter direkt auf dem Campus zufällig angesprochen und sie gebeten, an unser Versuchsreihe zu partizipieren.[22]

4.6.2 Technische Umgebung [JG]

Die folgenden Tabellen beschreiben die Technische Umgebung in der die Evaluation statt gefunden hat. Die wichtigsten Hardwaredetails der Zentraleinheit finden sich in Tabelle 4.5, während Tabelle 4.6 enthält Informationen zu den bedeutsamen Peripheriegeräten des Aufbaus.

4.6.3 Umgebung [MOR]

Die Versuche fanden in unserem Projektraum an der Universität Bremen statt. Der Raum bot eine Fläche von 2x3 Metern, die wir im Rahmen der Raumvermessung der HTC Vive zugeordnet haben. Weiterhin hatten wir einen kleinen Nebenraum, in dem die Probanden im Anschluss an die Versuche in Fragebögen beantworten durften. Hier wurden ihnen auch Getränke und Kekse gereicht.

4.6.4 Modellbeschreibung [MOR]

Wir nutzten für unsere Evaluation zwei öffentlich zugängliche Pakete von Apache. Die beiden Programme verfügen über ungefähr die gleiche Gesamtzahl Lines of Code. Der kleinere Paket, welches wie im weiteren als *Modell 1* bezeichnen, heißt 'commons-cli-1.4-src', hat 6.681 Lines of Code, besteht aus 50 Java-Dateien, 56 Klassen , 354 Methoden und 3122 Beziehungen zwischen diesen, die direkte Strecke der Rückwege Betrug bei Metrik A 23841,98 UEE und bei Metrik B 67544,7 UEE. Das von uns als *Modell 2* bezeichnet Paket heißt 'commons-chain-1.2-src', hat 8.010 Lines of Code verteilt auf 97 Java-Dateien, darin 94 Klassen mit 720 Methoden und 5488 Beziehungen, die direkte Strecke der Rückwege Betrug bei Metrik A 65046,16 UEE und bei Metrik B 87826,46 UEE.

[22] Wir möchten an dieser Stelle allen danken, die sich dazu bereit erklärten und uns ihre Zeit für uns verwendet haben. DANKE!

Kategorie	Typ	Beschreibung
OS		
	Operating System	Microsoft Windows 10 Professional (x64)
	Build	15063.608 (RS2)
CPU		
	Processor Name	Intel Core i5-7600K
	CPU Brand Name	Intel(R) Core(TM) i5-7600K CPU @ 3.80GHz
	Current Clock	3800 MHz
	Number Of Processor Cores	4
Motherboard		
	Model	ASUS Z170-P
	Chipset	Intel Z170 (Skylake PCH-H)
Memory		
	Total Size	16 GBytes
	Total Size [MB]	16384
	Type	DDR4 SDRAM
PCI Express		
	Device Name	Intel Skylake - PCI Express x16 Controller
	Version	3.0
	Current Link Width	16x
	Maximum Link Speed	8.0 GT/s
Graphic Card		
	Video Chipset	NVIDIA GeForce GTX 1070
	Video Chipset Codename	GP104-200
	Video Memory	8192 MBytes of GDDR5 SDRAM [Micron]
	Processor Clock	1556.5 MHz
	Video Unit Clock	1404.5 MHz
	Memory Clock	2003.4 MHz (Effective 8013.6 MHz)

Tabelle 4.5 Hardwarevoraussetzungen

Kategorie	Typ	Beschreibung
Monitor		
	Name	Acer [Unknown Model: ACR03F0]
	Name (Manuf)	B326HUL
	Max. Vertical Size	40 cm
	Max. Horizontal Size	71 cm
	Resolution	2560x1440, Pixel Clock 241.50 MHz
Mouse		
	Name	Logitech B100 Black Optical USB Wired Mouse
	P/N	810-003656
	Max. DPI	800
Keyboard		
	Name	CHERRY Cymotion Master Linux Keyboard
	P/N	G86-21070EUAAAC
HMD		
	Name	HTC Vive
	Resolution	2160×1200 (1080×1200 per eye)
	Refresh rate	90 Hz
	Field of view (Nominal)	About 110 degrees
	Tracking System	Lighthouse (2 base stations emitting pulsed IR lasers)
	Controller input	SteamVR wireless motion tracked controllers

Tabelle 4.6 Peripheriegeräte

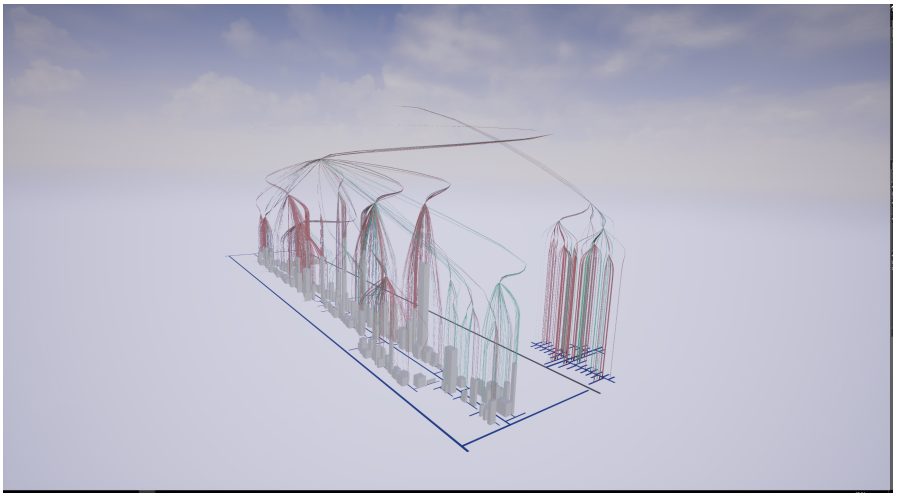


Abbildung 4.1 Evaluationsmodell 2

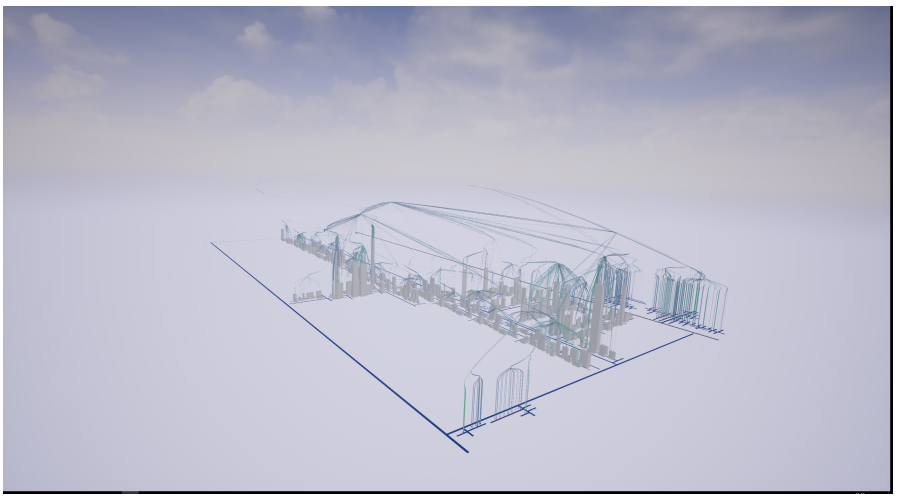


Abbildung 4.2 Evaluationsmodell 2

Da wir die Orientierung der Probanden im Modell nicht durch visuelle Hinweise zu stark erleichtern wollen, halten wir das eingesetzte Modell sehr spartanisch: es besteht aus gleichförmigen Straßenmodell, kastenförmigen Hausmodellen und röhrenartigen Verbindungen. Der Probanden sollen sich nicht Farbe oder Texturen, sondern nur an der Struktur der Darstellung orientieren können.

Kapitel 5

Ergebnisse [MOR]

In diesem Kapitel wollen wir kurz die Ergebnisse unserer Evaluation darstellen. Wir gehen dazu zunächst auf unsere Evaluationsdemographie, anschließend auf die quantitativen und qualitativen Daten ein.

5.1 Demographie [MOR]

Unsere Evaluation fand zwischen dem 5. und 8. Oktober 2017 an der Universität Bremen statt; insgesamt nahmen 15 Probanden teil. Von diesen waren 3 weiblich und 12 männlich. Drei der Probanden mussten wegen Übelkeit oder Augen-Problemen ihre Versuche vorzeitig abbrechen. Die Daten dieser Probanden fließen nicht in unsere Ergebnisse ein. Letztendlich bleiben 12 Teilnehmer, deren Daten wir in unserer Auswertung berücksichtigen. Von diesen sind 2 weiblich und 10 männlich.

Die meisten Probanden waren Studenten der Informatik oder anderer technischer Studiengänge. Weiterhin befand sich ein wissenschaftlicher Mitarbeiter unter ihnen.

Die Altersverteilung der berücksichtigten Probanden stellt sich wie in Abbildung 5.1 dar: die größte Gruppe war im Alter zwischen 21 und 30 Jahren. In Abbildung 5.2 haben wir gemäß Selbstauskunft die Vorerfahrungen der Probanden im Umgang mit Computern, Desktop-Computerspielen sowie Head Mounted Display-Systemen aufgeführt: Die Probanden haben alle Erfahrungen im Umgang mit Computern und nutzen solche fast alle mehrmals wöchentlich mehr als 1 Stunde täglich. Rund die Hälfte der Probanden nutzen 3D-Grafikprogramme und/oder Desktop-Computerspiele mindestens 1 Stunde je Woche; mit Head Mounted Display-Systemen hat keiner der Probanden regelmäßige Vorerfahrungen.

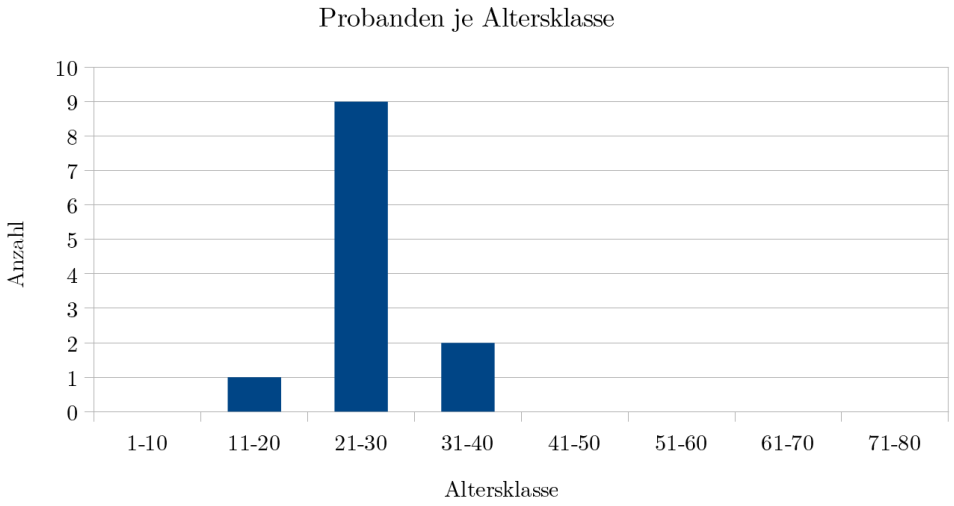


Abbildung 5.1 Altershäufigkeit der Probanden in 10-Jahres-Gruppen

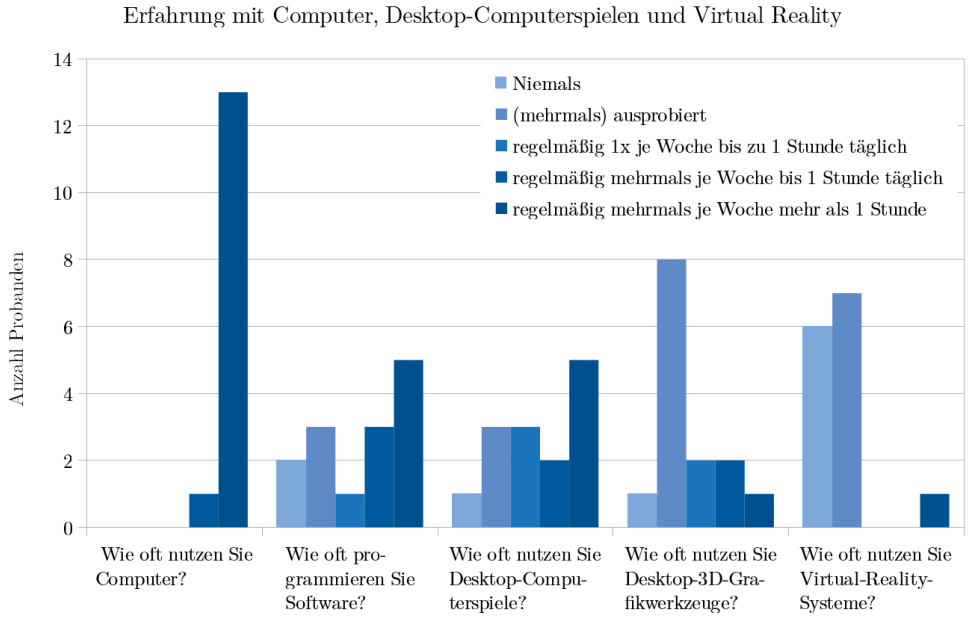


Abbildung 5.2 Erfahrungen der Probanden mit Computern, Desktop-Computerspielen und Virtual Reality

Bezüglich Ihrer Orientierungsfähigkeit in fremden Umgebungen[1] schätzten sich die meisten Probanden als „Eher gut“ ein. Zwei Probanden schätzen sich sogar als „Sehr gut“ und zwei andere hingegen als „Eher schlecht“ ein.

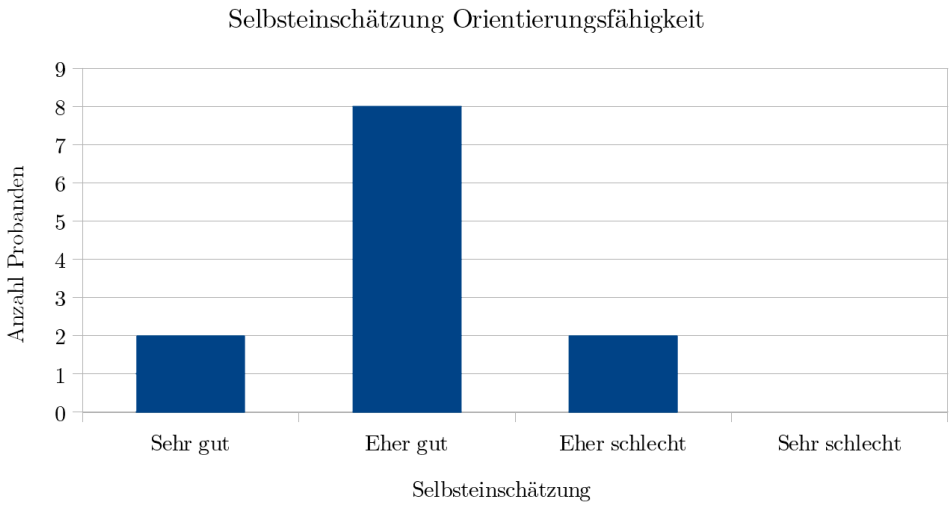


Abbildung 5.3 Selbsteinschätzung Orientierungsvermögen an unbekannten Orten

5.2 Quantitative Daten [MOR]

Für die quantitative Auswertungen definierten wir zwei abhängige Variablen für unsere Evaluation, welche durch Messung in Abhängigkeit von den unabhängigen Variablen[2] bestimmt wurden. Wir gehen im Folgenden zunächst auf die Effektivität, dann auf die Effizienz und schließlich noch auf weitere erhobene, quantitative Daten ein.

5.2.1 Effektivität [MOR]

In Kapitel 4.4 definierten wir Effektivität als Zielerreichung. In Abbildung 5.4 haben wir die Ergebnisse unserer Effektivitätsermittlung dargestellt. Es ist zu erkennen, dass die allermeisten der Probanden mit Ihrer Orientierungsstrategie den Weg zurück zum Startelement fanden. Für

[1] wie im Muster-Fragebogen im Anhang A.2 zu sehen ist, geben wir mit der Fragestellung auch zwei Beispiele, so dass die Probanden ein mentales Bild bekommen sollten oder auch eine konkrete erlebte Situation rezipieren können sollten.
[2] siehe Kapitel 4.2

die nicht erfolgreichen Versuche konnten wir keine Systematik erkennen: Probanden verloren an verschiedenen Abschnitten des Versuchs die Orientierung und gaben auf.

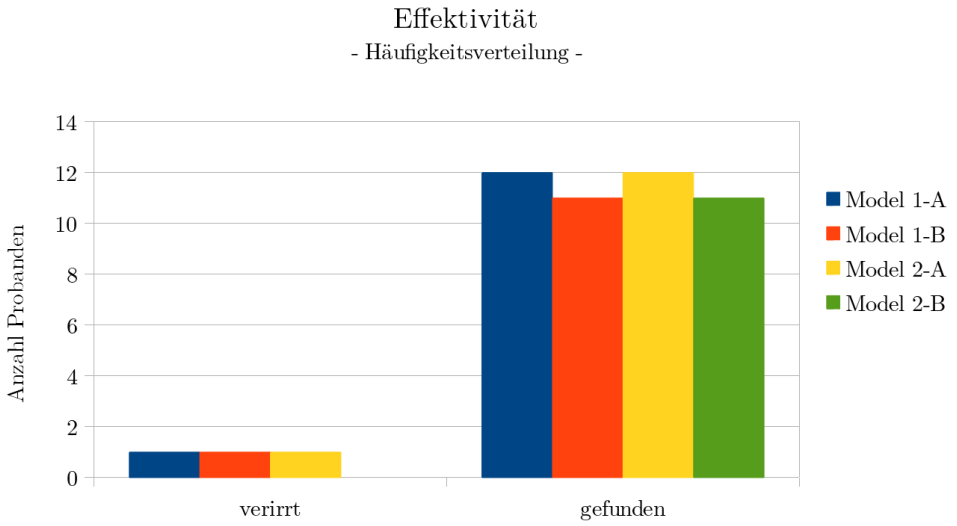


Abbildung 5.4 Quantitative Auswertung: Effektivität

5.2.2 Effizienz [MOR]

In Kapitel 4.3 definierten wir Effizienz als invertierte Flugdauer zurück zum Ausgangspunkt und rechneten sie in eine Geschwindigkeit (in $[UEE/s]$) um. Im Folgenden stellen wir dar, wie sich die gemittelte und die mediierte Effizienz unserer Probanden durch Änderung der unabhängigen Variablen – das sind: das System, das Modells und die Modellmetrik – ändert. Grundsätzlich lässt sich feststellen, dass die linearen Standardabweichungen der Mediane und der Mittelwerte deutlich voneinander abweichen. Die Standardabweichungen des Mittelwerts sind häufig höher als linearen, mittleren Abweichungen des Medians.

Wir werteten[3] die Zeiten, die Fehlversuche sowie die eigentliche Zielerreichung aus. Durch die wechselnde Reihenfolge der Versuchsabfolge[4] hatten wir somit insgesamt drei unabhängige Variablen, auf denen wir unsere Auswertung gründen: Die Umgebung (Desktop oder Head Mounted Display), die Visualisierung (Modell A oder B) und die Versuchsreihenfolge (Desktop oder Head Mounted Display als erstes).

[3] wie bereits beschrieben, siehe Kapitel 4
[4] Jeder 1te Proband begann mit dem Desktop-System, jeder 2te Proband mit dem Head Mounted Display-System, siehe dazu Kapitel 4.5.3

5.3 Effizienzvergleich zwischen Desktop- und Head Mounted Display-System [MOR]

In Abbildung 5.5 haben wir die gemittelten Ergebnisse der Effizienzberechnung des Vergleichs von Desktop- und Head Mounted Display-System dargestellt, in Abbildung 5.6 die gleichen Basisdaten noch einmal mediiert. Auch wenn die Tendenzen für beide Berechnungen ähnlich sind, lässt sich doch feststellen, dass die Mittelwerte selbst und deren Standardabweichung für beide Systeme höher sind als die mediierten Werte.

Die Mittelwerte beim Desktop-System sind grundsätzlich höher als die mediierten Werte. Sowohl die Mittelwert als auch der Median zeigen, dass das Modell 2 schneller im Desktop-System genutzt wurde, während das Modell 1 vom Mittelwert her schneller mit dem Head Mounted Display beflogen wurde, während mediiert das Desktop-System in Modell 1-A besser abschnitt.

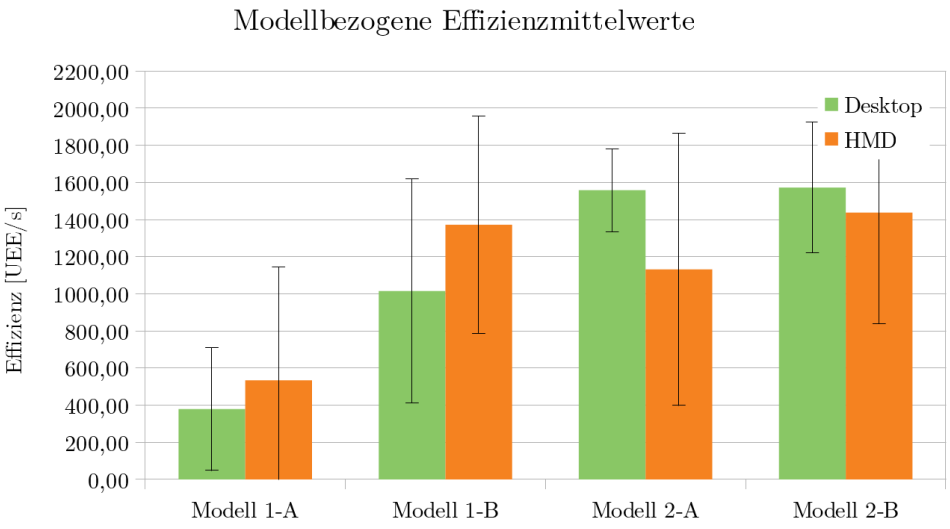


Abbildung 5.5 Quantitative Auswertung: modellbezogene Effizienzmittelwerte und mittlere, lineare Abweichung (Standardabweichung)

5.4 Effizienzvergleich zwischen Modell 1 und 2 [MOR]

In Abbildung 5.7 haben wir die gemittelten Ergebnisse der Effizienzberechnung im Vergleich der beiden genutzten Modelle dargestellt, in Abbildung 5.8 die gleichen Basisdaten noch einmal mediiert.

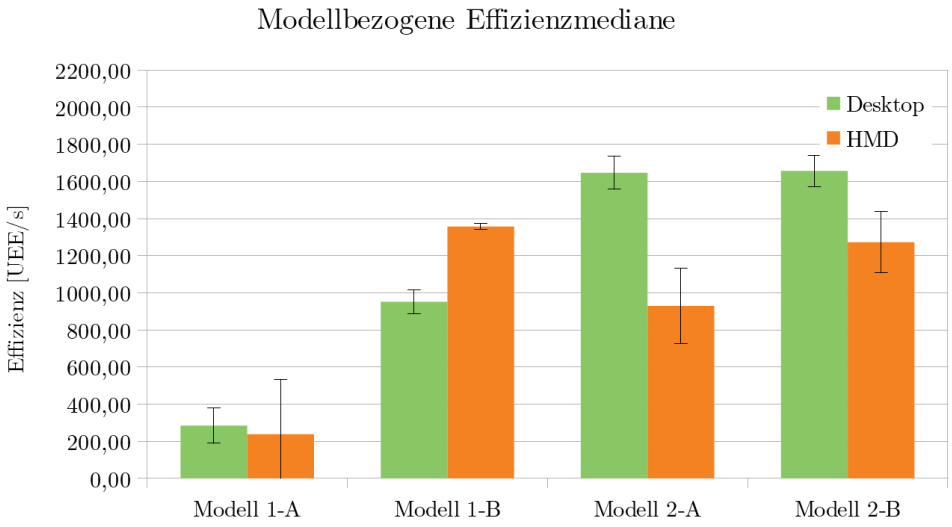


Abbildung 5.6 Quantitative Auswertung: modellbezogene Effizienzmediane und mittlere, lineare Abweichung

Unabhängig von der betrachteten Metrik stellen wir fest, dass das Modell 2 schneller als das Modell 1 bereist wurde. Weiterhin können wir sehen, dass sich für die Metrik B in allen Fällen wesentlich höhere Geschwindigkeiten nachweisen lassen.

Unsere Probanden scheinen das Modell 1 mit Metrik A interessanterweise schneller mit dem Desktop-System bereisen zu können und in der VR langsamer zu sein, das dreht sich bei Modell 1 mit Metrik B allerdings um. Im Modell 2 ist das Head Mounted Display-System in allen Fällen langsamer.

Bis auf die wieder sehr große Standardabweichung in den Mittelwerte sind die Daten beider Berechnung tendenziell vergleichbar.

5.5 Effizienzvergleich zwischen Metrik A und B [MOR]

In Abbildung 5.9 haben wir die gemittelten Ergebnisse der Effizienzberechnung im Vergleich der beiden genutzten Modellmetriken dargestellt, in Abbildung 5.10 die gleichen Basisdaten noch einmal mediiert.

Auch im Vergleich der Modellmetriken ist der Unterschied zwischen den Medianen und den Mittelwerten gering. Allerdings fällt auch hier die sehr hohe Standardabweichung bei den Mit-

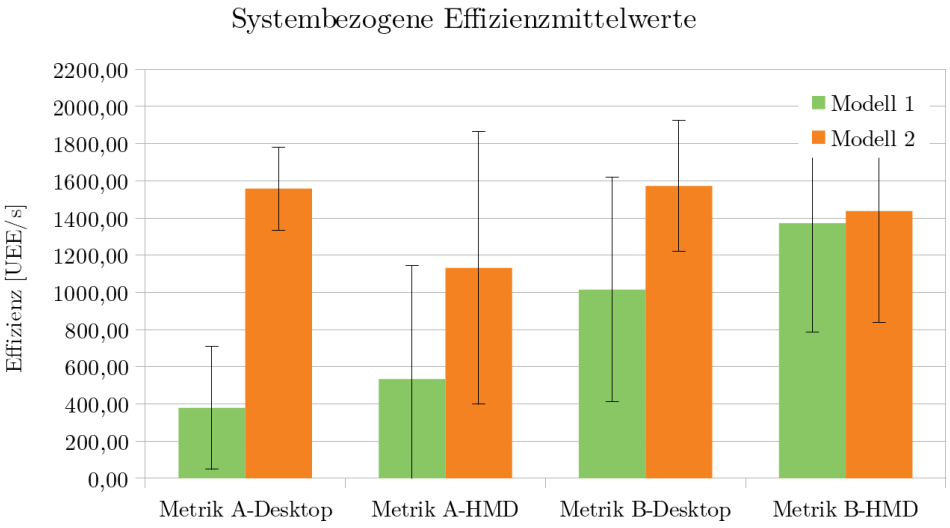


Abbildung 5.7 Quantitative Auswertung: systembezogene Effizienzmittelwerte

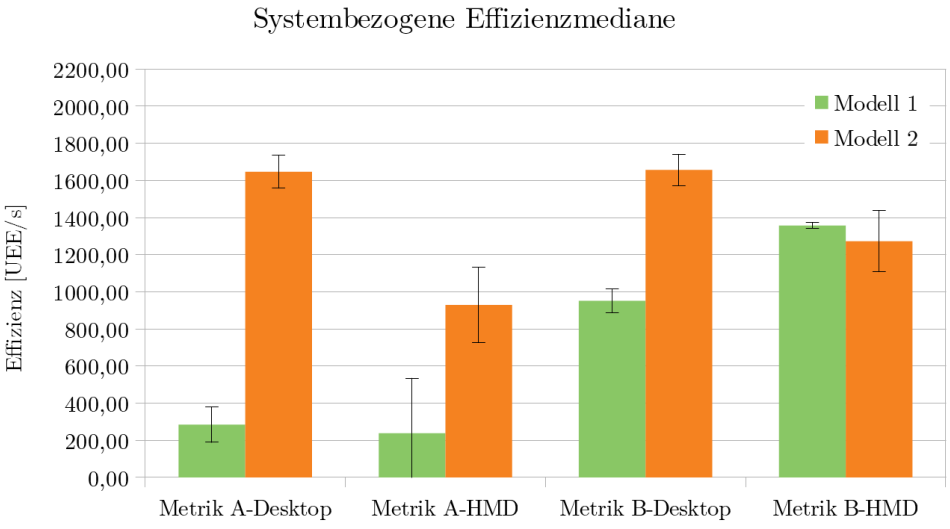


Abbildung 5.8 Quantitative Auswertung: systembezogene Effizienzmediane

telwerten auf. In der Metrik B erreichen die Probanden im Schnitt z. T. deutliche höhere Geschwindigkeiten als in der Metrik A.

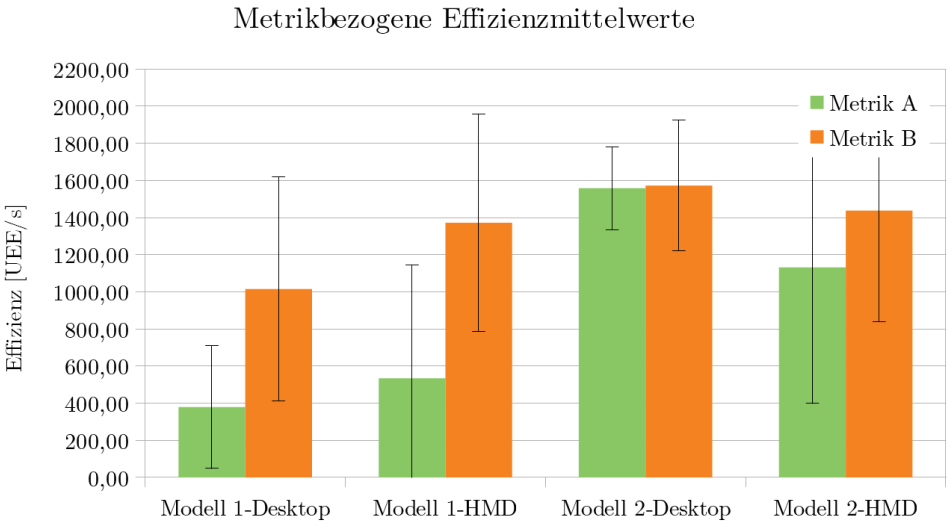


Abbildung 5.9 Quantitative Auswertung: metrikbezogene Effizienzmittelwerte

5.6 Zusammenfassender Vergleich über alle Versuche und Probanden [MOR]

Wenn wir unserer oben für die drei unabhängigen Variablen dargestellten Wert soweit über alle Versuche zusammenzufassen, dass wir nur noch die Ergebnisse des Desktop- mit denen des Head Mounted Display-System vergleichen, so erhalten wir die in Abbildung 5.11 dargestellt Grafik. Hierin unterscheidet sich das Desktop- so gut wie gar nicht vom Head Mounted Display-System, unabhängig davon, ob wir nun den Median oder den Mittelwert berechnen. Allerdings liegen auch hier Standardabweichungen für die Mittelwerte bei rund $+-50\%$.

5.7 Fehlversuche und Wege [MOR]

Wir protokollierten während des Versuchs die Anzahl der Fehlversuche der Probanden, um zu entscheiden, ob die Orientierungseffizienz allein aufgrund einer besseren individuellen Orientierung oder durch Glück zustande kam. Die Ergebnisse sind in Abbildung 5.12 zu finden.

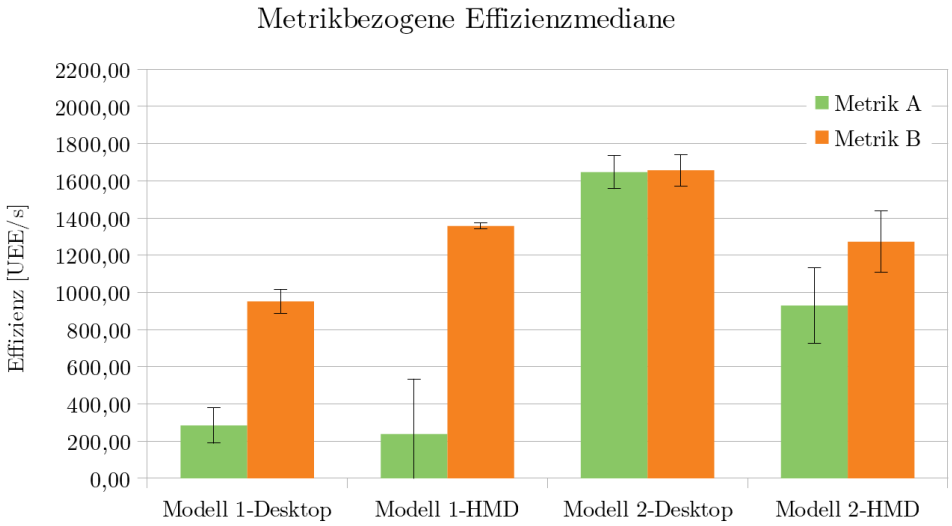


Abbildung 5.10 Quantitative Auswertung: metrikbezogene Effizienzmediane

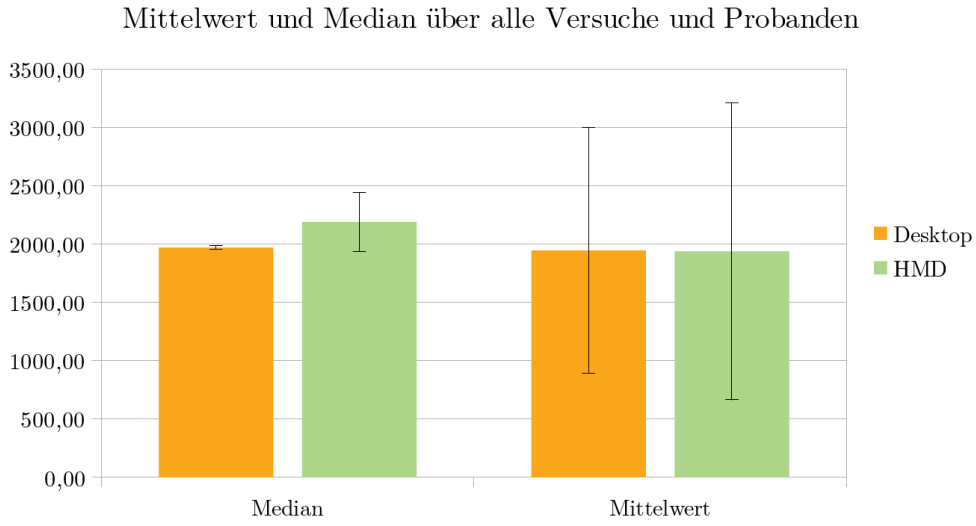


Abbildung 5.11 Quantitative Auswertung: Median und Mittelwert Effizienzmittelwerte

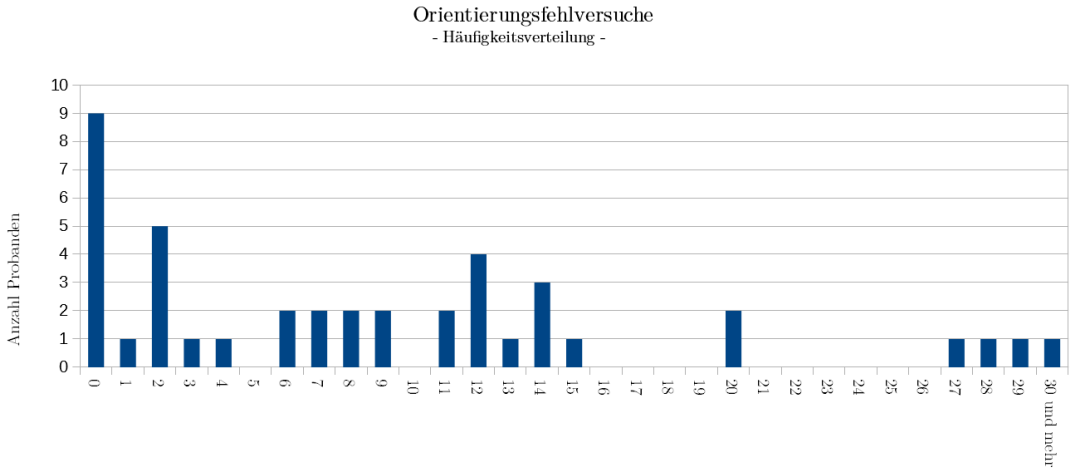


Abbildung 5.12 Fehlversuchshäufigkeit

Es ist zu erkennen, dass die weitaus größte Gruppe diejenigen sind, die ohne oder mit bis zu 3 Fehlversuchen den Weg zum Startelement zurück fanden. Darüber hinaus gab es auch Probanden, die offenbar durch Versuch und Irrtum das richtige Element wiederfanden. Einige brauchten dazu fast 40 Versuche.

Neben der reinen Anzahl der Versuche protokollierten wir auch grob den Weg, den die Probanden auf ihrem Rückweg einschlugen. Die Ergebnisse sind in den Abbildungen 5.13 und 5.14 grafisch als Overlay aller Probanden wiedergegeben. Aus den Abbildungen lässt sich grob erkennen, dass es offenbar ganz unterschiedliche Orientierungsstrategien gab. Manche Probanden überraschten selbst uns und flogen geradlinig direkt auf das Zielelement zu, während andere zunächst in eine vollkommen falsche Richtung flogen, dann aber anscheinend etwas wieder erkannten und das Zielelement auswählten. Wieder andere versuchten das System hinter der Benennung der Elemente zu entdecken, hatten sich offenbar die Nummer des Startelements gemerkt und suchten verschiedene Elemente ab. Einmal eingeschlagen behielten sie ihre Strategie alle weiteren Durchgänge bei.

5.8 Qualitative Daten [MOR]

You can observe a lot by watching.

— Yogi Berra

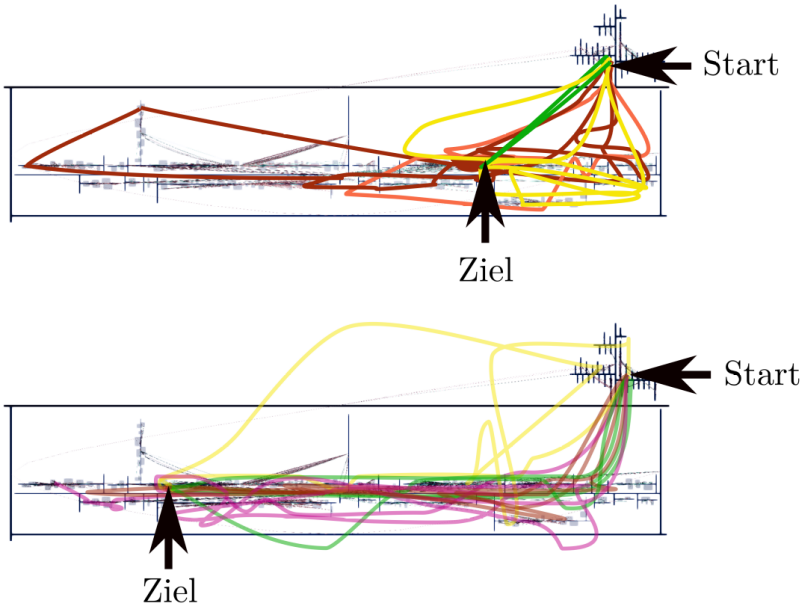


Abbildung 5.13 Wege im Modell 1-A (o) und 1-B (u)
Legende:
- 0 Fehlversuche
- 1-5 Fehlversuche
- 6-10 Fehlversuche
- 11-15 Fehlversuche
- 16-25 Fehlversuche
- 26+ Fehlversuche

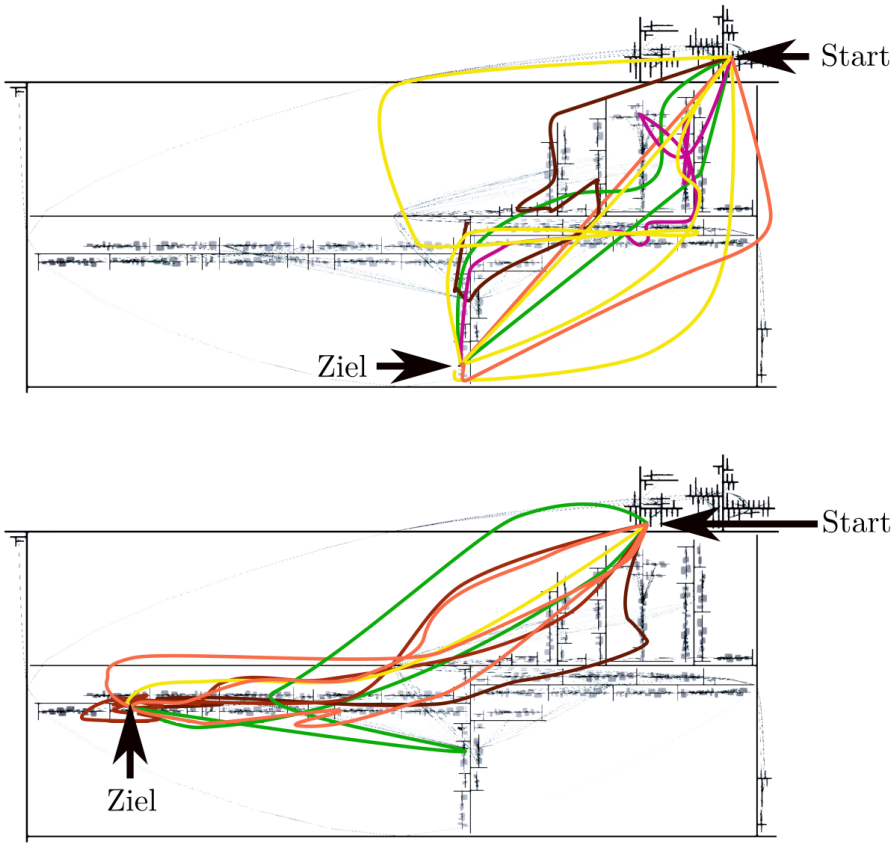


Abbildung 5.14 Wege im Modell 2-A (o) und 2-B (u)
Legende:
- 0 Fehlversuche
- 1-5 Fehlversuche
- 6-10 Fehlversuche
- 11-15 Fehlversuche
- 16-25 Fehlversuche
- 26+ Fehlversuche

Nach unseren Beobachtungen nutzten die Probanden die Freiheitsgrade in der Head Mounted Display-Umgebung nur teilweise aus. Während der Versuche gab es immer wieder Situationen, in denen das Info-Display nicht vollständig zu sehen war, da es sich automatisch versucht, zum Probanden auszurichten, dabei aber keine Rücksicht auf die anderen Elemente der Szene nimmt. Dies hatte zur Folge, dass die Beschriftung teilweise abgedeckt wurde und der Proband sich drehen oder bewegen musste. Im Head Mounted Display-System wären solche Situationen leicht durch eine kurze Bewegung des Körpers oder eine Blickrichtungsänderung möglich gewesen, allerdings nutzten nur zwei Probanden diese Option und dass auch nicht in allen Fällen. Körperdrehungen wurden von den Probanden hingegen häufig eingesetzt.

Die Probanden verfolgten unterschiedliche Strategien zur Orientierung. Wir konnten während der gesamten Evaluation keinen Strategiewechsel der Probanden zwischen der Desktop- und der VR-Umgebung feststellen: Haben sich die Probanden erst einmal eine Strategie eingeschlagen, hielten sie während des gesamten Versuchs daran fest. Lediglich einen Übergang zwischen keiner Strategie (also der Erkenntnis, dass man am Ende tatsächlich ohne Hilfe zurück finden soll) und einer Strategie konnten wir feststellen. Diese Feststellung trafen die Probanden in den entsprechenden Fällen im jeweils ersten Versuch: Probanden, die mit der Maussteuerung begonnen im ersten Teil der Maussteuerung, Probanden, die mit dem Head Mounted Display begonnen in der Head Mounted Display.

Von den oben[5] aufgeführten Orientierungstypen und Orientierungsmustern nutzten die meisten Probanden die Try&Error-Strategie. Wir hatten den Eindruck, dass es einigen Probanden wichtiger war, möglichst spielerisch schnell von einem Element zum anderen zu kommen und die Orientierungsaufgabe dabei in den Hintergrund rückte.

5.8.1 meCue-Fragebogen [MOR]

In den Abbildungen 5.15 bis 5.17 haben wir die Ergebnisse unserer meCue-basierten[6] Probandenbefragung dargestellt. Danach nahmen unsere Probanden beide Systeme als eher positiv wahr, fühlten sich aber nicht an sie gebunden oder in ihrem Status erhöht. Die Benutzbarkeit werteten die Probanden als eher gut. Die Nützlichkeit der Systeme wurde von den Probanden im mittleren Bereich gewertet. Das Model III (Konsequenzen) haben wir in der Auswertung in berücksichtigt. Die Standardabweichungen bei rund 20 - 30%.

Betrachten wir uns die Emotionen, die die Probanden mit den Systemen verbinden, so sind mehr positive als negative Emotionen vorhanden.

Das Gesamturteil unserer Probanden fiel für beide Systeme sehr positiv aus, wobei das Head Mounted Display-System einen leichten Vorsprung hat.

[5] siehe Tabelle 4.1 und Tabelle 4.2
[6] siehe Kapitel 2.6.7

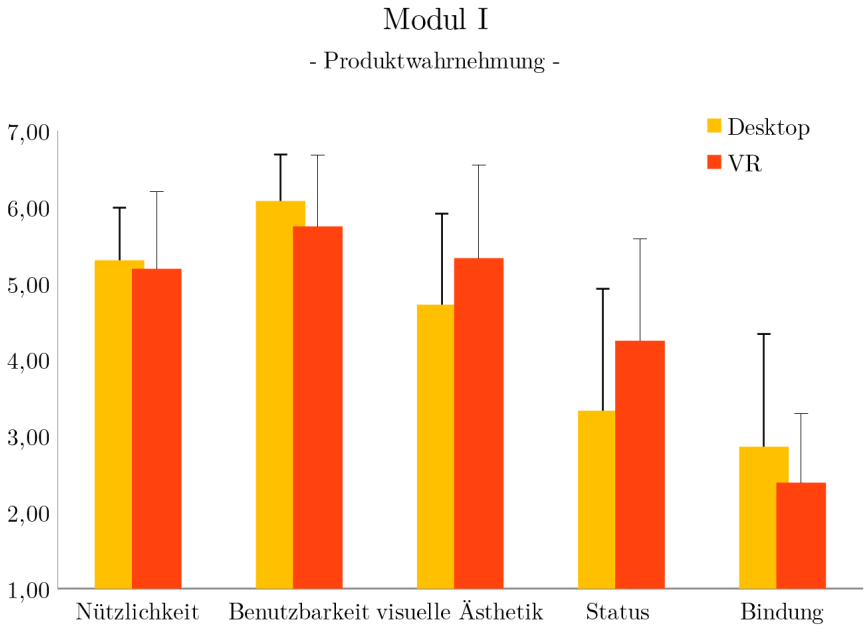


Abbildung 5.15 meCue: Modul 1 – Produktwahrnehmung

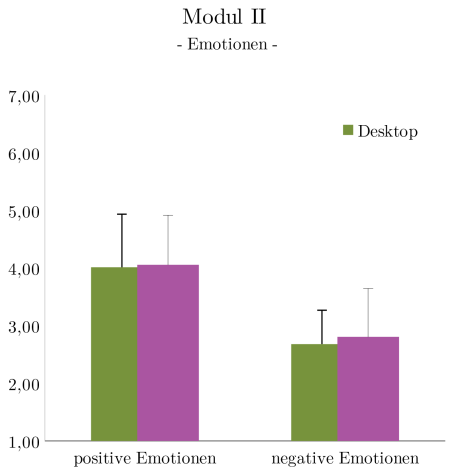


Abbildung 5.16 meCue: Modul 2 – Emotionen

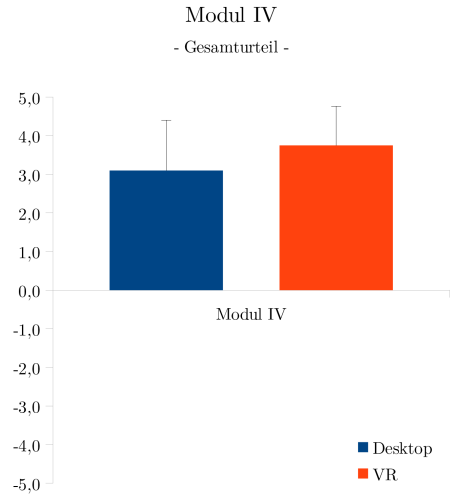


Abbildung 5.17 meCue: Modul 4 – Gesamturteil

5.8.2 SUS-Fragebogen [MOR]

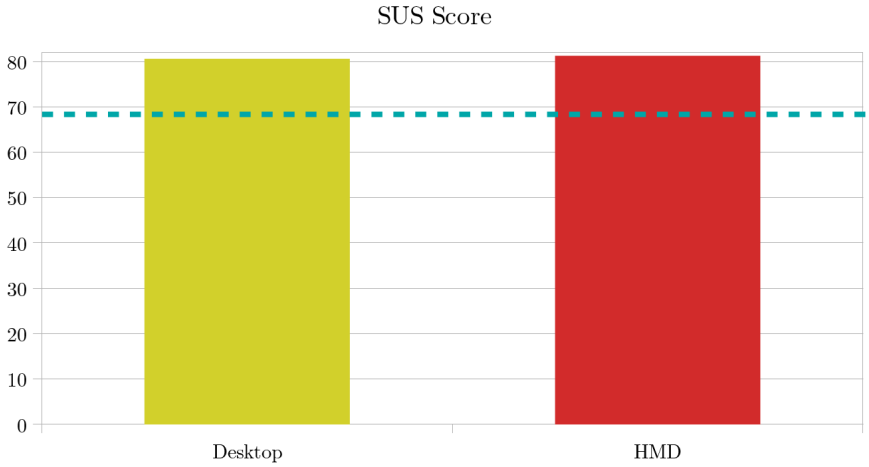


Abbildung 5.18 SUS Ergebnisse

In der Abbiuldung 5.18 haben wir die Ergebnisse aus dem 'Quick-&-Dirty'-Fragebogen Software Usablity Scale aufgeführt. Der Durchschnitt bei Software Usablity Scale-Prüfungen liegt laut Internetseite bei 68 Punkten. Unsere Systeme erhielten mit über 80,3 Punkten die Note 'A', sind daher nach dem Bewertungsschema dieses Fragebogens beide sehr gut benutzbar.

Kapitel 6

Diskussion [MOR]

Ein womöglich entscheidender optischer Unterschied zwischen den Modellen war, dass Modell 1 hatte zwar weniger Klassen und Methoden als Modell 2 aber die ungefähr gleiche Anzahl von Lines of Code. Da wir die Lines of Code auf die Höhe der Gebäude auftrugen, war Modell 1 insgesamt höher als Modell 2. Die Modelle unterscheiden sich bzgl. der Gebäudehöhe ein wenig wie ein Gewerbegebiet von einem Bankenzentrum.

6.1 Demographie [MOR]

Unsere Probanden waren zum größeren Teil Studierende der Informatik und somit sowohl computer- als auch computerspieleerfahren. Diese Auswahl der Probanden wird großen Einfluss auf unser Versuchsergebnis gehabt haben. Wir vermuten, dass sich hierin auch die große Standardabweichung in allen Auswertung begründet. Wir nehmen von diesen Probanden an, dass sie mit der `WASD`-Steuerung besser als die anderen zurecht kamen.

Die Altersgruppierung zeigt, dass sich recht junge Studenten für die Evaluationen zur Verfügung gestellt haben; Studenten aus einer Generation, die mit Technik aufgewachsen ist und insbesondere Navigationstechnik im täglichen Leben selbstverständlich nutzen. Diese Navigationstechnikaffinität könnte auch dafür gesorgt haben, dass die Probanden die Orientierungsaufgabe schlechter als möglich ausgeführt haben. Ein Proband brachte es zu Beginn seines Versuchs auf den Punkt: „Orientieren... Mit Navigationsgerät?“[1]

Wir gehen bzgl. der angegebenen Vorerfahrungen davon aus, dass die Probanden aufgrund des Rufs der Computerspielerei geringere Zeiten für die tägliche Nutzung von Computerspielen angegeben haben könnten, als tatsächlich anfallen. Die Fragen nach den Vorerfahrungen in den anderen Bereichen schätzen wir als unverfänglich und damit wirklichkeitskonform ein.

[1] sinngemäßes Zitat

Wir vermuten, dass die Effizienz, so wie wir sie berechnen, eine Gaußverteilung bei genügend großer Datenmenge ergeben würde. Dies müsste in einer anderen Evaluation untersucht werden. Hieraus könnten dann auch mittels zweiseitigem t-Test eventuell signifikante Unterschiede zwischen den Versuchsreihen vermittelt werden. In unserem Fall kommen wir je Versuchsreihe auf 5 bzw. 7 Probanden; damit sind wir weit ab von der empfohlenen Probandenzahl für eine t-Test. Perry, Porter und Votta (2000) schlagen zudem vor, besser die Aussage darüber, ob die Daten signifikant sind, dem Leser zu überlassen und nur die Fehlerwahrscheinlichkeiten anzugeben.

6.2 Quantitative Daten [MOR]

Wir sind uns aufgrund der Gruppengröße uneinig, ob besser die gemittelten oder die mediierten Werte genutzt werden sollten, da die Ausreißer einerseits einfach nur Ausreißer sein könnten, andererseits aber auch einen Hinweis auf eine Nutzergruppe geben könnte, die wir durch unsere Probanden nicht abgedeckt haben.

Die Abweichung zwischen den gemittelten und den mediierten Wertem deuten wir so, dass wir in den Ergebnissen ein große Spreizung haben. Diese kommt in den gemittelten Werte stärker zum Ausdruck und findet sich auch in der recht hohen Standardabweichung wieder. Übertragen auf die Probanden bedeutet dies, dass die Fertigkeiten im Umgang mit der beiden Systemen bei den Probanden unterschiedlich verteilt waren.

Es lässt sich feststellen, dass das Modell 2 – obwohl größer – einfacher zu bereisen war. Dies könnte daran liegen, dass im zweiten Modell die Gebäude deutlich kleiner waren und auch die Verbindungen tiefer lagen, was zu häufigeren Blicken auf die Struktur des System führte, auch bot das System nach unserem Empfinden deutlicher mehr strukturelle Anhaltspunkte. Dadurch entfielen eventuell im Modell 2 Orientierungsphasen, die in Modell 1 notwendig waren.

In Modell 2 ist die Reisegeschwindigkeit mit dem Desktop-System auch höher als in Modell 1. Dies könnte daran liegen dass alle Probanden die Modell 2 als Desktop-System bearbeitet haben, Erfahrungen und Strategien mit nehmen und voll einsetzen konnten, da sie tendenziell eine gewohnte Steuerung benutzten. Das die Geschwindigkeit zwischen Modell 2 mit Metrik A und Metrik B nur noch kaum steigt, würden wir so interpretieren, dass eine weitere Steigerung nicht mehr möglich ist (ohne die Strategie zu ändern). Das die Reisegeschwindigkeit für Probanden, die mit dem Desktop-System gestartet sind sich von Modell 1-A zu 1-B steigerte verstehen wir als Lerneffekt. Der Einbruch in der Reisegeschwindigkeit zum ersten Modell dieser Probanden mit dem Head Mounted Display-System könnte auf die ungewohnte Steuerung zurück zu führen sein, die es erschwert Erfahrungen aus dem Desktop-System zu übertragen. Den verhältnismäßig großen Unterschied zwischen Median und Mittelwert für die Probanden die mit dem Head Mounted Display-System gestartet sind, würden wir über den doppelten Lernaufwand erklären.

Die Probanden mussten sich sowohl zum ersten mal in die Aufgabe in einem echten Szenario erledigen als auch die Steuerung weiter erproben. Dementsprechend würden die Ergebnisse je nach Lerntyp und vor Erfahrung hier am stärksten auseinander gehen.

Im Vergleich der beiden Modellmetriken fällt auf, dass die Versuche der Modellmetrik B stets schneller bereist wurden als die Versuche der Metrik A. Einer der Gründe könnte sein, dass in beiden Fällen die Modelle identisch und die Entfernung zwischen Start- und Zielelement in dem Modellmetriken B weitaus größer als in der Modellmetrik A waren. Durch die längere Strecke, würde sich bei einem direkten Flug mit kurzen Orientierungsunterbrechungen ein geringer Einfluss durch einzelne Unterbrechungen auf die gesamt Geschwindigkeit ergeben. Außerdem würden wir dies auf Lerneffekte zurück führen. Von allen Probanden stellte allerdings nur einer Ähnlichkeiten zwischen den beiden Modellmetriken A und B fest, alle anderen waren im Nachgespräch verwundert darüber, dass die Modell identisch gewesen sein sollen.

Wir können in den Daten außerdem einen Anstieg der Effizienz bis zu einem gewissen Grenzwert sowohl für Desktop- als auch Head Mounted Display-System vermuten. Dies könnte bedeuten, dass die Übungsphase oder auch das Übungsmodell zu kurz resp. zu klein war.

Die Berechnungen zeigen trotz der zuvor diskutierten Unterschiede so gut wie keinen Unterschied zwischen der gemittelten und der mediierten Zusammenfassung. Lediglich fällt eine wesentlich höhere Standardabweichung für den Mittelwert auf.

6.3 Wege [MOR]

Der Darstellung der Wege ist zu entnehmen, dass es offenbar strukturell-visuelle „Leitlinien“[2] gibt, die die Orientierung im Modell erleichtern und offenbar den Probanden (unbewusst) in Erinnerung bleiben. Im Modell 2 kommen offenbar andere Orientierungsaspekte zum Tragen: Die eingeschlagenen Wege sind weitläufiger und es lassen sich keine deutlichen 'Hauptstraßen', die genutzt werden erkennen. Wir würden vermuten, dass die daran lag, dass die Probanden den Verbindungen folgen sollten und da diese im Modell 1 sehr hoch hinaus liefen, hatte sie weniger Chancen einen Blick auf die Stadt zu erhalten. Wir nehmen an, dass die Orientierung in Modell 1 einerseits durch die starke Strukturierung durch hohe Gebäude einfacher war, andererseits aber die hohen Häuser auch die Sicht auf dahinter liegende Häuser verdeckte und Verbindungsstrukturen als Orientierungsmerkmal stärker ausschloss. Dadurch war es in Model 2 leicht möglich durch leichtes Steigen einen guten Überblick über die Stadt zu erhalten, während die Probanden in Modell 1 vergleichsweise hoch steigen mussten und dort fraglich war, ob sie die Strukturen der Stadt wiedererkennen würden.

[2] Formulierung eines Probanden

6.4 Qualitative Daten [MOR]

Uns fiel auf, dass ein viele Probanden sehr spielerisch mit der Situation des Fliegens umgingen.[3] Sie schienen einen Reiz darin gefunden zu haben die Elemente der Voraufgaben möglichst schnell zu erreichen. Fraglich ist für uns daran, ob sie die Aufgabe so schnell erledigen wollten, weil sie diese Art der spielerischen, orientierungsfreien Fortbewegung besonders gut kannten, oder weil sie die Aufgabe nicht für wichtig genug hielten.

6.4.1 Orientierung [MOR]

Die Orientierungsfähigkeit der Probanden haben wir über eine Ordinalskala abgefragt. Die Werte der Skala sind weder inter- noch intrapersonel vergleichbar. Insofern ist die Aussage der Probanden zu ihrer Orientierungsfähigkeit in fremdem Umgebungen mit Vorsicht zu genießen und nur als grundsätzlicher Maßstab zu verstehen. Konkret konnten wir bei einem Probanden mit einer nach unseren Maßstäben ausgezeichneten Orientierung feststellen, dass seine Selbsteinschätzung auf „Eher gut“ lautete.

Im Grundlagenteil[4] haben wir angeführt, dass Orientierungslernen gemäß dem Stand der Forschung offenbar eine erlernbare Fertigkeit ist. Gleichzeitig hielten wir aber auch fest, dass diese Fertigkeit in der modernen Gesellschaft durch die hohe Verfügbarkeit von Navigationsgeräten verkümmert. Leider haben wir in unserer Teilnehmerbefragung diesen Aspekt nicht abgefragt, müssen aber aufgrund der Altersstruktur und der Technikaffinität davon ausgehen, dass sämtliche Teilnehmer selbstverständlich im Alltag Navigationsgeräte zur Wegfindung einsetzen.

Die Teilnehmer setzten nach unseren Beobachtungen unterschiedliche Strategien zu Orientierung ein, wobei der Try&Error-Strategie am weitesten verbreitet war. Einige Probanden flogen aber auch mit einer Ruhe zum Startelement zurück, ohne sich zuvor umgesehen oder für uns anderweitig orientiert zu haben. Auf Nachfrage konnten sie nicht erklären, woher sie wussten, wohin sie mussten.

Leider stellten wir erst im Anschluss an die Evaluation fest, dass offenbar unter bestimmten Bedingungen die Fluggeschwindigkeit im DesktopSystem trotz Justierung bis zu 0,6 mal schneller sein konnte als das Head Mounted Display-System.[5] Wir haben bei der Berechnung der entsprechenden Fälle mit einem konstanten Faktor von 1,5 multipliziert, um diesen Fehler zu nivellieren. Da die meisten Probanden keine kontinuierlich Flugbewegung hatten, haben wir nicht den Faktor 1,6 verwendet.

[3] ein Proband untermalte seinen Flug noch mit einem fröhlichen „Huiii“
[4] siehe Seite 9
[5] Als Quelle des Fehlers offenbarte sich das eingesetzte Unreal Engine-VR-Plugin, in dem die zurückgelegte Strecke je Zeiteinheit nicht an die vergangene Zeit gekoppelt sondern an die Bildwiederholungsrate gekoppelt wurde.

6.4.2 Try&Error [MOR]

Uns wunderte, dass die meisten Probanden eine Try&Error-Strategie mit z. T. über 25 Fehlversuchen verfolgten. Uns scheint hier ein systematischer Fehler oder ein Fehler in der Versuchsbeschreibung wahrscheinlich. Der gesunde Menschenverstand sagt, dass, wenn wir diese Ergebnisse auf die Realität übertrugen, Innenstädte voller Menschen sein müssten, die eine Vielzahl von Hotels auf dem Weg zu ihrem Hotel abklappern oder ein IKEA-Parkplatz voller Menschen sein müsste, die verzweifelt ihr Fahrzeug suchten. Dies deutet für uns darauf hin, dass wir eventuell die Bedeutung der Aufgabe nicht deutlich genug hevorgehoben haben, oder die negativen Begleiterscheinungen von Fehlversuchen im Gegensatz zur Wirklichkeit zu gering waren. Dies ist ein Aspekt, der ggfs. auch in Computerspielen zum Tragen kommt.

Uns wunderte während des Versuchs, dass die Try&Error-Strategie so selbstverständlich eingesetzt wurde. Die Probanden fragten nicht einmal, ob dies erlaubt wäre, sondern setzten es ein. Außerdem wurde diese Strategie bis zum Ende interessanterweise auch von Probanden beibehalten, die in den vorhergehenden Versuchen damit lange brauchten. Wir vermuten, dass die Probanden davon ausgingen es gäbe keine Konsequenzen durch das Ausprobieren, immerhin waren die Probanden nicht informiert worden, dass wir die Anzahl ihrer Versuche messen würden. Auf dem IKEA-Parkplatz sieht es mit der Konsequenzlosigkeit schon ganz anders aus, allein der Moment, wenn andere Menschen merken, dass jemand versucht das falsche Auto aufzuschließen, kann als peinlich empfunden werden. Die Motivation eine fehlerfreie Rückreise zu schaffen, mag unter dem mangel von Konsequenzen gelitten haben.

Das Festhalten an einer einmal eingeschlagenen Strategie könnte auch darauf zurückzuführen sein, dass die Versuche unter Ausschluss von Wettbewerb stattfanden und so die Probanden keinen Vergleich hatten, ob sie gut oder schlecht waren.

In der Häufigkeitsverteilung der Fehlversuche lässt sich erkennen, dass eine sehr große Gruppe keine Fehlversuche hatte, allerdings gab es auch viele Probanden die eine Vielzahl von Fehlversuchen in Kauf nahmen, bevor sie ggfs. aufgaben.

6.4.3 meCue und SUS [MOR]

Die Auswertung der beiden Fragebogensysteme ergab für die kreierten Systeme ein positives bis sehr positives Feedback. Dies mag der Tatsache geschuldet sein, dass die Probanden alle mit uns bekannt war.

Insgesamt hat bei beiden Fragebogensystem das Head Mounted Display einen leichten Vorsprung, der aber offenbar nicht durch eine Statuserhöhung zu erklären ist. Eventuell kommt hier die Neuheit der Technologie für so gut wie alle Probanden zum tragen.

6.5 Komplexität [MOR]

Aufgrund der Datenlage der Effektivitätsberechnung gehen wir davon aus, dass der Versuchsaufbau nicht zu kompliziert gestaltet war. Alle Probanden hatten eine gute Chance, den Weg zum Startelement zurück zu finden und so die Aufgabe zu erfüllen. Weiterhin schließen wir aus den beliebig verteilten verlorenen Orientierungen[6], dass die *Orientierung* an sich in unserem Versuchsaufbau nicht durch Lerneffekte wie in Kapitel 2.1.3 beeinflusst werden.

6.6 Place Illusion [MOR]

Eine der Kernfragen in unserer Evaluation war die Frage, ob der Proband durch die jeweilige Place Illusion, die er erfährt, einen Vorteil bei der Bewältigung der Orientierungsaufgabe hat. In dieser Hinsicht gehen in der Literatur die Meinungen stark auseinander. Wie in Kapitel 2.4.3 angemerkt, werden unsere Ergebnisse auch durch die Interaktionsmöglichkeiten der beiden Systeme beeinflusst sein oder im direkten Zusammenhang in den unterschiedlichen Fidelities bzw. Immersionen bezogen auf die einzelnen Sinne stehen. Wir würden allerdings davon ausgehen, dass die Unterschiede, die den größten Einfluss gehabt haben, erstens die höhere Auflösung am Desktop, die ein Lesen der dargestellten Informationen und verfolgen der Markierung auf höhere Distanz erlaubt, zweitens die stereoskopische Sicht mit dem Head Mounted Display, die im nahen und mittleren Bereich eine bessere Einschätzung der Entfernung und Größen erlaubt, sowie drittens die Art des Umguckens (natürliches Umgucken vs. Handbewegung). Beim Umgucken mit dem Head Mounted Display bietet dieses durch die Stimme des Versuchsleiters oder Unregelmäßigkeiten im Ausmessungsgitter des sicheren Bereichs, eindeutige Hinweise darauf in welche Richtung gerade geguckt wird, was am Desktop nicht geht.

Dennoch wurden trotz der größeren Freiheitsgrade der Head Mounted Display-Umgebung diese von den Probanden nicht im möglichen Maße genutzt. Dies könnte auf die geringere Gewohntheit und eine gewisse Vorsicht im Umgang mit diesem System zurück zu führen sein.

[6] die Probanden, die das Startelement nicht finden konnten und selbst die Orientierungsaufgabe beendeten

Kapitel 7

Zusammenfassung [MOR]

Das Gegenteil von schlecht muss nicht gut sein –
es kann noch schlechter sein.
— Paul Watzlawick

Im Zuge dieser Arbeit ist ein Plugin für die Unreal Engine entstanden mit dem Softwaregraphen visualisiert werden können. Des weiteren haben wir eine Anordnung für eine Stadtmetapher zur Visualisierung implementiert und diese mit hierarchisch gebündelten Abhängigkeiten die aus den Häusern der Stadt kommen erweitert. Diese Visualisierung haben wir soweit optimiert, dass sie bei einer Anzeige von über 2500 Knoten und Kanten flüssig lief.

Unser Startzustand war eine Machbarkeitsstudie zum Einsatz von heutigen Virtual Reality-System im Zusammenhang mit Softwarevisualisierung zu tätigen und hat sich von da aus hin zu unserer Aufgabenstellung während unserer Bachelorarbeit weiter entwickelt.

Wir haben untersucht, ob es Vorteile in Hinsicht auf Orientierungsaufgaben bringt die von uns entworfenen Software Cities mit einem Head Mounted Display-System gegenüber einem Desktop-System zu erleben. Wir haben uns mit Grundlagen zu vielen Themen auseinander setzen müssen, da unsere Arbeit all diese Bereiche miteinander vereinigt.

Unsere Evaluation war ein interne und wir die sie durch geführt haben, waren auch die Programmierer der Anwendung. Die Probanden waren weder Experten der Orientierung noch der Softwarevisualisierung. Die Probanden waren außerdem mit uns bekannt, was die Aussagekraft unserer Studie weiter einschränkt. Es ist uns nicht gelungen unsere Hypothese[1] aufgrund unserer Datenlage zu bestätigen, aber auch nicht verwerfen. Am Ende zeigte sich, dass wir in unseren Daten zu große Abweichungen zwischen mediierten und gemittelten Werten hatten, um valide Aussagen über eventuelle Vorteile der Head Mounted Display-Umgebung gegenüber einer Desktop-Umgebung zu treffen.

[1] siehe Kapitel 4.6

Dafür konnten wir feststellen, dass jeder Proband entweder eine Orientierungsstrategie mitbrachte oder sehr kurzfristig während des Versuchs eine entwickelte und dann – ob erfolgreich oder nicht – diese bis zum Ende beibehielt. Ob die Place Illusion oder die Immersion einen Effekt auf die Orientierungsfähigkeit haben, können wir nicht deutlich bejahen oder verneinen.

Auf informeller Ebene konnten wir allerdings feststellen, dass die Kantenbündelung die Probanden eher verwirrt als Orientierung verschafft hat. Die Verwunderung über die Führung der Verbindungen wurde von sehr vielen Probanden im Nachgespräch geäußert.

Während der Arbeit an unserem System stellten wir fest, dass wir eine andere Auffassung von Softwarevisualisierung vertreten, als Teile der Literatur. Visualisierung wird dort häufig als im Grunde zeitpunktfixierte, ggfs. noch interaktive Darstellung von Systemdaten gesehen. Zielgruppe dieser Visualisierungen ist häufig das Management oder die Projektleitung. Uns schwebt bei der Visualisierung allerdings eher vor, dies als Werkzeug während der täglichen Arbeit des Systemarchitekten oder sogar des Programmierers zu nutzen. Virtual Environments könnten dabei eine Möglichkeit sein, nicht nur ein System zu sehen, sondern das System im werkstofflichen Sinne zu bearbeiten.

Wir sehen die Aufgabe der Software-Visualisierung allerdings nicht in der reinen Produktion „hübscher“ Sichten auf einen Software – dabei ist auch weniger von Belang, ob die Software „fertig“ ist oder nicht. Ähnlich dem Besucher der Galerie kann der Betrachter einer Software-Visualisierung i.d.R. nicht entscheiden, ob eine Abbildung eine „fertige“ Software oder einen defekten Zwischenstand repräsentiert. In der Kunst kann lediglich der Künstler entscheiden, ob das Kunstwerk fertig ist. Er ist der „Fachmann“ und kann entscheiden, ob er sein Kunstwerk fachgerecht umgesetzt hat. Selbst sein (fiktiver) Auftraggeber oder späterer Käufer kann lediglich anhand seines eigenen Maßstabs (seine Metrik) entscheiden, ob das Kunstwerk seinen Ansprüchen genügt oder nicht.

In den meisten gesichteten Visualisierungen hatten wir den Eindruck, dass der Fokus mehr auf einer gefälligen Präsentation als auf einer praktischen Umsetzung lag. So auch bei den Software-City-Visualisierungen, seien sie nun Head Mounted Display- oder Desktop-Visualisierung: Sie schienen uns mehr dem Zweck zu dienen, hindurch zu navigieren, ähnlich, wie man sich durch eine Galerie oder durch ein Einrichtungshaus bewegt und die Schönheit der Sache an sich bewundert.

Trotz der Untersuchung von Koschke (2003) erhielten wir in einigen informellen Gesprächen mit Programmierern und Qualitätsmanagers aus dem Bereich Software den Eindruck, solche Visualisierungen werden in der Praxis wenig genutzt. Dies könnte an der fehlenden Standardisierung oder an dem für die Zielgruppe nicht-erkennbaren Praxisbezug liegen.

Ein zentraler Aspekt der Softwarevisualisierung ist die Veränderung des Mapping von Metriken zu Mesh-Attributen. Doch wie groß ist der Wiedererkennungseffekt, wenn die Stadt sich verändert? Durch unsere zwei verschiedenen Modellmetriken und die Reaktionen der Probanden auf die Information, die Modelle seien identisch gewesen, schließen wir, dass eine Veränderung des

Mapping schwierig für die Wiedererkennung der Software-City ist. Belegen können wir dies nicht.

Kapitel 8

Ausblick [MOR]

Pollack schreibt 1989 einen enthusiastischen Artikel in der New York Times über Architekten, die davon träumen, Ihre Gebäude virtuell betreten zu können. Heute – nicht einmal 20 Jahre später – ist dies in Ansätzen möglich.

Vielleicht sind wir in der Informatik auf dem Weg in eine solche Zukunft: Vielleicht bekommen wir Informatiker durch Software-Visualiserung – wir wollen sogar sagen durch *visuelle Software-Exploration* um uns ein wenig von der statischen Ansicht klassischer Visualisierung zu abzuheben und und mehr eine Mitmach-Visualisierung zu propagieren – einen neuen Zugang zu und ein neues Verständnis von Software; letztlich vielleicht sogar ein neues Selbstverständnis unseres Berufsstands.

Natürlich haben wir während unserer Arbeit einige Ideen entwickelt, die weit über das hinaus gehen, was wir hier präsentieren – einige im Spaß, einige im Spaß mit ernsthaften Wurzeln. Nachfolgend wollen wir die Ansätze kurz vorstellen die in nahe Zukunft am erfolgversprechendsten oder interessantesten in Erinnerung geblieben sind.

8.1 Software-City-Metapher [MOR]

Während unserer Arbeit an dem System der Software-City drängte sich der Gedanke auf, das eventuell die Stadt-Metapher für eine Software die falsche Metapher sei. Zwar eignet sich die Stadt-Visualisierung grundsätzlich für die Darstellung hierarchischer Daten, doch das gilt für andere Metaphern auch. Bei der Visualisierung von Software ist aber nicht nur die statische Struktur, sondern auch – vielleicht sogar vor allem – die Assoziationen zwischen den Einheiten der Software und vor allem das was aus der textuellen Repräsentation der Software nicht erkennbare, dynamische Verhalten wichtig. Somit deckt die Software-City-Metapher nur einen Teil der Aspekte einer Software ab – auch wenn dies der Teil ist, der mit den meisten anderen

Visualisierungen auch fokussiert wird.[1] Ein Vergleich der Software-City mit anderen Visualisierungsmetaphern könnte interessant sein.

In Bildungsbereich gibt es den Ausdruck, dass man ein *Sprachbad* nimmt. Damit ist gemeint, dass kleine Kinder ein Fremdsprache dadurch lernen, dass sie sie einfach hören und nachahmen, wenn ihre Umgebung diese Fremdsprache spricht. Eventuell können wir in einigen Jahren ein *Software-Bad* nehmen.

8.2 Augmented Reality statt Virtual Reality [MOR]

Interessant wäre es auch, das von uns eingesetzte Virtual Reality-System gegen ein ortserkennendes Augmented Reality-System auszutauschen, so dass die Software in das reale Leben integriert wird. Nicht nur hätten wir dadurch, dass wir dann sichtbare 'Maschinen' konstruieren, einen Brückenschlag in die Ingenieurwissenschaften erreicht, Effekte der VR, die die Arbeit mit diesem System erschweren[2] könnten durch Augmented Reality-System abgemildert werden.

8.3 Visuelles Debugging und dynamische Sichten [MOR]

Wenn es möglich wäre, durch Software-Visualisierung auch ein grafisches Debugging zu ermöglichen, so hätte dies aus unserer Sicht große Vorteile. Zwar gibt es bereits Ansätze eine grafische Visualisierung als Lernsystem bspw. für Zeigerstrukturen einzusetzen, doch ein grafische Debugging auf höherem Niveau hätte wohl bedeutendere Effekte.

8.4 Optimierung der grafischen Darstellung [MOR]

Durch weitere Optimierungen wäre es ggfs. auch möglich, größere System zu visualisieren. Wir haben bspw. aufgrund unserer zum Zwecke der Orientierung sehr rudimentär gehaltenen Visualisierung gerade die grafischen Möglichkeiten bzgl. Materialien und Texturen der Unreal Engine nur gestriffen. Werden solche Elemente in der Software-Visualisierung eingesetzt, wird die Nutzung von sogenannten *Levels of Detail* interessant, mit denen in Abhängigkeit von eine gestuften Objektentfernung eine einfachere grafische Darstellung eingestellt werden kann.

[1] So bietet die Unified Modeling Language eine Vielzahl unterschiedlicher Diagrammformen, die statische Klassenstruktur ist aber die Diagrammart, die wohl am häufigsten eingesetzt wird.
[2] das könnte gerade auch die Immersion und die Präsenz sein

8.5 Orientierungshilfen für komplexe Virtual Environments [MOR]

Bereits die Orientierung in einem lediglich einige hunderte Klassen umfassenden System wie den von uns für die Evaluation eingesetzten ist schwierig. Ideen, die eine Orientierung im Virtual Environment vereinfachen – sozusagen 'virtuelle Navigationssysteme' könnten hier helfen. Alternativ gäbe es Möglichkeiten, Ideen des Gedächtnistrainings zu nutzen und Elemente des Virtual Environments an reale Gegenstände zu knüpfen. Es könnte sein, dass unsere Aufgabenstellung den modernen Zeiten unangemessen ist: Ggfs. müsste man nicht die Fertigkeit der Orientierung prüfen, sondern virtuelle Navigationsgeräte für Software-Cities evaluieren.

8.6 Andere 3D-Engine [MOR]

Auch wenn die Unreal Engine in der Fachpresse gute Kritiken erhält und vergleichsweise schnell an neue Technologien angepasst wird, so kann es sein, dass andere, insbesondere jüngere Grafik-Engines bessere Ergebnisse liefern, da sie bspw. besser an ein Betriebssystem oder Gerätetyp angepasst sind. Hier gilt es, den Markt zu erforschen.

8.7 Anderer Display-Typ [MOR]

Wir haben für unsere Arbeit die HTC Vive eingesetzt, die auf dem Markt momentan noch das beste, aber eines der teuersten Gerät ist. In Zukunft wird es alternative Geräte dazu geben. Ein Markt, der im Auge behalten werden sollte, sind Smartphone-System mit als Virtual Reality- oder Augmented Reality-System. In Verbindung mit einer sehr genauen Lokalisierung könnten Software-Städte in die Landschaft integriert oder dort exploriert werden. Programmieren könnte, durch 'Coding'-Rätsel beigebracht werden, die gelöst werden müssen um Klassen zu 'claimen', ähnlich dem Augmented Reality Spiel Ingress[3].

8.8 Andere Meshes [MOR]

In unserem System haben wir auf den Einsatz von komplexen Mesh-Modellen aus systematischen Gründen verzichtet. Vorstellbar (und mit unserem System auch bereits möglich) ist allerdings,

[3] *https://www.ingress.com/*

statt eines Würfels und einer Röhre auch anderen grafische Modelle zu nutzen. Die momentan eingesetzt Modelle haben wir im Hinblick auf die Systemperformance *getunt*. Ebenso wäre aber auch denkbar, die Modelle durch andere zu ersetzen, die eventuell sogar jedem Elementtyp des Softwaregraphen eine eigene Metapher zuweisen. Ob dies inhaltlich sinnvoll ist, wäre eine interessante Fragestellung vor dem Hintergrund des modernen Ansatzes der *Gamification*, doch sollte geklärt werden, ob dies auch für den Einsatz in Software-Cities von Vorteil wäre.

8.9 Hierarchical Instanced Static Meshes für Splines [MOR]

Durch den Einsatz von Instanced Static Meshes für Splines gäbe es eine Möglichkeit, die Rechenlast für die Generierung der Meshes an die Grafikkarte zu übertragen. Da dabei eine Spline aus anderen Meshes zusammen gesetzt werden müsste, sind hier aufwändigere Berechnung und Repositionierungen notwendig. Außerdem muss bei dieser Lösung damit gerechnet werden, dass – je nach verwendetem Mesh – Lücken in der Spline entstehen. Eine Möglichkeit wäre, die Lücken zu provizieren (bspw. in Form einer Kugelkette als Spline), dies macht das Gesamtbild allerdings sehr unruhig.

8.10 Kantenbündelung oder Kantendeaktivierung [MOR]

Für zukünftige Evaluationen kann es interessant sein, ob die andere Kantenbündelungsalgorithmen oder nur eine Kantendeaktivierung zielführend für eine Orientierung oder eine Verbesserung der Übersichtlichkeit ist. Die untersuchten Kantenbündelungen haben alle das Ziel, möglichst alle Verbindungen zwischen den Elementen zu verdeutlichen. Dies könnte für lebende Software-Visualisierungen wie unsere nachteilig oder gar der komplett falschen Ansatz sein.

8.11 Optionale Anzeige von Splines [MOR]

Gemäß unserer ersten Konzeption halten wird es immer noch für sinnvoll, dass der Benutzer selbst auswählen kann, welche Elemente oder Arten von Elementen er angezeigt haben will. Hierdurch könnte auch eine Konzentration auf die essentiellen Elemente erfolgen und eine Überfrachtung mit Informationen verhindert werden, denn manchmal ist weniger mehr.

8.12 Freiheitsgrade des Head Mounted Display [MOR]

Interessant wäre es zu untersuchen, ob Probanden die Freiheitsgrade in der realen und in der Virtuellen Welt gleichermaßen nutzen, oder ob Hemmungen einen Unterschied im Umgang mit den Realitäten provozieren. In diesem Zusammenhang wäre auch interessant zu eruieren, ob die Hemmungen einem zeitlichen Faktor unterworfen sind.

8.13 Programming Lifecycle [JG]

Ebenfalls von großen Interesse wäre eine Langzeitstudie, bei der unser Ansatz in den Alltag des Programmierens integriert wird um, längerfristige Trainingseffekte untersuchen zu können. Eine Vergleichsstudie zwischen zwei Gruppen mit der selben Reengineeringaufgabe würde sich da anbieten. Wobei eine Zugriff zu unserm Tool hat und dieses auch verwenden muss und die andere dies nicht kann.

Literaturverzeichnis

In: ().

Albertz, Jörg (1997). »Die dritte Dimension - Elemente der räumlichen Wahrnehmung«. In: Schriftenreihe der Freien Akademie. Hrsg. von Jörg Albertz.

Andrews, K. (2002). »Visual exploration of large hierarchies with information pyramids«. In: *Proceedings Sixth International Conference on Information Visualisation*, S. 793–798. DOI: *10.1109/IV.2002.1028871*.

Arnold, Robert S. (1993). *Software Reengineering*. Los Alamitos, CA, USA: IEEE Computer Society Press. ISBN: 0818632712.

Athenstädt, Jan Christoph, Robert Görke, Marcus Krug und Martin Nöllenburg (2012). »Visualizing Large Hierarchically Clustered Graphs with a Landscape Metaphor«. In: Springer, Berlin, Heidelberg, S. 553–554. DOI: *10.1007/978-3-642-36763-2{_}49*. URL: *https://link.springer.com/content/pdf/10.1007%2F978-3-642-36763-2_49.pdf*.

B. Welch, Robert, Theodore Blackmon, Andrew Liu, Barbara Mellers und Lawrence W. Stark (Juni 1996). »The Effects of Pictorial Realism, Delay of Visual Feedback, and Observer Interactivity on the Subjective Sense of Presence«. In: 5, S. 263–273.

Balzer, M. und O. Deussen (2007). »Level-of-detail visualization of clustered graph layouts«. In: *6th International Asia-Pacific Symposium on Visualization, 2007*. Hrsg. von Seok-Hee Hong. Piscataway, NJ: IEEE Service Center, S. 133–140. ISBN: 1-4244-0808-3. DOI: *10.1109/APVIS.2007.329288*.

Barnard, Jack und Art Price (März 1994). »Managing Code Inspection Information«. In: *IEEE Softw.* 11.2, S. 59–69. ISSN: 0740-7459. DOI: *10.1109/52.268958*. URL: *http://dx.doi.org/10.1109/52.268958*.

Bauer, Friedrich L. (1993). »Software Engineering - wie es begann«. In: *Informatik-Spektrum* 16.5, S. 259–260. ISSN: 0170-6012. URL: *http://dblp.uni-trier.de/db/journals/insk/insk16.html#Bauer93a*.

Baumöl, Ulrike, Jens Borchers, Stefan Eicker, Knut Hildebrand, Reinhard Jung und Franz Lehner (1996). »Einordnung und Terminologie des Software Reengineering«. In: *Informatik-Spektrum* 19.4, S. 191–195. DOI: *10.1007/s002870050030*.

Bertin, Jacques (1974). *Graphische Semiologie: Diagramme, Netze, Karten*. Übers. von Georg Jensch, Dieter Schade und Wolfgang Scharfe. Berlin: DE GRUYTER. ISBN: 9783110036602. DOI: *10.1515/9783110834901*. URL: *http://www.degruyter.com/doi/book/10.1515/9783110834901*.

Boehm, Barry W. (Mai 1988). »A Spiral Model of Software Development and Enhancement«. In: *Computer* 21.5, S. 61–72. ISSN: 0018-9162. DOI: *10.1109/2.59*. URL: *http://dx.doi.org/10.1109/2.59*.

Bowman, Doug A., Elizabeth T. Davis, Larry F. Hodges und Albert N. Badre (1999). »Maintaining Spatial Orientation during Travel in an Immersive Virtual Environment«. In: *Presence: Teleoperators and Virtual Environments* 8.6, S. 618–631. DOI: *10.1162/105474699566521*. eprint: *https://doi.org/10.1162/105474699566521*. URL: *https://doi.org/10.1162/105474699566521*.

Braude, Eric J. und Michael E. Bernstein (2011). *Software engineering: modern approaches*. 2. ed. XVI, 782 S. : Ill., graph. Darst. Hoboken, NJ: Wiley. ISBN: 0471692085 and 9780471692089 and 9780471692089.

Brooke, J (Jan. 2013). »SUS: a retrospective«. In: 8, S. 29–40.

Brooke, John (1996). *SUS: A quick and dirty usability scale*.

Buchheim, Christoph, Markus Chimani, Carsten Gutwenger, Michael Jünger und Petra Mutzel (2012). »Crossings and Planarization«. In: *Handbook of Graph Drawing and Visualization*. Hrsg. von R. Tamassia. CRC Press.

Card, Stuart K., Jock D. Mackinlay und Ben Shneiderman, Hrsg. (1999). *Readings in information visualization: Using vision to think*. [Nachdr.] The Morgan Kaufmann series in interactive technologies. San Francisco, Calif.: Morgan Kaufmann. ISBN: 1558605339.

Chance, Sarah S., Florence Gaunet, Andrew C. Beall und Jack M. Loomis (Apr. 1998). »Locomotion Mode Affects the Updating of Objects Encountered During Travel: The Contribution of Vestibular and Proprioceptive Inputs to Path Integration«. In: *Presence: Teleoper. Virtual Environ.* 7.2, S. 168–178. ISSN: 1054-7460. DOI: *10.1162/105474698565659*. URL: *http://dx.doi.org/10.1162/105474698565659*.

Chardonnet, Jean-Rémy, Mohammad Ali Mirzaei und Frédéric Mérienne (2015). »Visually Induced Motion Sickness Estimation and Prediction in Virtual Reality using Frequency Components Analysis of Postural Sway Signal«. In: *ICAT-EGVE 2015 - International Conference on Artificial Reality and Telexistence and Eurographics Symposium on Virtual Environments*. Hrsg. von Masataka Imura, Pablo Figueroa und Betty Mohler. The Eurographics Association. ISBN: 978-3-905674-84-2. DOI: *10.2312/egve.20151304*.

Chikofsky, Elliot J. und James H. Cross [II] (1990). »Reverse Engineering and Design Recovery: A Taxonomy«. In: *IEEE Software* 7(1), S. 13–18.

Creswell, John W (2009). *Research design : qualitative, quantitative, and mixed methods approaches*. English. Third Edition. Previous ed.: 2002. Los Angeles SAGE. ISBN: 9781412965569. URL: *http://swbplus.bsz-bw.de/bsz285083325vlg.htm*.

DeMarco, T. (2009). »Software Engineering: An Idea Whose Time Has Come and Gone?« In: *IEEE Software* 26.4, S. 96–96. ISSN: 0740-7459. DOI: *10.1109/MS.2009.101*.

DeMarco, Tom (1982). *Controlling software projects: Management, measurement and estimation*. New York: Yourdon. ISBN: 0-917072-32-4.

DeMarco, Tom, Doris Märtin und Christian Märtin (1997). *Warum ist Software so teuer?: und andere Rätsel des Informationszeitalters*. VIII, 178 S. ; 23 cm : graph. Darst. München [u.a.]: Hanser. ISBN: 3446189025 and 9783446189027. URL: *http://katalog.suub.uni-bremen.de/DB=1/LNG=DU/CMD?ACT=SRCHA&IKT=8000&TRM=24083759**.

Deore, Akshata D. und R. L. Paikrao (2013). »Survey on Graph-Hierarchy Visualization Techniques«. In: *International Journal of Emerging Technology and Advanced Engineering* 3.3, S. 477–481.

Diehl, Stephan (2007). *Software Visualization: Visualizing the Structure, Behaviour, and Evolution of Software*. Berlin, Heidelberg: Springer-Verlag Berlin Heidelberg. ISBN: 9783540465041. DOI: *10.1007/978-3-540-46505-8*. URL: *http://site.ebrary.com/lib/alltitles/docDetail.action?docID=10230347*.

Domingue, John, Blaine Price und Marc Eisenstadt (1992). »A framework for describing and implementing software visualization systems«. In: *Proceedings of Graphics Interface '92*. GI 1992. Vancouver, British Columbia, Canada: Canadian Information Processing Society, S. 53–60. ISBN: 0-9695338-1-0. DOI: *10.20380/GI1992.07*.

Domingue, John, Blaine A. Price und Marc Eisenstadt (1994). »Viz: A Framework for Describing and Implementing Software Visualization Systems«. In: *User-Centred Requirements for Software Engineering Environments*. Hrsg. von David J. Gilmore, Russel L. Winder und Françoise Détienne. Berlin, Heidelberg: Springer Berlin Heidelberg, S. 197–212. ISBN: 978-3-662-03035-6. DOI: *10.1007/978-3-662-03035-6_16*. URL: *https://doi.org/10.1007/978-3-662-03035-6_16*.

Dörner, Ralf, Wolfgang Broll, Paul Grimm und Bernhard Jung, Hrsg. (2013). *Virtual und Augmented Reality (VR/AR): Grundlagen und Methoden der Virtuellen und Augmentierten Realität*. eXamen.press. Berlin: Springer Vieweg. ISBN: 9783642289026. DOI: *10.1007/978-3-642-28903-3*. URL: *http://dx.doi.org/10.1007/978-3-642-28903-3*.

Drucker, Peter F. (1963). »Managing for business effectiveness«. In: *Harvard business review* 41.3, S. 53–60. URL: *https://hbr.org/1963/05/managing-for-business-effectiveness*.

– (1967). *The effective executive / Peter F. Drucker*. English. Heinemann London, viii, 148 p. ;

Dwyer, Tim (2001). »Three Dimensional UML Using Force Directed Layout«. In: *Proceedings of the 2001 Asia-Pacific Symposium on Information Visualisation - Volume 9*. APVis '01. Sydney, Australia: Australian Computer Society, Inc., S. 77–85. ISBN: 0-909925-87-9. URL: *http://dl.acm.org/citation.cfm?id=564040.564050*.

Dwyer, Tim und Peter Eckersley (2002). »WilmaScope — An Interactive 3D Graph Visualisation System«. In: *Graph Drawing: 9th International Symposium, GD 2001 Vienna, Austria, September 23–26, 2001 Revised Papers*. Hrsg. von Petra Mutzel, Michael Jünger und Sebastian

Leipert. Berlin, Heidelberg: Springer Berlin Heidelberg, S. 442–443. ISBN: 978-3-540-45848-7. DOI: *10.1007/3-540-45848-4_37*. URL: *https://doi.org/10.1007/3-540-45848-4_37*.

Eicker, Stefan (2011). *Software-Reengineering – Enzyklopaedie der Wirtschaftsinformatik*. URL: *http://www.enzyklopaedie-der-wirtschaftsinformatik.de/wi-enzyklopaedie/lexikon/is-management/Integration-und-Migration-von-IT-Systemen/Software-Reengineering*.

Eisenstadt, Marc, Blaine A. Price und John Domingue (1992). »Software visualization as a pedagogical tool«. In: *Instructional Science* 21.5, S. 335–364. ISSN: 1573-1952. DOI: *10.1007/BF00121202*. URL: *https://doi.org/10.1007/BF00121202*.

Ellis, Stephen R. (1994). »What Are Virtual Environments?« In: *IEEE Computer Graphic and Applications* 14.1, S. 17–22.

Fenton, Norman E. und Shari Lawrence Pfleeger (1998). *Software Metrics: A Rigorous and Practical Approach*. 2nd. Boston, MA, USA: PWS Publishing Co. ISBN: 0534954251.

Fettke, Peter (2005). »Unified Modeling Language«. In: *Encyclopedia of Information Science and Technology, Volume I-V*. Hrsg. von Mehdi Khosrow-Pour. Hershey, PA, USA: IGI Global, S. 2921–2928. ISBN: 159140553X.

Fowler, M. (2003). *Patterns of Enterprise Application Architecture*. A Martin Fowler signature book. Addison-Wesley. ISBN: 9780321127426.

Fowler, M. und K. Beck (1999). *Refactoring: Improving the Design of Existing Code*. Addison-Wesley object technology series. Addison-Wesley. ISBN: 9780201485677.

Gabler Wirtschaftslexikon (2013). *Die ganze Welt der Wirtschaft: Betriebswirtschaft, Volkswirtschaft, Wirtschaftsrecht, Recht und Steuern*. 18., aktualisierte Auflage. Wiesbaden: Gabler Verlag.

GamesIndustry.biz, Hrsg. (2005). *GDC: Epic's Rein slams publishers over next-gen costs "scare"*. URL: *http://www.gamesindustry.biz/articles/gdc-epics-rein-slams-publishers-over-next-gen-costs-scare* (abgerufen am 21. 07. 2017).

Gansner, Emden R., Yifan Hu, Stephen North und Carlos Scheidegger (2011). »Multilevel agglomerative edge bundling for visualizing large graphs«. In: *2011 IEEE Pacific Visualization Symposium*. IEEE, S. 187–194. ISBN: 978-1-61284-935-5. DOI: *10.1109/PACIFICVIS.2011.5742389*.

Grechenig, Thomas (2010). *Softwaretechnik: mit Fallbeispielen aus realen Entwicklungsprojekten*. it Informatik. 687 S. : Ill., zahlr. graph. Darst. München [u.a.]: Pearson Studium. ISBN: 9783868940077 and 3868940073.

Green, T.R.G. und M. Petre (1996). »Usability Analysis of Visual Programming Environments: A 'Cognitive Dimensions' Framework«. In: *Journal of Visual Languages & Computing* 7.2, S. 131 –174. ISSN: 1045-926X. DOI: *https://doi.org/10.1006/jvlc.1996.0009*. URL: *http://www.sciencedirect.com/science/article/pii/S1045926X96900099*.

Guan, Yepeng und Mingen Zheng (2008). »Real-time 3D pointing gesture recognition for natural HCI«. In: *2008 7th World Congress on Intelligent Control and Automation*, S. 2433–2436. DOI: *10.1109/WCICA.2008.4593304*.

Guo, Lin, Wanli Zuo, Tao Peng und Binod Kumar Adhikari (2015). »Attribute-based edge bundling for visualizing social networks«. In: *Physica A: Statistical Mechanics and its Applications* 438, S. 48–55. ISSN: 0378-4371. DOI: *10.1016/j.physa.2015.06.015*.

Hasler, Béatrice S., Bernhard Spanlang und Mel Slater (Apr. 2017). »Virtual race transformation reverses racial in-group bias«. In: *PLOS ONE* 12.4, S. 1–20. DOI: *10.1371/journal.pone.0174965*. URL: *https://doi.org/10.1371/journal.pone.0174965*.

Hering, Ekbert (1984). »Der Datenflußplan nach DIN 66001«. In: *Software-Engineering: Mit 77 Bildern und 22 Übungsaufgaben*. Wiesbaden: Vieweg+Teubner Verlag, S. 56–62. ISBN: 978-3-322-86222-8. DOI: *10.1007/978-3-322-86222-8_7*. URL: *https://doi.org/10.1007/978-3-322-86222-8_7*.

Hettinger, Lawrence J. und Gary E. Riccio (1992). »Visually Induced Motion Sickness in Virtual Environments«. In: *Presence: Teleoperators and Virtual Environments* 1.3, S. 306–310. DOI: *10.1162/pres.1992.1.3.306*. eprint: *https://doi.org/10.1162/pres.1992.1.3.306*. URL: *https://doi.org/10.1162/pres.1992.1.3.306*.

Holten, Danny (2006). »Hierarchical Edge Bundles: Visualization of Adjacency Relations in Hierarchical Data«. In: *IEEE transactions on visualization and computer graphics* 12.5. ISSN: 1077-2626.

HTC. *Über die Vive Controller*. URL: *https://www.vive.com/de/support/category_howto/about-the-controllers.html* (abgerufen am 15. 09. 2017).

Hunicke, Robin, Marc Leblanc und Robert Zubek (2004). »MDA: A Formal Approach to Game Design and Game Research«. In: 1.

»IEEE Standard for a Software Quality Metrics Methodology« (1998). In: *IEEE Std 1061-1998*, S. i–. DOI: *10.1109/IEEESTD.1998.243394*.

»IEEE Standard Glossary of Software Engineering Terminology« (1990). In: *IEEE Std 610.12-1990*, S. 1–84. DOI: *10.1109/IEEESTD.1990.101064*.

J. Rieser, John (Dez. 1989). »Access to Knowledge of Spatial Structure at Novel Points of Observation«. In: 15, S. 1157–65.

Kan, Stephen H. (2002). *Metrics and Models in Software Quality Engineering*. 2nd. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc. ISBN: 0201729156.

Kathrin Hollmer (2013). *Wie verbessere ich meinen Orientierungssinn?* URL: *http://www.jetzt.de/lexikon/wie-verbessere-ich-meinen-orientierungssinn-564060*.

Kennedy, Robert S., Julie Drexler und Robert C. Kennedy (2010). »Research in visually induced motion sickness«. In: *Applied Ergonomics* 41.4. Special Section - The First International Symposium on Visually Induced Motion Sickness, Fatigue, and Photosensitive Epileptic Seizures (VIMS2007), S. 494 –503. ISSN: 0003-6870. DOI: *https://doi.org/10.1016/j.apergo.2009.11.006*. URL: *http://www.sciencedirect.com/science/article/pii/S0003687009001574*.

Koschke, Rainer (2003). »Software visualization in software maintenance, reverse engineering, and re-engineering: a research survey«. In: *JOURNAL OF SOFTWARE MAINTENANCE AND EVOLUTION: RESEARCH AND PRACTICE* 15, S. 87–109. DOI: *10.1002/smr.270*.

Koschke, Rainer (2013). *Software-Reengineering – Vorlesung an der Universität Bremen*. URL: *http://www.informatik.uni-bremen.de/st/lehre/re13/allefolien.pdf* (abgerufen am 03. 08. 2017).

Kruchten, Philippe (2001). »Common Misconceptions about Software Architecture«. In: URL: *https://pdfs.semanticscholar.org/6a83/08cfb3b5f3c650c9d9c218b255d0fc364558.pdf* (abgerufen am 18. 07. 2017).

Lichter, Jochen Ludewig; Horst (2007). *Software Engineering: Grundlagen, Menschen, Prozesse, Techniken*. 1. Aufl. XXI, 618 S. ; 240 mm x 165 mm : graph. Darst. Heidelberg: dpunkt. ISBN: 3898642682 and 9783898642682 and 9783898642682.

Maier, Peter H. (1999). »Räumliches Vorstellungsvermögen: ein theoretischer Abriß des Phänomens räumliches Vorstellungsvermögen ; mit didaktischen Hinweisen für den Unterricht«. 360 S. ; 21 cm : Ill., graph. Darst. Diss. Donauwörth. ISBN: 3403030903 and 9783403030904 and 9783403030904. URL: *http://katalog.suub.uni-bremen.de/DB=1/LNG=DU/CMD?ACT=SRCHA&IKT=8000&TRM=34785013**.

McCauley, Michael E. und Thomas J. Sharkey (8. Dez. 2005). »Cybersickness: Perception of Self-Motion in Virtual Environment.« In: *Presence* 1.3, S. 311–318. URL: *http://dblp.uni-trier.de/db/journals/presence/presence1.html#McCauleyS92*.

McClure, Carma (1992). *The Three Rs of Software Automation: Re-engineering, Repository, Reusability*. Upper Saddle River, NJ, USA: Prentice-Hall, Inc. ISBN: 0-13-915240-7.

McConathy, Deirdre A. (Feb. 1993). »Evaluation Methods in Visualization: Combating the Emperor's New Clothes Phenomenon«. In: *SIGBIO Newsl.* 13.1, S. 2–8. ISSN: 0163-5697. DOI: *10.1145/163424.163425*. URL: *http://doi.acm.org/10.1145/163424.163425*.

McCormick, B. H., T. A. DeFanti und M. D. Brown (1987). *Visualization in Scientific Computing: Computer Graphics 21(6), November 1987*. (Abgerufen am 22. 07. 2017).

Milgram, Paul, Haruo Takemura, Akira Utsumi und Fumio Kishino (1995). »Augmented reality: a class of displays on the reality-virtuality continuum«. In: *Proceedings of SPIE 2351, Telemanipulator and Telepresence Technologies*, S. 282–292. DOI: *10.1117/12.197321*.

Mine, Mark R. (1995). *Virtual Environment Interaction Techniques*. Techn. Ber. Chapel Hill, NC, USA.

Müller, Bernd (1997). *Reengineering: eine Einführung*. Leitfäden der Informatik. X, 197 S. ; 23 cm : I11., graph. Darst. Stuttgart: Teubner. ISBN: 3519029421.

Myers, B. A. (1986). »Visual Programming, Programming by Example, and Program Visualization: A Taxonomy«. In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. CHI '86. Boston, Massachusetts, USA: ACM, S. 59–66. ISBN: 0-89791-180-6. DOI: *10.1145/22627.22349*. URL: *http://doi.acm.org/10.1145/22627.22349*.

Myers, Brad A., Rishi Bhatnagar, Jeffrey Nichols, Choon Hong Peck, Dave Kong, Robert Miller und A. Chris Long (2002). »Interacting at a Distance: Measuring the Performance of Laser Pointers and Other Devices«. In: *Proceedings of the SIGCHI Conference on Human Factors in*

Computing Systems. CHI '02. Minneapolis, Minnesota, USA: ACM, S. 33–40. ISBN: 1-58113-453-3. DOI: *10.1145/503376.503383*. URL: *http://doi.acm.org/10.1145/503376.503383*.

Naur, Peter und Brian Randell, Hrsg. (1969). *Software Engineering: Report on a conference sponsored by the NATO Science Committee*. URL: *http://homepages.cs.ncl.ac.uk/brian.randell/NATO/nato1968.PDF* (abgerufen am 17. 07. 2017).

Neumann, Petra, Stefan Schlechtweg und Sheelagh Carpendale (2005). »ArcTrees: Visualizing Relations in Hierarchical Data«. In: *EUROGRAPHICS - IEEE VGTC Symposium on Visualization (2005)*. Hrsg. von K. W. Brodlie, D. J. Duke und K. I. Joy. URL: *https://pdfs.semanticscholar.org/356a/9e93297a5827cf05fa2f86ae3e19dbdf2bbb.pdf*.

Oh, Ji-Young und Wolfgang Stuerzlinger (2002). *Laser Pointers as Collaborative Pointing Devices*.

Orchard, David (2004). *Achieving Loose Coupling*. URL: *https://web.archive.org/web/20080309074842/http://dev2dev.bea.com/pub/a/2004/02/orchard.html* (abgerufen am 18. 07. 2017).

Panas, T., R. Berrigan und J. Grundy (2003). »A 3D metaphor for software production visualization«. In: *Proceedings on Seventh International Conference on Information Visualization, 2003. IV 2003.* S. 314–319. DOI: *10.1109/IV.2003.1217996*.

Panas, T., T. Epperly, D. Quinlan, A. Saebjornsen und R. Vuduc (2007). »Communicating Software Architecture using a Unified Single-View Visualization«. In: *12th IEEE International Conference on Engineering Complex Computer Systems (ICECCS 2007)*, S. 217–228. DOI: *10.1109/ICECCS.2007.20*.

Passig, Kathrin und Johannes Jander (2013). *Weniger schlecht programmieren*. 1. Aufl. Köln u.a.: O'Reilly. ISBN: 3-89721-567-5.

Perry, Dewayne E., Adam A. Porter und Lawrence G. Votta (2000). »Empirical Studies of Software Engineering: A Roadmap«. In: *Proceedings of the Conference on The Future of Software Engineering*. ICSE '00. Limerick, Ireland: ACM, S. 345–355. ISBN: 1-58113-253-0. DOI: *10.1145/336512.336586*. URL: *http://doi.acm.org/10.1145/336512.336586*.

Playfair, William (1801). »The commercial and political atlas: representing, by means of stained copper-plate charts, the progress of the commerce, revenues, expenditure, and debts of England, during the whole of the eighteenth century: London: J. Wallis [etc.]; first publish 1786; third Edition«. In: *Annenberg Rare Book and Manuscript Library*. Bd. HF3501 .P5 1801. URL: *http://sceti.library.upenn.edu/sceti/printedbooksNew/index.cfm?TextID=hf3501_p5_1801&PagePosition=2*.

Pollack, Andrew (1989). *For Artificial Reality, Wear A Computer*. Hrsg. von New York Times. URL: *http://www.nytimes.com/1989/04/10/business/for-artificial-reality-wear-a-computer.html* (abgerufen am 21. 07. 2017).

Regian, J. Wesley, Wayne L. Shebilske und John M. Monk (1992). »Virtual Reality: An Instructional Medium for Visual-Spatial Tasks«. In: *Journal of Communication* 42.4, S. 136–149. ISSN:

1460-2466. DOI: *10.1111/j.1460-2466.1992.tb00815.x*. URL: *http://dx.doi.org/10.1111/j.1460-2466.1992.tb00815.x*.

Reichelt, Stephan, Ralf Häussler, Gerald Fütterer und Norbert Leister (2010). *Depth cues in human visual perception and their realization in 3D displays*. DOI: *10.1117/12.850094*. URL: *http://dx.doi.org/10.1117/12.850094*.

Rekhter, Yakov und Tony Li (1983). *DIN 66001:1983-12 Informationsverarbeitung; Sinnbilder und ihre Anwendung*. DIN 66001. Deutsches Institut für Normung. URL: *http://www.rfc-editor.org/rfc/rfc1654.txt*.

Renaud, Patrice, Dominique Trottier, Joanne-Lucine Rouleau, Mathieu Goyette, Chantal Saumur, Tarik Boukhalfi und Stéphane Bouchard (2014). »Using immersive virtual reality and anatomically correct computer-generated characters in the forensic assessment of deviant sexual preferences«. In: *Virtual Reality* 18.1, S. 37–47. ISSN: 1434-9957. DOI: *10.1007/s10055-013-0235-8*. URL: *https://doi.org/10.1007/s10055-013-0235-8*.

Rheingolf, H. (1992). *Virtuelle Welten - Reisen im Cyberspace*. Berlin: Rowohlt.

Riecke, Bernhard E., Douglas W. Cunningham und Heinrich H. Bülthoff (2007). »Spatial updating in virtual reality: the sufficiency of visual information«. In: *Psychological Research* 71.3, S. 298–313. ISSN: 1430-2772. DOI: *10.1007/s00426-006-0085-z*. URL: *https://doi.org/10.1007/s00426-006-0085-z*.

Rigamonti, Anna (2015). *1 + 1 = 2: Informationsvisualisierung - Missbrauch und Möglichkeit ; Grundlagen des Informationsdesigns*. 168 Seiten ; 26 cm : Illustrationen. Stuttgart: avedition. ISBN: 9783899862287 and 3899862287.

Rinderle, Stefan (2015–6). »Softwarevisualisierung mit SonarQube in 3-D«. In: *Eclipse Magazin*.

Rohr, Oliver (2012). *Softwarevisualisierung in mehreren Dimensionen: Visualisierungsmethoden für objektorientierte Software*. neue Ausg. Saarbrücken: AV Akademikerverlag. ISBN: 3639410114.

Sansen, Joris, Frederic Lalanne, David Auber und Romain Bourqui (2015). »Adjasankey: Visualization of Huge Hierarchical Weighted and Directed Graphs«. In: *2015 19th International Conference on Information Visualisation*. IEEE, S. 211–216. ISBN: 978-1-4673-7568-9. DOI: *10.1109/iV.2015.46*.

Santos, C. Russo Dos, P. Gros, P. Abel, D. Loisel, N. Trichaud und J. P. Paris (2000). »Mapping information onto 3D virtual worlds«. In: *2000 IEEE Conference on Information Visualization. An International Conference on Computer Visualization and Graphics*, S. 379–386. DOI: *10.1109/IV.2000.859785*.

Santos, Selan und Ken Brodlie (Juni 2004). »Gaining understanding of multivariate and multidimensional data through visualization«. In: 28, S. 311–325.

Schmidt, D. (1993). »Was heißt hier eigentlich ingenieurmäßig?« In: *Informatik-Spektrum* 16.5, S. 300–301.

Schreiber, Kai (2005). *Riesenmaschine - Der Walzahn des Zahnwals: Das brandneue Universum*. URL: *http://riesenmaschine.de/index.html?nr=20051215114232* (abgerufen am 16. 09. 2017).

Shneiderman, B. (1993). *Sparks of Innovation in Human-computer Interaction*. Human/computer interaction. Ablex Publishing Company. ISBN: 9781567500783. URL: *https://books.google.de/books?id=G0AdPjbIoVUC*.

Shneiderman, Ben (Jan. 1992). »Tree Visualization with Tree-maps: 2-d Space-filling Approach«. In: *ACM Trans. Graph.* 11.1, S. 92–99. ISSN: 0730-0301. DOI: *10.1145/102377.115768*. URL: *http://doi.acm.org/10.1145/102377.115768*.

Sieber, Rene (1996). »Visuelle Wahrnehmung dreidimensionaler parametrisierter Objekte und Objektgruppen«. In: *Geoprocessing Reihe Universität Zürich* 26.

Slater, Mel (2009). »Place illusion and plausibility can lead to realistic behaviour in immersive virtual environments«. In: *Philosophical Transactions of the Royal Society of London B: Biological Sciences* 364.1535, S. 3549–3557. ISSN: 0962-8436. DOI: *10.1098/rstb.2009.0138*. eprint: *http://rstb.royalsocietypublishing.org/content/364/1535/3549.full.pdf*. URL: *http://rstb.royalsocietypublishing.org/content/364/1535/3549*.

Slater, Mel und Sylvia Wilbur (1997). »A Framework for Immersive Virtual Environments (FIVE): Speculations on the Role of Presence in Virtual Environments«. In: *Presence: Teleoperators and Virtual Environments* 6.6, S. 603–616. DOI: *10.1162/pres.1997.6.6.603*. eprint: *https://doi.org/10.1162/pres.1997.6.6.603*. URL: *https://doi.org/10.1162/pres.1997.6.6.603*.

Sneed, Harry, Martin Hasitschka und Maria-Therese Teichmann (Jan. 2005). *Software-Produktmanagement - Wartung und Weiterentwicklung bestehender Anwendungssysteme*. ISBN: 978-3-89864-274-3.

Starke, Gernot (2014). *Effektive Software-Architekturen. Ein praktischer Leitfaden*. München: Carl Hanser. ISBN: 9783446436145.

Steinbrückner, Frank (Juni 2013). »Consistent Software Cities : supporting comprehension of evolving software systems«. Diss. Cottbus: Brandenburgischen Technischen Universität Cottbus. URL: *https://opus4.kobv.de/opus4-btu/frontdoor/index/index/docId/1681*.

Steuer, Jonathan (1992). »Defining Virtual Reality: Dimensions Determining Telepresence«. In: *Journal of Communication* 42.4, S. 73–93. ISSN: 1460-2466. DOI: *10.1111/j.1460-2466.1992.tb00812.x*. URL: *http://dx.doi.org/10.1111/j.1460-2466.1992.tb00812.x*.

Stockmann, Reinhard (2004). *Was ist eine gute Evaluation? Einführung zu Funktionen und Methoden von Evaluationsverfahren*. Bd. 9, S. 18. URL: *http://nbn-resolving.de/urn:nbn:de:0168-ssoar-118018*.

Stone, R.J., R.A. Earnshaw, M.A. Gigante und Jones H. (1993). *Virtual Reality Systems*. London: Academic Press.

Sutherland, I. E. (1965). *The ultimate display*.

The American Heritage Dictionary of the English Language (2017). 5th ed., Dell mass market ed. New York: Dell. ISBN: 0553583220.

Tukey, John Wilder (1977). *Exploratory data analysis*. Addison-Wesley series in behavioral science. Reading, Mass.: Addison-Wesley. ISBN: 0201076160.

Vion-Dury, Jean-Yves, Miguel Santana und Susan Bull (1994). »Virtual Images: Interactive Visualization of Distributed Object-oriented Systems«. In: *Proceedings of the Ninth Annual Con-*

ference on Object-oriented Programming Systems, Language, and Applications. OOPSLA '94. Portland, Oregon, USA: ACM, S. 65–84. ISBN: 0-89791-688-3. DOI: *10.1145/191080.191095*. URL: *http://doi.acm.org/10.1145/191080.191095*.

Vlahos, Petro (1965). »Three-Dimensional Display: Ist Cues and Techniques«. In: *Information Display* 2, S. 10–20.

Wang, Ranxiao Frances (2004). »Between reality and imagination: When is spatial updating automatic?« In: *Perception & Psychophysics* 66.1, S. 68–76. ISSN: 1532-5962. DOI: *10.3758/BF03194862*. URL: *https://doi.org/10.3758/BF03194862*.

Ware, C. und G. Franck (1994). »Viewing a graph in a virtual reality display is three times as good as a 2D diagram«. In: *Proceedings of 1994 IEEE Symposium on Visual Languages*, S. 182–183. DOI: *10.1109/VL.1994.363621*.

Ware, Colin und Glenn Franck (März 1996). »Evaluating stereo and motion cues for visualizing information nets in three dimensions«. In: 15.

Ware, Colin, David Hui und Glenn Franck (1993). »Visualizing Object Oriented Software in Three Dimensions«. In: *In CASCON'93 Proceedings*, S. 612–620.

Wertheimer, Max (1923). »Untersuchungen zur Lehre von der Gestalt. II.« In: *Pschologische Forschung*. URL: *http://gestalttheory.net/download/Wertheimer1923_Lehre_von_der_Gestalt.pdf* (abgerufen am 25. 07. 2017).

Wettel, R. und M. Lanza (2008a). »Visual Exploration of Large-Scale System Evolution«. In: *2008 15th Working Conference on Reverse Engineering*, S. 219–228. DOI: *10.1109/WCRE.2008.55*.

Wettel, Richard und Michele Lanza (2008b). »Visually Localizing Design Problems with Disharmony Maps«. In: *Proceedings of the 4th ACM Symposium on Software Visualization*. SoftVis '08. Ammersee, Germany: ACM, S. 155–164. ISBN: 978-1-60558-112-5. DOI: *10.1145/1409720.1409745*. URL: *http://doi.acm.org/10.1145/1409720.1409745*.

Wong, Pak Chung und R. Daniel Bergeron (1997). »30 Years of Multidimensional Multivariate Visualization«. In: *Scientific Visualization*. Hrsg. von Gregory M. Nielson, Hans Hagen und H. Muller. Los Alamitos, CA: IEEE Comuter Society Press, S. 3–33. URL: *http://citeseerx.ist.psu.edu/viewdoc/download;jsessionid=821852415E8DDA3142E353175E2E1002?doi=10.1.1.30.4639&rep=rep1&type=pdf* (abgerufen am 22. 07. 2017).

Yau, Nathan und Martina Hesse-Hujber (2014). *Einstieg in die Visualisierung: Wie man aus Daten Informationen macht*. 1. Aufl. Weinheim: Sybex Wiley-VCH. ISBN: 9783527760503.

Zhou, Chaofeng (2008). »A Survey of Edge Bundling Methods for Graph Visualization«. In: *Introduction to Securitization*. Hrsg. von Frank J. Fabozzi und Vinod Kothari. Hoboken, NJ, USA: John Wiley & Sons, Inc, S. 1–12. ISBN: 9781118266892. DOI: *10.1002/9781118266892.ch1*.

Zhou, Hong, Xiaoru Yuan, Weiwei Cui, Huamin Qu und Baoquan Chen (2008). »Energy-Based Hierarchical Edge Clustering of Graphs«. In: *2008 IEEE Pacific Visualization Symposium*. IEEE, S. 55–61. ISBN: 978-1-4244-1966-1. DOI: *10.1109/PACIFICVIS.2008.4475459*.

Zielasko, D., B. Weyers, B. Hentschel und T. W. Kuhlen (2016). »Interactive 3D Force-Directed Edge Bundling«. In: *Computer Graphics Forum* 35.3, S. 51–60. ISSN: 01677055. DOI: *10.1111/cgf.12881*.

Zimmermann, Peter (2008). »Virtual Reality Aided Design. A survey of the use of VR in automotive industry«. In: *Product Engineering: Tools and Methods Based on Virtual Reality*. Hrsg. von Doru Talaba und Angelos Amditis. Dordrecht: Springer Netherlands, S. 277–296. ISBN: 978-1-4020-8200-9. DOI: *10.1007/978-1-4020-8200-9_13*. URL: *https://doi.org/10.1007/978-1-4020-8200-9_13*.

Zwecker, Loel (2010). *Was bisher geschah: Eine kleine Weltgeschichte*. 1. Aufl. München: Pantheon. ISBN: 9783570551271.

Anhang A

Appendix

A.1 Evaluation

Versuchsleiterskript

Wir haben eine abstrakte Stadt modelliert und wollen testen, wie gut Sie sich in dieser zurechtfinden. Die Stadt besteht aus hausähnlichen und straßenähnlichen Elementen sowie rohrähnlichen Verbindungen zwischen diesen Elementen.

Wir werden Sie mit ein paar Aufgaben durch die abstrakte Stadt leiten. Ihre Hauptaufgabe soll sein, am Ende wieder zurück zum Startelement zu finden. Stellen Sie es sich vor – wie im demographischen Fragebogen bereits angedeutet – wie ihr Urlaubshotel, zu dem Sie Abends nach einem Rundgang durch den Urlaubsort zurück finden müssen.

Einführung – Übungsmodell

Steuerung

Zunächst erklären wir Ihnen die Steuerung für diesen Versuch.

Sie können fliegen und einzelne Objekte auswählen. Ausgewählte Objekte sind grün-schimmernd umrahmt und zeigen einen Info-Kasten. Der Auswahlrahmen fächert in der Entfernung auf. Sie können stets höchstens 1 Objekt auswählen. Im Info-Kasten stehen Informationen, die während des Versuchs wichtig sind, u. a. der Name des ausgewählten Objekts.Die Auswahl hat eine Begrenzung in der Reichweite. Können Sie ein Objekt nicht auswählen, müssen Sie näher ran fliegen.

Maus und Tastatur

Mit den Tasten WASD können Sie vorwärts, rückwärts und seitwärts fliegen. Die

Vorwärts-Flugrichtung bewegt Sie in die Mitte des Bildschirms. Mit den Tasten Q und E können sie hoch und runter fliegen.

Zum Umsehen halten Sie die **rechte** Maustaste gedrückt und ziehen Sie die Maus über den Bildschirm; ggfs. müssen Sie mehrfach ziehen um sich weiter umzusehen.

Mit der **linken** Maustaste können Sie Dinge auswählen und den Info-Kasten anzeigen.

HMD und Controller

Sie können sie normal bewegen, das System erkennt Ihre Bewegungen und führt das Bild nach. Sie bekommen zwei sogenannte *Controller*.

Der linke Controller dient zum Navigieren. Auf diesem können Sie das Pad oben mit dem Daumen benutzen um zu navigieren. Halten Sie dieses Pad **oben** gedrückt, fliegen Sie in Richtung des Controllergriffs vorwärts und wenn Sie das Pad **unten** gedrückt halten, fliegen Sie entgegen der Richtung des Controllergriffs. Durch Bewegen des Controllers können Sie Ihre Flugrichtung beeinflussen.

Der rechte Controller dient zum Auswählen. Auf der Unterseite befindet sich ein Schalter, den Sie mit Ihrem Zeigefinger bedienen können. Dieser aktiviert einen Laser, mit dem sie ein Objekt auswählen können.

{Proband kann Fliegen und Auswählen üben.}

Navigationsaufgabe

Wir wollen jetzt testweise mit Ihnen ein Szenario durchspielen, wie wir es anschließend in der eigentlichen Evaluation mit Ihnen testen wollen. Wir bitten Sie, wieder einige Elemente in der angesagten Reihenfolge abzufliegen und dann zum Startelement zurück zu finden.

1. Navigieren Sie vom Startelement **N1** zum anderen Ende der Verbindung **E2** vom Typ **Static Call**. Dort befindet sich das Element **N4**.
2. Navigieren Sie vom Element **N4** zum anderen Ende der Verbindung **E10** vom Typ **Static Call**. Dort befindet sich das Element **N5**
3. Navigieren Sie vom Element **N5** zum anderen Ende der Verbindung **E11** vom Typ **Static Call**. Dort befindet sich das Element **N8**
4. Navigieren Sie von diesem Element auf dem schnellsten Wege zurück zum Startelement

Evaluation – Modell_1-1 (Cli)

Navigationsaufgabe

Wir bitten Sie, wieder einige Elemente in der angesagten Reihenfolge abzufliegen und dann zum Startelement zurück zu finden.

1. Es wird darum gehen einige Elemente ab zu fliegen und dann zum Startelement zurückzufinden.
2. Navigieren Sie vom Startelement **N143** zum anderen Ende der Verbindung **E1288** vom Typ **Static Call**. Dort befindet sich das Element **N12**
3. Navigieren Sie vom Element **N12** zum anderen Ende der Verbindung **E1136** vom Typ **Static Call**. Dort befindet sich das Element **N124**
4. Navigieren Sie vom Element **N124** zum anderen Ende der Verbindung **E1134** vom Typ **Dispatching Call**. Dort befindet sich das Element **N556**
5. Navigieren Sie von diesem Element auf dem schnellsten Wege zurück zum Startelement

Evaluation – Modell_1-2 (Cli)

Navigationsaufgabe

Wir bitten Sie, wieder einige Elemente in der angesagten Reihenfolge abzufliegen und dann zum Startelement zurück zu finden.

1. Es wird darum gehen einige Elemente ab zu fliegen und dann zum Startelement zurückzufinden.
2. Navigieren Sie vom Startelement **N309** zum anderen Ende der Verbindung **E1039** vom Typ **Dispatching Call**. Dort befindet sich das Element **N119**
3. Navigieren Sie vom Element **N119** zum anderen Ende der Verbindung **E1045** vom Typ **Static Call**. Dort befindet sich das Element **N74**
4. Navigieren Sie vom Element **N74** zum anderen Ende der Verbindung **E815** vom Typ **Static Call**. Dort befindet sich das Element **N510**
5. Navigieren Sie von diesem Element auf dem schnellsten Wege zurück zum Startelement

Evaluation – Modell_2-1 (Chain)

Navigationsaufgabe

Wir bitten Sie, wieder einige Elemente in der angesagten Reihenfolge abzufliegen und dann zum Startelement zurück zu finden.

1. Es wird darum gehen einige Elemente ab zu fliegen und dann zum Startelement zurückzufinden.
2. Navigieren Sie vom Startelement **N804** zum anderen Ende der Verbindung **E2017** vom Typ **Override**. Dort befindet sich das Element **N223**
3. Navigieren Sie vom Element **N223** zum anderen Ende der Verbindung **E2021** vom Typ **Static Call**. Dort befindet sich das Element **N553**
4. Navigieren Sie vom Element **N553** zum anderen Ende der Verbindung **E3944** vom Typ **Static Call**. Dort befindet sich das Element **N856**
5. Navigieren Sie von diesem Element auf dem schnellsten Wege zurück zum Startelement

Evaluation – Modell_2-2 (Chain)

Navigationsaufgabe

Wir bitten Sie, wieder einige Elemente in der angesagten Reihenfolge abzufliegen und dann zum Startelement zurück zu finden.

1. Es wird darum gehen einige Elemente ab zu fliegen und dann zum Startelement zurückzufinden.
2. Navigieren Sie vom Startelement **N731** zum anderen Ende der Verbindung **E4863** vom Typ **Static Call**. Dort befindet sich das Element **N544**
3. Navigieren Sie vom Element **N544** zum anderen Ende der Verbindung **E3262** vom Typ **Static Call**. Dort befindet sich das Element **N427**
4. Navigieren Sie vom Element **N427** zum anderen Ende der Verbindung **E3255** vom Typ **Dispatching Call**. Dort befindet sich das Element **N1080**
5. Navigieren Sie von diesem Element auf dem schnellsten Wege zurück zum Startelement

A.2 Fragebögen

Nachfolgend drucken wir die eingesetzten Online-Fragebögen ab. Insgesamt setzten wir drei Fragebögen ein: zu Beginn einen demographischen Fragebogen sowie am Ende zwei identische Fragebögen – einmal für das Head Mounted Display und einmal für das Desktop-System. Der Fragebogen am Ende ist ein kombinierter Fragebogen aus meCue[1] und SUS[2].

[1] siehe Kapitel 2.6.7
[2] siehe Kapitel 2.6.8

A.2.1 Demographischer Fragebogen

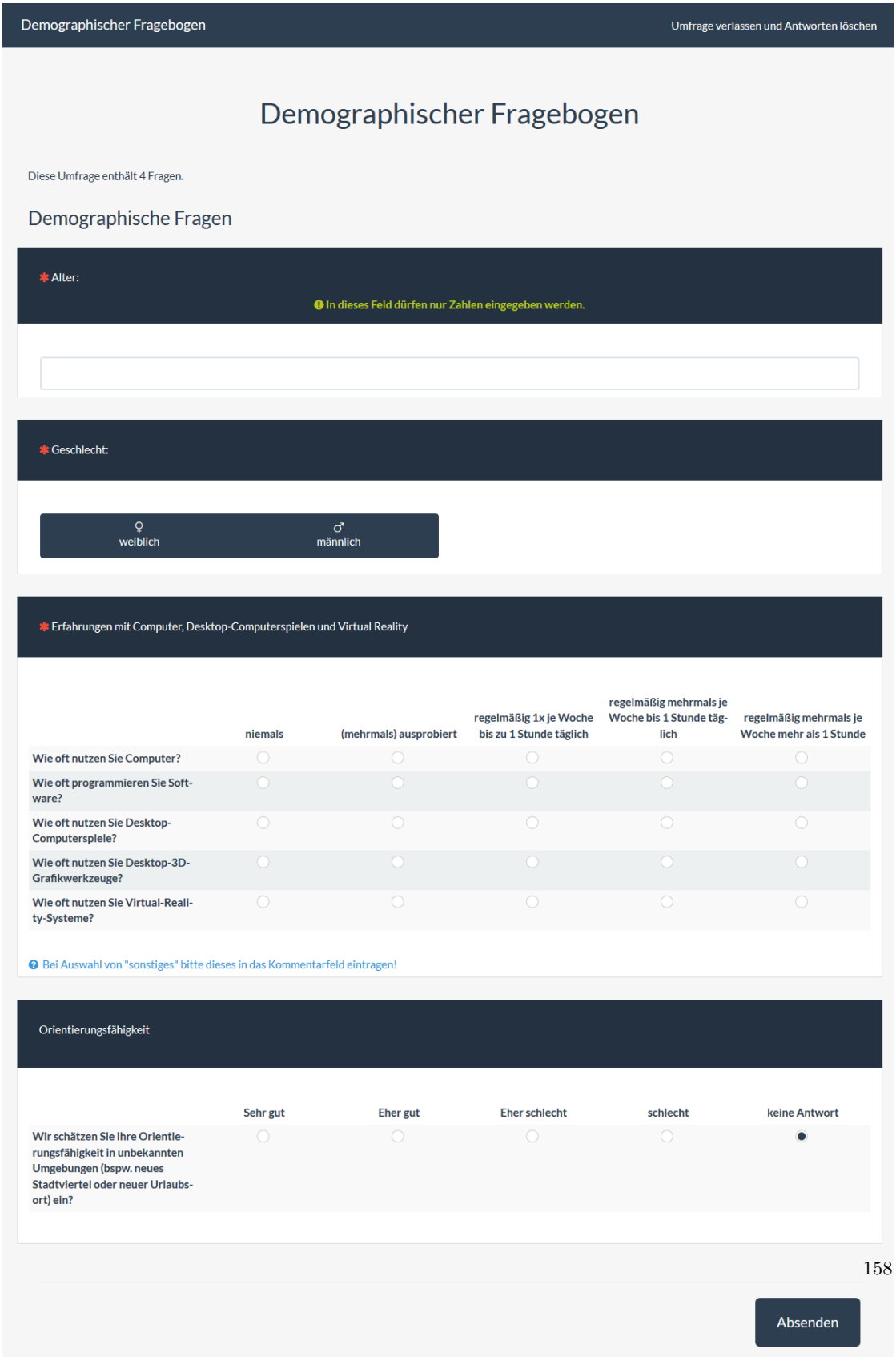


Abbildung A.1 Demographischer Fragebogen

A.2.2 Abschlussbewertungsfragebogen: meCue und SUS

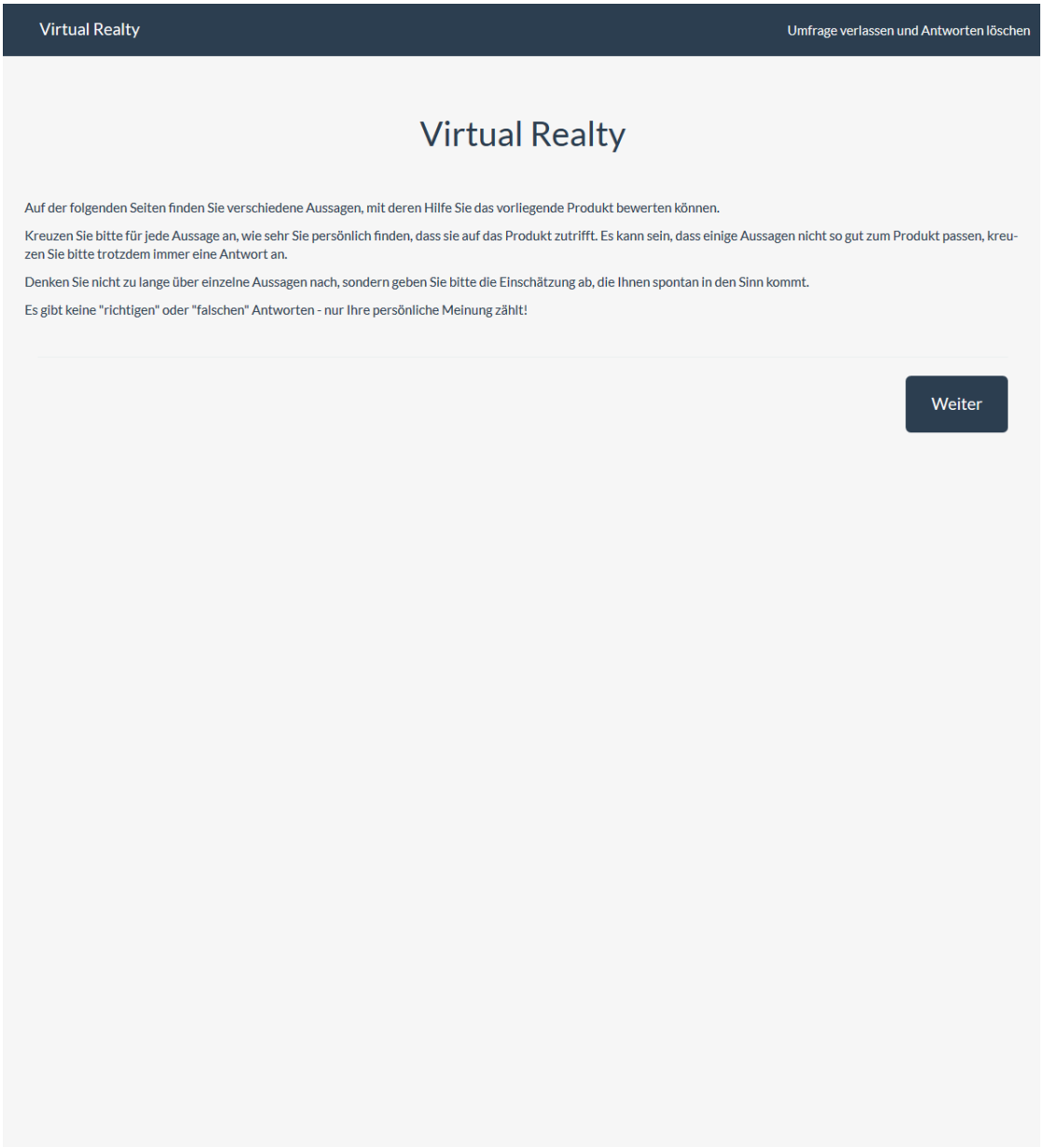


Abbildung A.2 Abschlussfragebogen – Vorseite

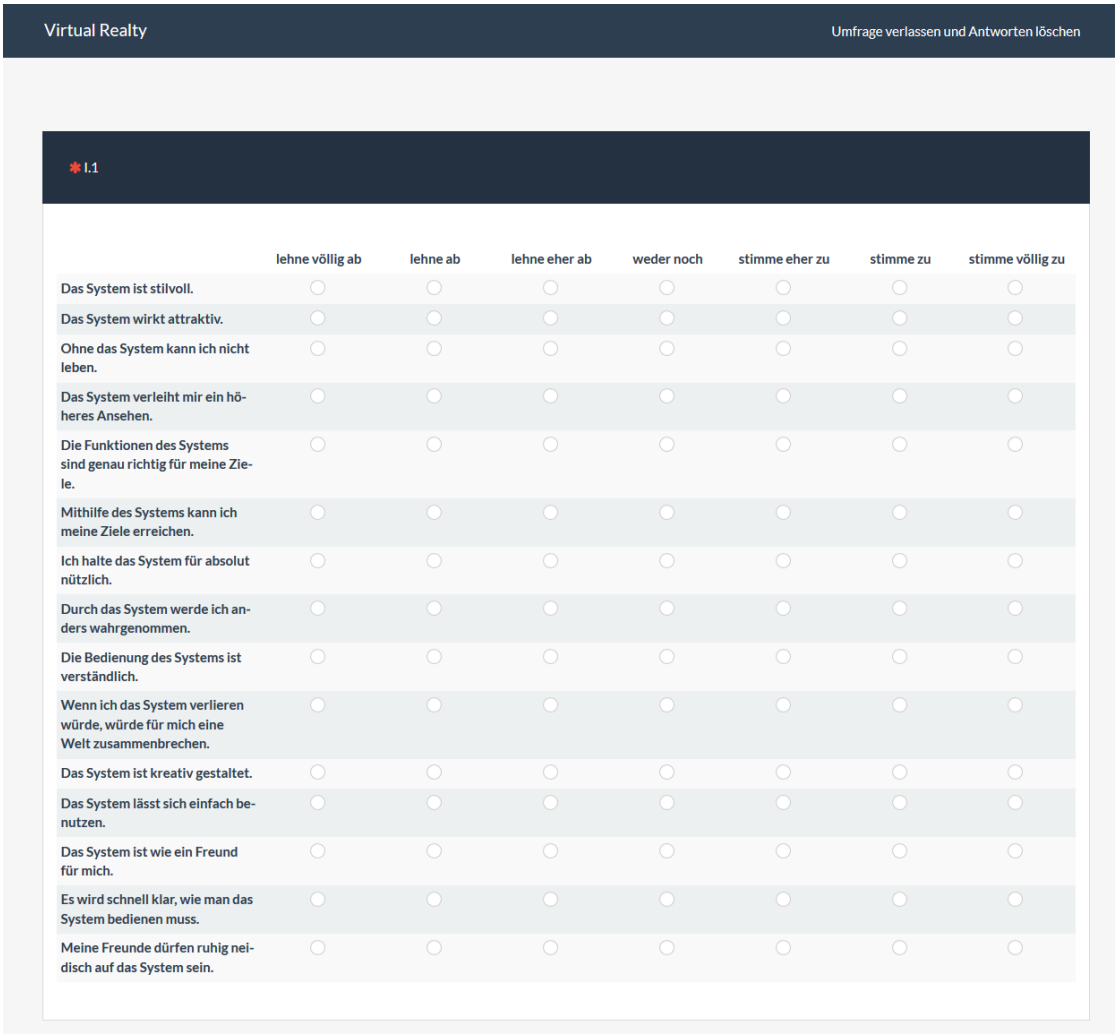


Abbildung A.3 Abschlussfragebogen I

II.1

	lehne völlig ab	lehne ab	lehne eher ab	weder noch	stimme eher zu	stimme zu	stimme völlig zu
Das System beschwingt mich.	○	○	○	○	○	○	○
Das System macht mich müde.	○	○	○	○	○	○	○
Das System nervt mich.	○	○	○	○	○	○	○
Das System entspannt mich.	○	○	○	○	○	○	○
Durch das System fühle ich mich erschöpft.	○	○	○	○	○	○	○
Durch das System fühle ich mich ausgeglichen.	○	○	○	○	○	○	○
Das System frustriert mich.	○	○	○	○	○	○	○
Das System stimmt mich euphorisch.	○	○	○	○	○	○	○
Durch das System fühle ich mich passiv.	○	○	○	○	○	○	○
Das System beruhigt mich.	○	○	○	○	○	○	○
Durch das System fühle ich mich fröhlich.	○	○	○	○	○	○	○
Das System verärgert mich.	○	○	○	○	○	○	○

III1

	lehne völlig ab	lehne ab	lehne eher ab	weder noch	stimme eher zu	stimme zu	stimme völlig zu
Wenn ich könnte, würde ich das System täglich nutzen.	○	○	○	○	○	○	○
Ich würde das System gegen kein anderes eintauschen.	○	○	○	○	○	○	○
Ich kann es kaum erwarten, das System erneut zu verwenden.	○	○	○	○	○	○	○
Im Vergleich zu diesem System wirken andere Systeme unvollkommen.	○	○	○	○	○	○	○
Ich würde mir genau dieses System jederzeit (wieder) zulegen.	○	○	○	○	○	○	○
Wenn ich mit dem System zu tun habe, vergesse ich schon mal die Zeit.	○	○	○	○	○	○	○

Abbildung A.4 Abschlussfragebogen II und III

Abbildung A.5 Abschlussfragebogen IV und SUS

A.3 Gruppenergebnisse meCue [MOR]

A.3.1 Desktop [MOR]

Modul	Skala	Median	Mittelw.	StAbw	Minimum	Maximum
Modul I	Nützlichkeit	5,00	5,26	1,31	1,00	6,33
	Benutzbarkeit	5,83	6,03	1,00	3,00	7,00
	visuelle Ästhetik	5,00	4,72	1,37	1,33	6,00
	Status	3,83	3,38	1,61	1,00	5,33
	Bindung	2,67	2,95	1,49	1,00	6,67
Modul II	positive Emotionen	4,00	4,01	1,17	1,00	5,67
	negative Emotionen	2,83	2,81	1,21	1,83	6,50
Modul III	Nutzungsintention	4,00	4,13	1,24	1,00	5,67
	Produktloyalität	3,33	3,33	0,78	1,33	4,00
Modul IV	Gesamturteil	3,50	3,2	2,4	-4,0	5,0

A.3.2 HMD [MOR]

Modul	Skala	Median	Mittelw.	StAbw	Minimum	Maximum
Modul I	Nützlichkeit	5,00	5,15	1,20	2,33	7,00
	Benutzbarkeit	6,00	5,69	0,95	3,67	7,00
	visuelle Ästhetik	5,67	5,38	1,14	2,00	6,33
	Status	4,00	4,23	1,50	1,00	6,00
	Bindung	2,33	2,51	0,97	1,00	4,00
Modul II	positive Emotionen	4,08	4,05	1,04	1,50	5,50
	negative Emotionen	2,83	2,90	1,12	1,50	5,67
Modul III	Nutzungsintention	4,50	4,85	1,04	2,33	6,33
	Produktloyalität	4,17	3,79	1,31	1,00	5,67
Modul IV	Gesamturteil	3,50	3,8	1,3	0,5	5,0

A.4 Vermessung einiger Apache-Commons Module

apache-commons	SIZE [SLOC]	#Java Dateien	#Classes Total	#Methods Total
commons-chain-1.2-src	8.010	97	94	799
commons-cli-1.4-src	6.681	50	73	398
commons-launcher	1.544	17	82	355
Demoprojekt	74	4	4	9

A.5 Risiko-Beurteilung [MOR]

When you have exhausted all possibilities,
remember this – you haven't.
—Thomas A. Edison

Ein Teil unseres Exposés für diese Bachelorarbeit war eine Risikoabschätzung inklusive möglicher Reaktionen. Von den genannten Risiken sind während unserer Arbeit nahezu alle eingetreten. Nachfolgend wollen wir kurz zu den einzelnen Risiken Stellung nehmen.

Systemkenntnisse

Die Einarbeitung in die C++-Programmierung von Unreal Engine dauert länger als geplant → Umstieg auf Blueprints, Einschränkung der Ziele

Dieses erste Risiko traf uns am heftigsten – auch wenn seine Nennung an erster Stelle nicht bewusst geschah. Unsere Vorerfahrungen

Evaluation

Experten für die Beurteilung des Systems lassen sich nicht finden, Evaluationsszenario schlägt fehl → auf Kommilitonen umsteigen, in anderen Arbeitsgruppen fragen

Über den Evaluationsprozess und die konkreten Aufgaben der Evaluation haben wir uns sehr lange Gedanken gemacht. Dabei bekamen wir große Hilfe aus der Arbeitsgruppe, haben aber bis zuletzt unsere konkreten Aufgabenstellungen stets hinterfragt und mehrfach umgebaut. Zielgenauigkeit und ein Informatikbezug war uns wichtig, erschwerte jedoch gleichzeitig die Definition der Aufgaben.

Teamarbeit

Team-Abstimmungen und Team-Aufgabenteilungen schwieriger als erwartet; gemeinsames Schreiben an der Arbeit → häufiges Treffen, eindeutige Aufgabenverteilung, agiler Ansatz

Bezüglich der Abstimmungen im Team und der Arbeit im Team gab es glücklicherweise keinerlei Schwierigkeiten.

Implementierung

Bei der C++-Programmierung oder durch Stromausfall geht Fortschritt verloren → GIT, regelmäßige BackUps auf eigenen Hardwareträgern

Daten haben wir glücklicherweise keine verloren, doch sind wir einige Male dadurch, dass wir irgendwann keine abzuarbeitende Planung mehr hatten, sondern nach dem Versuch-und-Irrtum-Prinzip arbeiteten, in Situationen gelangt, in denen das Zusammenführen verschiedener Versionsstände schwierig wurde oder zu inkonsistenten Zuständen führte. In diesen Fällen mussten wir unsere Systeme zeilenweise manuell zusammenführen.

Implementierung - Kanten

Bundled Edges lassen sich in der Unreal Engine nicht umsetzen oder sind zu rechenintensiv → lange Wartezeiten hinnehmen, Kantenbündelung während Laufzeit nachholen, nicht bündeln, Algorithmus vereinfachen (oder parametrisierbar machen)

Wir bereits vermutet, sind die gebündelten Kanten rechenintensiv – sobald allerdings die Rechnung einmal durchgeführt war, spielte dies keine Rolle mehr. Doch allein die Darstellung der Kanten stellte uns bis zum Ende der Arbeit vor große Schwierigkeiten.

In der Unreal Engine werden drei verschiedene grafische Elemente vorgestellt, die geeignet sind, geschwungene Verbindungslinien darzustellen.[3] Am Ende haben wir an der vierten und fünften gearbeitet, da die drei eingebauten Varianten inperformant waren. In der Tat müssen wir dadurch zwar die in der Risikoabschätzung angedeuteten langen Wartezeiten hinnehmen, doch können auch größere Softwarestrukturen flüssig darstellen. Durch eine Speicherung der erzeugten Meshes konnten wir Berechnungen im Vorfeld zu erledigen und für die Nutzung sowie Evaluation lediglich abzurufen.

Implementierung

Dynamisches Mapping und Erweiterbarkeit mit neuen Leveln misslingt → MVC-inspriertes Vorgehen

Unsere initiale Modellierung ging von einem Live-System[4] aus, in dem Daten In-Game, also zur Laufzeit der Anwendung ausgelesen, anhand der Mapping und sonstiger Parameter verarbeitet und schließlich interaktionsbereit dargestellt wird. Nachdem wir die einzelnen Teile für diesen

[3] siehe Kapitel 2.5.2
[4] im Weiteren auch In-Game-System genannt

Lösungsansatz festgestellt hatten, mussten wir bei der Integration feststellen, dass die Performance leider schon bei kleinen Software-Modellen so stark reduziert wird, dass keine flüssige Arbeit mehr mit der HTC Vive möglich ist – Übelkeit und Unlust war beim Tragen des HMD die Folge. wir stellten das komplette Live-System basierend auf einem Hinweis von Prof. Dr. Gabriel Zachmann testweise auf eine Vorabgenerierung der grafischen Elemente um. Obwohl diese Lösung alleine uns unserem Visualisierungsziel nur ein Stückchen näher gebracht hat, behielten wir diesen Ansatz bei und bauten ihn noch soweit aus, dass wir letztlich mit einigem Zeitverzug eine akzeptable Lösung mit einer akzeptablen Performance vorweisen können.

GXL-Dateien

Interpretation der inhaltlichen Struktur basiert auf falscher oder unvollständiger Annahme und schlägt fehl → Definition der GXL-Dateien erfragen, Software die GXL-Dateien erstellt selbst ausprobieren

Das eigentliche Auslesen der GXL-Dateien erzeugte zwar weit weniger Problem als anfänglich vermutet, allerdings verfügen die meisten der zur Verfügung gestellten Dateien lediglich über eine Metrik: Lines of Code.

Modellierung

Modellierung passt nicht zur Unreal Engine, HUD-Design nicht realisierbar mit echten Daten → frühzeitige Tests, an bestehenden Beispielen orientieren

Im Laufe des Projekts sind wir – beeinflusst von externen Ideen und Ideengebern sowie eigenen Recherchen – ein gutes Stück von der ursprünglichen Modellierung abgerückt: Aus einem geplanten In-Game-System ist ein In-Editor-System geworden. Dies ging einher mit einer kompletten Neumodellierung der Nutzerinteraktion, der Testleiterinteraktion (für die Festlegung der Rahmenparameter) sowie des dynamischen Mappings.

Während der gesamten Modellierung orientierten wir uns an veröffentlichten Projekten anderer Programmierer. Letztlich ist die entstandene Struktur nicht sonderlich kompliziert, würden wir aber behaupten, wir hätten diese wasserfallartig im Vorfeld geplant und dann realisiert, so würden wir flunkern. Unsere Systemstruktur hat sich eher unserem wachsenden Erfahrungs- und Kenntnisstand hinterherhinkend angepasst. Die Anzahl unserer geschriebenen Lines of Code bewegt sich im Rahmen von ein paar Tausend. Wir möchten diese allerdings gerne als *Netto-LOC* bezeichnen, der zugehörige Brutto-Lines-Of-Code-Wert ist um ein vielfaches größer, da er unsere Irr- und Umwege berücksichtigen würde. Wir versuchten zwischendurch auch, unsere Software – unserem Thema angemessen – zu visualisieren, scheiterten mit automatischen Verfahren jedoch

sehr schnell an der Größe der Unreal Engine, die im Quellcode vorliegt und mit berücksichtigt werden muss, um die Zusammenhänge und Abläufe zu verstehen. Handgezeichnete Skizzen kleiner statischer Ausschnitte und Abläufe war die zeitsparende Alternative.

Hardware

starke Abhängikeit von VR-Hardware und begrenzte Zugangszeiten erschaffen Bottle Necks, wir werden daher nicht rechtzeitig fertig → Teamviewer, Multi User System für Windows, Zugangschips, alleiniger Zugang zum Rechner

Dank der Software *ASTER* konnten wir den VR-Rechner zeitgleich mit zwei Nutzer nutzen, obwohl es sich um einen Windows-PC handelt. Lediglich die Nutzung der HTC Vive war in diesem Modus nicht möglich. Durch den Einbau einer zweiten Grafikkarte hätte es offenbar diese Option gegeben, doch haben wir die reale Notwendigkeit unterschätzt und – als sie gegen Projektende gekommen war – keine Muße mehr für die Umsetzung gehabt.

A.6 DemoProject GXL-Datei

Listing A.1 DemoProject.GXL

```
<?xml version="1.0" encoding="UTF-8"?>
<gxl xmlns:xlink="http://www.w3.org/1999/xlink">
   <graph id="Code Facts" edgeids="true">
      <node id="N1">
         <type xlink:href="File"/>
         <attr name="Source.Region.Length"><int>15</int></attr>
         <attr name="Source.Name"><string>Class_B.java</string></attr>
      </node>
      <node id="N10">
         <type xlink:href="Method"/>
         <attr name="Source.Name"><string>plusY</string></attr>
         <attr name="Source.Region.Length"><int>3</int></attr>
      </node>
      <node id="N11">
         <type xlink:href="Method"/>
         <attr name="Source.Name"><string>squareX</string></attr>
         <attr name="Source.Region.Length"><int>3</int></attr>
      </node>
      <node id="N12">
         <type xlink:href="Method"/>
         <attr name="Source.Name"><string>setX</string></attr>
         <attr name="Source.Region.Length"><int>3</int></attr>
```

```
23          </node>
24          <node id="N13">
25              <type xlink:href="Method"/>
26              <attr name="Source.Name"><string>toString</string></attr>
27              <attr name="Source.Region.Length"><int>3</int></attr>
28          </node>
29          <node id="N14">
30              <type xlink:href="Member"/>
31              <attr name="Source.Region.Length"><int>1</int></attr>
32              <attr name="Source.Name"><string>x</string></attr>
33          </node>
34          <node id="N15">
35              <type xlink:href="File"/>
36              <attr name="Source.Region.Length"><int>25</int></attr>
37              <attr name="Source.Name"><string>Class_A2.java</string></attr>
38          </node>
39          <node id="N16">
40              <type xlink:href="Class"/>
41              <attr name="Source.Region.Length"><int>21</int></attr>
42              <attr name="Source.Name"><string>Class_A2</string></attr>
43          </node>
44          <node id="N17">
45              <type xlink:href="Method"/>
46              <attr name="Source.Name"><string>potXY</string></attr>
47              <attr name="Source.Region.Length"><int>3</int></attr>
48          </node>
49          <node id="N18">
50              <type xlink:href="Method"/>
51              <attr name="Source.Name"><string>setX</string></attr>
52              <attr name="Source.Region.Length"><int>3</int></attr>
53          </node>
54          <node id="N19">
55              <type xlink:href="Method"/>
56              <attr name="Source.Name"><string>toString</string></attr>
57              <attr name="Source.Region.Length"><int>3</int></attr>
58          </node>
59          <node id="N2">
60              <type xlink:href="Class"/>
61              <attr name="Source.Region.Length"><int>11</int></attr>
62              <attr name="Source.Name"><string>Class_B</string></attr>
63          </node>
64          <node id="N20">
65              <type xlink:href="Member"/>
66              <attr name="Source.Region.Length"><int>1</int></attr>
67              <attr name="Source.Name"><string>x</string></attr>
68          </node>
69          <node id="N21">
70              <type xlink:href="Package"/>
71              <attr name="Source.Name"><string>myPackage</string></attr>
```

```
72          </node>
73          <node id="N22">
74             <type xlink:href="Package"/>
75             <attr name="Source.Name"><string>package_A</string></attr>
76          </node>
77          <node id="N23">
78             <type xlink:href="Package"/>
79             <attr name="Source.Name"><string>package_A1</string></attr>
80          </node>
81          <node id="N24">
82             <type xlink:href="Package"/>
83             <attr name="Source.Name"><string>package_B</string></attr>
84          </node>
85          <node id="N3">
86             <type xlink:href="Method"/>
87             <attr name="Source.Name"><string>toString</string></attr>
88             <attr name="Source.Region.Length"><int>5</int></attr>
89          </node>
90          <node id="N4">
91             <type xlink:href="Member"/>
92             <attr name="Source.Region.Length"><int>1</int></attr>
93             <attr name="Source.Name"><string>Class_As</string></attr>
94          </node>
95          <node id="N5">
96             <type xlink:href="File"/>
97             <attr name="Source.Region.Length"><int>9</int></attr>
98             <attr name="Source.Name"><string>Class_A.java</string></attr>
99          </node>
100         <node id="N6">
101            <type xlink:href="Class"/>
102            <attr name="Source.Region.Length"><int>4</int></attr>
103            <attr name="Source.Name"><string>Class_A</string></attr>
104         </node>
105         <node id="N7">
106            <type xlink:href="Method"/>
107            <attr name="Source.Name"><string>toString</string></attr>
108            <attr name="Source.Region.Length"><int>3</int></attr>
109         </node>
110         <node id="N8">
111            <type xlink:href="File"/>
112            <attr name="Source.Region.Length"><int>25</int></attr>
113            <attr name="Source.Name"><string>Class_A1.java</string></attr>
114         </node>
115         <node id="N9">
116            <type xlink:href="Class"/>
117            <attr name="Source.Region.Length"><int>21</int></attr>
118            <attr name="Source.Name"><string>Class_A1</string></attr>
119         </node>
120         <edge id="E1" from="N2" to="N1"><type xlink:href="Enclosing"/></edge>
```

```
121         <edge id="E10" from="N13" to="N9"><type xlink:href="Enclosing"/></edge>
122         <edge id="E11" from="N14" to="N9"><type xlink:href="Enclosing"/></edge>
123         <edge id="E12" from="N16" to="N15"><type xlink:href="Enclosing"/></edge>
124         <edge id="E13" from="N17" to="N16"><type xlink:href="Enclosing"/></edge>
125         <edge id="E14" from="N18" to="N16"><type xlink:href="Enclosing"/></edge>
126         <edge id="E15" from="N19" to="N16"><type xlink:href="Enclosing"/></edge>
127         <edge id="E16" from="N20" to="N16"><type xlink:href="Enclosing"/></edge>
128         <edge id="E17" from="N19" to="N7"><type xlink:href="Dispatching_Call"/><↩
                /edge>
129         <edge id="E17" from="N9" to="N7"><type xlink:href="Dispatching_Call"/></↩
                edge>
130         <edge id="E18" from="N9" to="N7"><type xlink:href="Extend"/></edge>
131         <edge id="E19" from="N16" to="N7"><type xlink:href="Extend"/></edge>
132         <edge id="E2" from="N3" to="N2"><type xlink:href="Enclosing"/></edge>
133         <edge id="E20" from="N22" to="N21"><type xlink:href="Enclosing"/></edge>
134         <edge id="E21" from="N24" to="N21"><type xlink:href="Enclosing"/></edge>
135         <edge id="E22" from="N23" to="N22"><type xlink:href="Enclosing"/></edge>
136         <edge id="E23" from="N1" to="N24"><type xlink:href="Enclosing"/></edge>
137         <edge id="E24" from="N5" to="N22"><type xlink:href="Enclosing"/></edge>
138         <edge id="E25" from="N15" to="N22"><type xlink:href="Enclosing"/></edge>
139         <edge id="E26" from="N8" to="N23"><type xlink:href="Enclosing"/></edge>
140         <edge id="E27" from="N1" to="N5"><type xlink:href="Import"/></edge>
141         <edge id="E28" from="N18" to="N20"><type xlink:href="Member_Set"/></edge↩
                >
142         <edge id="E29" from="N12" to="N14"><type xlink:href="Member_Set"/></edge↩
                >
143         <edge id="E3" from="N4" to="N2"><type xlink:href="Enclosing"/></edge>
144         <edge id="E30" from="N19" to="N20"><type xlink:href="Member_Use"/></edge↩
                >
145         <edge id="E31" from="N13" to="N14"><type xlink:href="Member_Use"/></edge↩
                >
146         <edge id="E32" from="N3" to="N4"><type xlink:href="Member_Use"/></edge>
147         <edge id="E33" from="N4" to="N6"><type xlink:href="Of_Type"/></edge>
148         <edge id="E34" from="N7" to="N13"><type xlink:href="Override"/></edge>
149         <edge id="E35" from="N7" to="N19"><type xlink:href="Override"/></edge>
150         <edge id="E4" from="N6" to="N5"><type xlink:href="Enclosing"/></edge>
151         <edge id="E5" from="N7" to="N6"><type xlink:href="Enclosing"/></edge>
152         <edge id="E6" from="N9" to="N8"><type xlink:href="Enclosing"/></edge>
153         <edge id="E7" from="N10" to="N9"><type xlink:href="Enclosing"/></edge>
154         <edge id="E8" from="N11" to="N9"><type xlink:href="Enclosing"/></edge>
155         <edge id="E9" from="N12" to="N9"><type xlink:href="Enclosing"/></edge>
156     </graph>
157 </gxl>
```