UNIVERSITÄT KOBLENZ · LANDAU
Fachbereich 4: Informatik

DLR Deutsches Zentrum für Luft- und Raumfahrt e.V.

Visualisierung OSGi-basierter Software Architektur in Augmented Reality

Masterarbeit

zur Erlangung des Grades Master of Science (M.Sc.)
im Studiengang Computervisualistik

vorgelegt von

Yessika Katrin Legat

Erstgutachter:	Prof. Dr.-Ing. Stefan Müller (Institut für Computervisualistik, AG Computergraphik)
Zweitgutachter:	Andreas Schreiber DLR (Simulations- und Softwaretechnik, Intelligente und verteilte Systeme)

Koblenz, im Juni 2018

Erklärung

Ich versichere, dass ich die vorliegende Arbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe.

	Ja	Nein
Mit der Einstellung der Arbeit in die Bibliothek bin ich einverstanden.	☐	☐

.....

(Ort, Datum) (Unterschrift)

Zusammenfassung

Es wird ein Augmented-Reality Ansatz zur Erforschung modularer OSGi-Softwaresysteme präsentiert. Der Prototyp wird unter der Verwendung der Microsoft HoloLens implementiert. Module, wie Komponenten und Packages, werden in einer virtuellen Stadt dargestellt. Dieser Ansatz ermöglicht es dem Anwender, die Software-Architektur mittels intuitiver Navigation zu erkunden: Spracheingabe, Blickpunkt- und Gestenkontrolle. Eine multifunktionale Benutzeroberfläche wird vorgestellt, die für verschiedene Zielgruppen adaptiert werden kann. Viele veröffentlichte Visualisierungen weisen keine klare Zielgruppendefinition auf. Das Konzept kann leicht auf andere Darstellungsformen, wie beispielsweise der Inselmetapher übertragen werden. Erste Ergebnisse einer Evaluierung, die mittels kleiner strukturierter Interviews gewonnen werden konnten, werden präsentiert. Die Probanden mussten vier Programm-verständnis Aufgaben lösen und ihren Aufwand, sowie ihre Arbeitsbelastung einschätzen. Die Ergebnisse bilden eine gute Grundlage für weitere Forschung im Bereich der Software-Visualisierung in Augmented Reality.

We present an augmented reality approach to explore modular software systems based on OSGi by using the Microsoft HoloLens. Modules, such as components and packages, are displayed in a virtual city. This approach allows users to explore software architecture with intuitive navigation: Voice, gaze, and gesture control. We present a multipurpose user interface, which can be adapted for different target groups since many published visualizations lacks in a clear target group definition. Our concept is easily transferable to other forms of representation, such as the island metaphor. We give early results of an evaluation, which we conducted by small structured interviews where participants had to solve four program comprehension tasks and rate their effort on questions to obtain workload estimates. The results provide a good basis for further research in AR for software visualization.

Inhaltsverzeichnis

1 Einleitung

Große Softwareprojekte sind meist komplex und nicht intuitiv. Software als solches ist schwierig zu verbildlichen, abstrakte Klassen oder Services können nicht leicht verständlich abgebildet werden. Besonders bei großen Projekten ist eine Betrachtung auf einer höheren Abstraktionsebene sinnvoll, um einen Gesamtüberblick zu erhalten und Zusammenhänge zu verstehen [1, 2]. Daher existieren bereits einige Verfahren in 2D und 3D, um Software Architektur greifbarer zu gestalten [3]. Viele Visualisierungs-Methoden für Software Architektur verwenden keine Metaphern. In der jüngeren Zeit wird die Zuhilfenahme dieser Abstraktionsmethoden als relevantes Forschungsgebiet angesehen.
Dank vielen entwicklerunterstützenden IDEs (integrated development environment) beschränkt sich die Arbeit eines Softwareentwicklers heutzutage vermehrt auf das Schreiben von Quellcode und die abstrakte Software Architektur bleibt zumeist unangetastet im Hintergrund. Soll sich ein neuer Entwickler in ein bestehendes Projekt einarbeiten, so wird oftmals viel Zeit in das Verstehen der Software investiert [4]. Ein Überblick über das System kann die Arbeit erleichtern und dadurch die Einarbeitungszeit verringern. Zudem wird das Verständnis der Software gestärkt [3].
Viele Visualisierungsansätze scheitern an der Praxistauglichkeit. Dies trifft besonders auf komplexe OSGi-basierte Anwendungen zu, obwohl diese oftmals einen großen Umfang besitzen und so eine Veranschaulichung besonders vorteilhaft wäre. Zusätzlich zu allen Konzepten der „normalen“ Java Programmierung werden Konzepte von Bundles und Services hinzugenommen, um eine Modularisierung des Codes zu ermöglichen. Diese Konzepte stellen eine bedeutende Rolle in der Software Architektur dar, weswegen herkömmliche Klassendiagramme zur Darstellung nicht ausreichen. Um die Gesamtheit einer Software zu fassen, reicht der eigentliche Quellcode nicht aus und es müssen weitere Informationen gegeben werden. In OSGi-Projekten existieren zudem noch andere XML-Textdateien mit wichtigen Inhalten über den Aufbau. Bisweilen existieren kaum Tools, welche diese Besonderheiten berücksichtigen [5]. Ein Ansatz zur Visualisierung OSGi-basierter Architektur in VR (Virtueller Realität) wurde 2017 [4] entwickelt und dient als Grundlage für diese Arbeit.

In der vorliegenden Arbeit werden kurz die Grundlagen (Kapitel 2) zu Software Architektur Visualisierung im Allgemeinen aufgelistet und eine detaillierte Literaturrecherche zu interessanten Arbeiten mit neuen Darstellungsmedien, wie Augmented und Virtual Reality (Kapitel 3) vorgestellt. Die Microsoft *HoloLens* bietet als erstes tragbares MR-HMD viel Potenzial für 3D-Darstellungen. Die *HoloLens* ist nicht auf Räumlichkeiten beschränkt, denn kleinster Raum reicht zur Verwendung aus. Die Besonderheiten und Interaktionsmöglichkeiten der *HoloLens* werden herausgearbei-

tet und vorgestellt (Kapitel 4). Um eine immersive Anwendung zu entwickeln müssen die benötigten speziellen Anforderungen erforscht werden (Kapitel 5). Eine in Augmented Reality (AR) verwirklichte Software Visualisierung wird in dieser Arbeit konzipiert und implementiert (Kapitel 7). Die Umsetzung in AR ermöglicht eine weitere Dimension der Erkundung und Interaktions- Möglichkeiten. Die Visualisierung soll nicht nur eine Variante der Software Architektur darstellen sondern eine Exploration dieser ermöglichen. Dazu muss der Benutzer mit der Visualisierung interagieren und beispielsweise Informationen selektieren oder filtern können. In diesem Kontext findet eine Visualisierungsvariante mit der sogenannten City-Metapher Anwendung (Kapitel 7). Erste Nutzertests (Kapitel 8) werden ausgeführt, um die Benutzerfreundlichkeit der Applikation herauszufinden. Dabei sollen die Probanden Programmverständnis-Aufgaben lösen und anschließend subjektives Feeback über den benötigten Arbeitsaufwand geben. Abschließend werden die Ergebnisse ausgewertet und zukünftige Arbeit (Kapitel 9) beschrieben.

Diese Masterarbeit wird in Kooperation mit dem Deutschen Zentrum für Luft- und Raumfahrt, Einrichtung „Simulations- and Softwaretechnik“, in Köln durchgeführt. In dieser Abteilung wird unter anderem das *Remote Component Environment* (RCE) entwickelt[1]. RCE ist eine OSGi-Software, weswegen die prototypische Augmented Realitiy Anwendung speziell für OSGi-Software entwickelt wird. Das Visualisierungs-Verfahren ist keineswegs auf diese Software beschränkt.

[1] `www.dlr.de/sc/desktopdefault.aspx/tabid-5625/9170_read-17513` zuletzt eingesehen am 02.01.2018.

2 Grundlagen

Java[2] ist eine statisch typisierte, objektorientierte Programmiersprache. Der Java-Quellcode wird vor der Ausführung in Bytecode kodiert und dadurch plattformunabhängig maschinenlesbar gemacht. Zur Laufzeit wird Bytecode mittels der Java Virtual Machine (JVM) kompiliert. Dies bedeutet, dass lediglich eine JVM auf dem Ziel-Betriebssystem installiert sein muss, um den Code auszuführen.
Zur Modularisierung und Aufteilung des Quellcodes werden Klassen in Packages organisiert. Diese vermeiden Namenskonflikte durch hierarchische Namensräume. Dieses Modulkonzept reicht bei dynamischer und komplexen Programmen nicht aus, weswegen die OSGi-Spezifikation Anwendung findet [5, 6, 7].

2.1 Open Service Gateway Initiative Service (OSGi)

Die Open Service Gateway Initiative Service (OSGi) Plattform ist ein dynamisches Modulsystem für Java. Es handelt sich um eine Softwareplattform, welche die dynamische Integration und das Management von Softwarekomponenten (Bundles) und Services ermöglicht. Bundles und Services können zur Laufzeit installiert, gestartet, gestoppt und deinstalliert werden. Dies geschieht während der Laufzeit, die Plattform muss nicht angehalten werden [6].
Die folgende Auflistung erläutert die Besonderheit der einzelnen Komponenten:

Bundles Eine fachlich oder technisch zusammenhängende Einheit von (mehreren) Klassen und Ressourcen, die eigenständig im Framework in- oder deinstalliert werden können. Der eigentliche Inhalt ist für andere Bundles erst nach dem Export sichtbar, wenn das Package von dem nutzenden Bundle importiert wird. Es handelt sich demnach um eine Art Abkapslung des Inhalts. Der Inhalt des Bundles ist immer lauffähig, testbar und vollständig. In einer Manifestdatei wird der Im- und Export der Packages festgehalten. Zur Laufzeit sorgt das Framework für die Bereitstellung jedes geforderten Packages. Ist ein Package nicht im System auffindbar, kann das Bundle nicht zur Ausführung gebracht werden [7].

Services Bundles können zur weiteren Entkopplung Services verwenden. Dabei handelt es sich um Objekte in Java, die einen Interface-Namen besitzen. Ein Service wird registriert und von der zentralen Service Registry verwaltet. Bundles, die einen bestimmten Dienst benötigen,

[2] `https://www.java.com/de/`

können den entsprechenden Service von der Service Registry abfragen. Dabei werden keine Informationen über den konkreten Inhalt der Bundles an den fragenden Service weitergereicht [7].

Managment Agent Der im Framework eingebaute Management Agent verwaltet die Bundles (installieren und deinstallieren, starten und stoppen) und besteht meist selbst aus einem oder mehreren Bundles. Im einfachsten Fall stellt das sogenannte Management Bundle eine textbasierte Konsole zur Verfügung [7].

Die Basiskomponente bildet das OSGi-Framework, welches einen Container für Bundles und Services bereitstellt. Das Framework fördert die Modularität der Software. Zusammengehörige Programmteile werden gruppiert und stellen Services für andere Module zur Verfügung. Die Wiederverwendung und Wartbarkeit des Codes wird dadurch gefördert und die Komplexität durch Zerlegung verringert. In diesem Sinne sind Klassen, Methoden, Packages und Bundles jeweils eine Art Modul. Im weiteren Verlauf werden diese allerdings als Komponente bezeichnet. Die Bezeichnung Modul wird hingegen nur noch für die Komponenten in der Abstraktionsebene der Bundles verwendet. In OSGi können nicht nur Abhängigkeiten unter Klassen sondern auch zwischen Bundles bestehen [7].
Die gleichen Vorteile könnten auch mit anderen Mechanismen und Mitteln erzeugt werden. OSGi bietet jedoch ein speziell darauf ausgerichtetes Tool, welches die Verwendung vereinfacht. Bekannte Implementierungen sind Eclipse Equinox, Knopflerfish oder Apache Felix [8]. Bei jedem Eclipse-Plugin handelt es sich um ein OSGi-Bundle.

2.2 Augmented Reality

Augmented Reality (AR) [9, 10] bezeichnet die erweiterte Realität. Sie ist ein Teil der Mixed Reality (MR), der gemischten Realität. MR setzt sich aus der Augmented Reality und der Augmented Virtuality (AV) zusammen. Der Grad des realen oder virtuellen Umfelds bestimmt dabei die Zugehörigkeit. Abbildung 1 veranschaulicht dies. Augmented Reality liegt nahe am realen Umfeld des Nutzers. Nach [11] wird in der erweiterten Realität im Gegensatz zur virtuellen Realität keine Welt nachgebaut sondern die reale um virtuelle Informationen erweitert. Sie erlaubt es dem Nutzer die reale Welt zu sehen, welche mit 3D-Objekten überlagert wird. Allerdings werden nicht nur Objekte eingeblendet, die Objekte können auch mit realen Objekten zusammengesetzt werden. Zusätzlich kann der Nutzer mit diesen interagieren. Es handelt sich um dreidimensionale virtuelle Gegenstände, die perspektivisch korrekt hinzugefügt werden. Überlagerungen und realistische Verdeckungen der Objekte bestärken den Eindruck der Echtheit. AR ergänzt die reale Welt: Im Idealfall existieren für den Benutzer

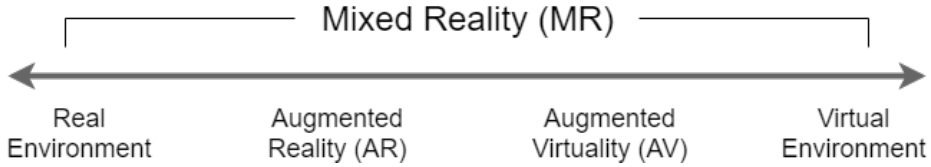


Abbildung 1: Veranschaulichung der Einordnung AR aus [11].

die imaginären und realen Objekte zugleich.
Die technologischen Anforderungen der Augmented Reality sind viel höher als die der Virtual Reality, weshalb die Entwicklung der Augmented Reality länger dauerte als die der virtuellen Realität [12]. Die Schlüsselkomponenten für den Aufbau eines Augmented Reality-Systems sind jedoch seit der Pionierarbeit von Sutherland [13, 14] in den 60er Jahren gleich geblieben. Displays, Tracker, Grafikcomputer und Software bleiben in vielen Augmented Reality-Erlebnissen unverzichtbar [12].
Eine AR-Anwendungen soll die reale und imaginäre Welt kombinieren, Interaktivität in Echtzeit unterstützen und zudem einen dreidimensionalen Bezug zwischen virtuellen und realen Objekten herstellen. Das Einblenden von zusätzlichen Objekten oder Instruktionen kann Benutzern Informationen bereitstellen [11, 15].

2.3 Head Mounted Displays

Ein Head Mounted Display (HMD) ist ein visuelles Ausgabegerät, welches auf dem Kopf getragen wird. Je nach Gerät, wird direkt auf die Netzhaut oder ein in Augennähe angebrachtes Display projiziert. Zumeist handelt es sich um Geräte in Brillen-Form [16, 15].
Die ersten Konzepte für tragbare Computer oder persönliche Computer Interfaces werden schon seit den späten 1950er diskutiert. Zuerst nur in Science-Fiction Filmen und später dann in realen technischen Forschungen [12, 14, 17].

Augmented Reality (AR) HMDs sind Erweiterungen der Smart Glasses zu einem „see-trough“ Device mit größerem FOV (Field of View). Meistens ist das Head-Tracking bereits integriert. Probleme in der Umsetzung sind die „see-trough“ Qualität, der kleine Sichtwinkel und deren Reichweite [18].

Mixed Reality (MR) HMDs sind eine Weiterentwicklung der Standard AR HMDs. Sie bieten ein akkurates Head-Tracking, Gestenerkennung und eingebaute Tiefensensoren ermöglichen 3D-Mapping der Umgebung. Tiefenmapping kann zur Berechnung realistischer Beleuchtung verwendet werden, wodurch eine erhebliche visuelle Verbesserung gegenüber herkömmlichen AR HMDs erreicht wird [18].

Abbildung 2: Abbildung der *HoloLens* und ihren eingebauten Sensoren [23].

Spezielle Virtual Reality HMDs, wie der Oculus Rift und das HTC Vive, könnten so modifiziert werden, dass sie als AR HMDs funktionieren. Dies wird ermöglicht, indem eine Kamerakomponente hinzugefügt und zusätzlich die Aufnahmen anstatt der virtuellen Welt verarbeitet und anzeigt werden. Die hierbei benötigte, hohe Rechenleistung gilt als größter Nachteil, weswegen für die eigentliche Anwendung weniger Ressourcen zur Verfügung stehen [19]. AR/MR-Devices ermöglichen die Exploration der Umgebung mit sechs Freiheitsgraden (DOF: Degrees of Freedom) [12].

2.3.1 Microsoft *HoloLens*

Die *HoloLens* ist der erste eigenständige, holografische tragbare Computer, der es ermöglicht mit Hologrammen zu interagieren. Steuerungsmöglichkeiten zum Rotieren, Skalieren oder Verschieben der Hologramme ergeben sich über die Kombination des Blickpunktes, Spracheingabe, einen Hand-Klick-Controller und der Gesten-Steuerung [20, 21].
Die *HoloLens* ist ein auf nicht Ethernet basierendes Mixed Reality HMD. Dadurch benötigt das Gerät zusätzlich eine speziell angepasste GPU, die sogenannte HPU: Holographic Processing Unit. Die *HoloLens* verfügt über 2GB Arbeitsspeicher, davon 1GB CPU und 1GB HPU [18, 19].
Insgesamt sind 5 Kameras verbaut: Davon sitzen zwei links, zwei rechts und eine in der Mitte des Gerätes. Die Kameras verfolgen die Kopfbewegungen. Die mittlere Kamera ist zudem in der Lage, Screenshots und Videos aufzunehmen. Zusätzlich sind Inside-out Sensoren (Infrarot) mit eingebauten dualen Head Tracking Kameras auf jeder Seite und 3D-Tiefenkameras eingebaut, welche unter anderem für sensitive Gestenerkennung und die Lokalisierung der Objekte sorgen. Dadurch kann sehr schnell eine grobe Karte der Umgebung und der darin befindlichen Objekten angelegt werden. Diese verfeinert sich, wenn der Nutzer sich bewegt oder mit der Umgebung interagiert [22]. Der Großteil der Sensoren ist an einer Art Bügel im vorderen Bereich des Geräts angebracht (vgl. Abbildung 2).

Die *HoloLens* verfügt über ein Mikrofon, sodass der Benutzer während der Laufzeit Sprachbefehle als Eingabemechanismus nutzen kann. So kann

Prozessor	Intel x86-Prozessorarchitektur, HPU 1.0
Software	Windows 10
Speicher	64 Gigabyte
RAM	2 Gigabyte
Video	2 Megapixel Foto-, HD-Videokamera
Gewicht	579 Gramm

Tabelle 1: Technische Spezifikation *HoloLens* [18, 23]

eine Anwendung mit einem Sprachbefehl gestartet, Manipulationen erfolgen oder beendet werden [22].
Die Winkelauflösung des Devices entspricht etwa dem Sichtbereich des menschlichen Auges, sie liegt bei circa 60 Grad [12, 18]. Die Limitation dieses HMDs liegen in der Leistungsfähigkeit der eingebauten HPU und dem noch zu kleinen Sichtfeld. Die *HoloLens* bietet aber vielversprechende Grundlagen für weitere Arbeiten in AR bzw. MR. Die technischen Details sind in Tabelle 1 zusammenfassend aufgeführt. Die derzeit zugängliche Brille ist eine frühe Entwickler-Version, eine weiterentwickelte Verbraucher-Version soll 2019 veröffentlicht werden [19].

Zusätzlich ist die optische Auflösung entscheidend für den Erfolg der Verschmelzung von virtuellen und realen Szenen. Wenn das Auge auf das virtuelle Bildschirmbild fokussiert und dann auf ein Objekt in vier Metern Entfernung schaut, kann es zu einer Fehlanpassung des Fokus kommen. Dies wirft den Benutzer aus der erzeugten Illusion. Dieses Problem soll durch die so genannten Lichtfeld-Displays gelöst werden, die zu den scheinbaren echten Brennweiten passen. Die meisten Menschen verfügen nicht über das technische Fachwissen um zu erklären, warum verschiedene Augmented Reality-Lösungen nicht funktionieren. Sie sind trotzdem in der Lage zu bemerken, falls etwas nicht korrekt ist und dadurch komisch wirkt. Binnen Sekunden können Fehler erkannt werden [12].

Angesichts der neuen Augmented Reality-Technologie sind keine langfristigen Studien über Gesundheitsrisiken bekannt. Einige Risiken sind jedoch bei der Verwendung des Devices sofort erkennbar: Es besteht die Gefahr von Augenschmerzen, da die Augen häufig über einen längeren Zeitraum auf eine Ebene innerhalb eines Zentimeters Abstand zu den Augen fokussiert werden müssen. Auch wenn dies bei jeder Art von Digitalanzeige ein Risiko darstellt, gibt besonders die Position des *HoloLens* -Displays Anlass zur Sorge. Ein weiteres Risiko ist die Belastung des Nackenbereichs. Das Headset ist erstaunlich leicht für die bereitgestellte Leistung, jedoch mit seinen fast 600 Gramm ziemlich schwer für einen längeren Gebrauch. Stellt der Benutzer die Kopfbänder nicht perfekt ein, kann das Gewicht und die Haltung zu Nacken-Verspannungen führen [19].

Während bei AR/MR Anwendungen die Gefahr an Cyber Sickness [12] zu erkranken deutlich gering ausfällt und kaum dokumentiert ist, ist sie auf keinen Fall auszuschließen. Die Cyber Sickness wird durch den Konflikt zwischen dem was das Gehirn und der Körper des Benutzers registrieren und dem was seine Augen sehen (meistens gepaart mit großen Verzögerungen des Gesehenen) hervorgerufen. Diese Diskrepanz wird als Toxin interpretiert und der menschliche Körper probiert alles, um dieses Toxin herauszubekommen (z.B. Erbrechen). Jedoch hat jede Person andere Schwellenwerte und Toleranzgrenzen, weswegen nicht jeder betroffen ist. Vor allem bei Virtual Reality tritt dieses Phänomen häufig auf, da der Nutzer komplett von der realen Welt abgeschottet ist und keinerlei Orientierungspunkte zur Verfügung stehen. Eine zu geringe Framerate kann auch bei der *HoloLens* zu Cyber Sickness führen. Eine zu hoch gewählte Rate kann hingegen nicht schnell genug von der *HoloLens* berechnet werden. Dies liegt an der eigenständigen und daher limitierten HPU des Geräts [22].

2.4 Unity

Die Unity Game Engine [24] ist hauptsächlich eine Plattform für Spiele-Entwicklung und auch AR- oder VR- Anwendungen. Sie bietet anhand ihrer integrierten Funktionen, speziell für 2D und 3D Programme, eine gute Unterstützung für Entwickler. Dadurch kann der Entwickler sich vermehrt auf den Inhalt des Programms fokussieren. Unity bietet bereits ein inkludiertes Physik-, Rendering-, Network- und Animationssystem. Die Erweiterbarkeit mit vielen Frameworks und Plugins ist zudem ein großer Vorteil dieser Entwicklungsumgebung.
Speziell für MR und AR wurde das Mixed Reality Toolkit entwickelt, welches eine Erweiterung für Unity darstellt. Grundeinstellungen zur Kamera und Eingabemechanismen werden unterstützt. Da dieses Toolkit aktuell sehr stark weiterentwickelt wird, existieren noch einige Bugs und inkonsistente Versionen.

2.5 Mixed-Reality Toolkit

Das Mixed-Reality-Toolkit [25] ist eine Sammlung aus Skripten und Komponenten zur Beschleunigung der Entwicklung von Anwendungen, die speziell auf Microsoft *HoloLens* und Windows Mixed-Reality-Headsets ausgerichtet sind. Durch diese Sammlung werden Grundkomponenten und Voreinstellungen (wie spezifizierte Kamera- und Eingabeeinstellungen) zur Unterstützung der Arbeit mit Mixed-Reality-Anwendungen geschaffen.
Das Mixed-Reality-Toolkit-Unity verwendet Quellcode von der Basis des Mixed-Reality-Toolkits und vereinfacht das Entwickeln unter Unity. Die Überschneidungen sind in Abbildung 3 ersichtlich. Durch das ständige Update des Toolkits werden viele Bugs behoben und auch weitere Features

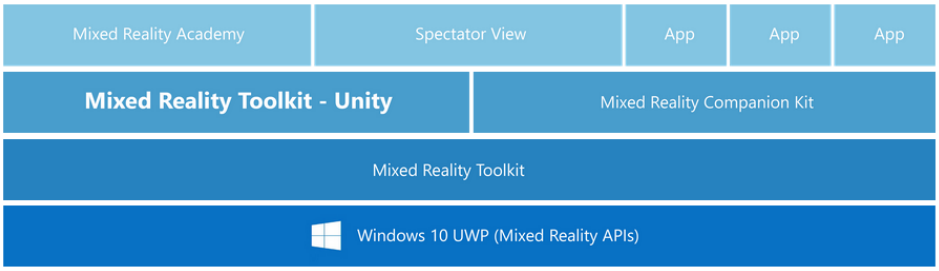


Abbildung 3: Komponenten des Mixed-Reality Toolkits [25]

hinzugefügt. Allerdings müssen so oftmals auch Anpassungen an bereits entwickelter Methodik in Kauf genommen werden.
Für die Entwicklung mit der *HoloLens* ist Windows 10 Voraussetzung. Ein Emulator ist zudem verwendbar, wenn eine Windows 10 Pro oder Education Version vorliegt. Zu Prüfen sind bei der Entwicklung mit dem MR-Toolkit immer die unterstützten Unity Versionen.

2.6 Software Architektur Visualisierung

Software-Entwicklung ist komplex und beschäftigt sich mit der Lösung schwieriger Aufgaben. Für die umfangreichen Problemstellungen existieren meist keine einfachen oder auf den ersten Blick verständlichen Umsetzungen. Software ist abstrakt und schwer zu begreifen. Es fehlen vergleichbare physische Artefakte. Wie sieht beispielsweise ein Betriebssystem oder ein Compiler aus? Code als solcher kann gelesen werden und existiert, wie dieser allerdings funktioniert oder mit anderen Komponenten interagiert muss verstanden werden. Das Verhalten des Programmes kann sich während der Laufzeit ändern und wird meist von mehr als einer Person erschaffen. Software altert und wird stetig weiterentwickelt. Neue Mitarbeiter schreiben das Projekt weiter, neues technisches Wissen wird angewendet oder Fehler treten auf und werden mit anderen Ansätzen gelöst [3, 6, 26, 27]. Daher entsprechen die ursprünglichen Design-Entwürfe einer Software selten bis nie dem tatsächlichen Software-Zustand.

Software Visualisierung verwendet Darstellungen, um Software sichtbarer und greifbarer zu gestalten. Es bezeichnet die grafische Repräsentation der Informationen, um diese für den Benutzer verständlicher darzustellen. Verschiedene Aspekte der Software können fokussiert werden, da es viele verschiedene Interessenten gibt: Entwickler, Tester, Projekt Manager und auch die Endnutzer können von guten Darstellungsformen profitieren [3, 5, 28]. So existieren Visualisierungen der einzelnen Programmier-Pattern, Algorithmen, Web Services, Code History, Datenbank Schemata und *Software Architektur Visualisierung* (SAV) [27, 29]. SAV beschäftigt sich mit der Darstellung der Struktur des Software Systems inklusive der ver-

knüpften Entitäten und der Metriken. Ziel ist es einen Überblick über den Entwicklungsprozess und die Architektur zu vermitteln. Die bestehenden Strukturen in der Architektur sollen erkannt und verstanden und auch überdacht/geprüft werden können. Mögliche spezifische Schwachstellen können ggf. aufgezeigt werden [5].
Die statische Software Visualisierung stellt Informationen dar, die ohne Programmausführung zugänglich sind. Diese sind jedoch für alle möglichen Ausführungen gültig. Hingegen zeigt die dynamische Visualisierung Ergebnisse, die eine Programmausführung erfordert und bei der die Ergebnisse nur für einen einzigen Programmablauf gültig sind. Zusätzlich existiert noch die Darstellung der Evaluation einer Software. Hierbei werden zumeist die statischen Aspekte gezeigt, welche sich in der Zeit verändern [30]. In dieser Arbeit wird sich vorerst nur auf die statische Visualisierungsform beschränkt.
Generell lassen sich fünf Hauptdimensionen für Software Visualisierung festlegen [31, 32]:

Task Warum wird die Visualisierung benötigt?

Zielgruppe Wer wird diese Visualisierung verwenden?

Target Welche Daten werden berücksichtigt?

Repräsentation Wie werden die Daten repräsentiert?

Medium Wo wird die Visualisierung dargestellt?

Der Fokus dieser Arbeit liegt auf der Software Architektur Visualisierung, wobei Teile der anderen Visualisierungen inkludiert sind. Daher wird im Folgenden der Ausdruck Software Visualisierung gleichbedeutend mit der Abkürzung SAV verwendet. Die Repräsentation und das Darstellungsmedium sind Kernelemente der Thesis. Die anderen Dimensionen werden nur angeschnitten.

Die Architektur einer Software wird aus den bestehenden Komponenten, Schnittstellennutzungen und deren Beziehungen untereinander aufgebaut. Eine Visualisierung dieser Struktur soll die wichtigen Eigenschaften darstellen. Folgende Fragen sollten mit einer guten SAV beantwortet werden können [2, 3]:

- Aus welchen Komponenten besteht die Software?
- Welche Beziehungen bestehen zwischen den Komponenten?
- Über welche Schnittstellen arbeiten die Komponenten zusammen?

Eine detailliertere Anforderungsliste für gute Software Visualisierung beinhaltet zudem laut Young et al.[33] folgende Punkte:

Einfache Navigation Es sollte darauf geachtet werden, dass die Navigation einfach und übersichtlich erfolgen kann, damit keine Orientierungslosigkeit eintritt. Landmarken oder Karten können zur Hilfe genommen werden.

Hoher Informationsgehalt Die Visualisierung sollte so viele Informationen wie möglich beinhalten, allerdings dabei den Nutzer nicht überfordern.

Einfache Darstellungsart der Visualisierung Gut strukturierte und einfach zu interpretierende Visualisierungen erzeugen eine intuitive Navigation und Lesbarkeit. Keine komplexen Gebilde, aber viele Informationen sollten untergebracht werden.

Mehrere Detaillevel Granulare Level, verschiedene Abstraktionsebenen und der Informationsgehalt sollten variabel und auf den jeweiligen Usecase des Nutzers abgestimmt sein.

Stabilität gegenüber geringen Änderungen Bei kleineren Code Änderungen sollte sich die Darstellung nicht viel verändern, um Konsistenz zu erzeugen.

Visuelle Metaphern Das Verwenden von sinnvollen Metaphern kann das Verständnis und die Lesbarkeit der Visualisierung verstärken. Expressiv und effektiv sollten die verwendeten Metaphern sein. Siehe hierzu auch Kapitel 2.6.1.

Anpassbares User Interface Das User Interface sollte flexibel und intuitiv gestaltet sein. Unnötige Funktionen oder Informationen sollten nicht integriert werden. Nach [5, 34, 35] benötigt ein gutes User Interface (UI) für SAV folgende Punkte:

- Navigation: Rotation, Verschiebung, Zoom und das Wechseln von Ansichten enthalten
- Filterung: das Reduzieren der angezeigten Daten ermöglichen
- Selektion: das Auswählen von Elementen, mit besonderer Interessen unterstützen
- Kodierung: Anpassung von grafischen Variablen ermöglichen

Integration von anderen Informationsquellen Eine Integration von Daten aus anderen Quellen ist sinnvoll. Beispielsweise den ursprünglichen Code mit der manuellen Dokumentation der Software verknüpfen und andere Ressourcen einzubetten können hilfreich sein.

Anpassbare Automatisierung Eine manuelle Visualisierung würde keinen großen Nutzen erzeugen, da sie bei jeder kleinen Änderung händisch angepasst werden müsste. Daher ist ein automatischer Prozess anzustreben.

Werden die Vielzahl der bereits entwickelten und veröffentlichten Arbeiten (allein Merino et. al [28] untersuchte 86 relevante wissenschaftliche Paper zur Visualisierung von Software) betrachtet, wird deutlich, dass es sich um keine leichte Aufgabe handelt. Die obige Anforderungsliste dient als anzustrebende optimale Lösung. Bisher konnte dies allerdings noch nicht umgesetzt werden.
In [36] geben 40 % von 11 Befragten (Wissenschaftler & Software Entwickler in verschiedenen Bereichen) an, dass eine Software Visualisierung notwendig für ihre Arbeit ist. 42 % halten eine Verbildlichung für wichtig, aber nicht für zwingend notwendig. Ansonsten ist bisher kaum erforscht, wie viele Entwickler tatsächlich eine Software Architektur Visualisierung nutzen würden oder für notwendig erachten. Es ist schwierig an genaue Daten zu kommen. Eine Bedarfsanalyse aus [37] zeigt, dass die Kluft der entwickelten Visualisierungen und derer, die sich gewünscht wird stark auseinander gehen. Zukünftige Arbeiten sollten sich mit der Ergründung dieser Sachverhalte beschäftigen.

Um zu evaluieren, wie gut eine Visualisierung für den jeweiligen Use Case ist [3], müssen die Zielgruppe und die Fragen, welche durch die Darstellung beantwortet werden sollen, formuliert werden. Zusätzlich ist zu prüfen, wie gut verwendete Metaphern interpretiert werden können und wie innovativ die Interaktion ist. Diese Aspekte werden in Kapitel 8 weiter berücksichtigt.

2.6.1 Verwendung von Metaphern

Viele der bereits existierenden Visualisierungen verwenden Metaphern für die Umsetzung. Einige setzen auf abstrakte Körper, wie Quader, Kugeln, Kegel [38] oder ähnliches, während andere Metaphern aus dem „echten“ Leben verwenden. Einige Wissenschaftler vertreten die Meinung, dass bekannte Konzepte aus der Realität dem menschlichen Gehirn helfen, die Visualisierung einfacher zu verstehen und auch zu behalten. Menschen können grafische Bilder besser verarbeiten als Zahlen [1]. Die Steigerung der Komplexität ist allerdings ein Nachteil der Verwendung von realen Metaphern [3].
Visualisierungen verfolgen bestimmte Zwecke und es existiert keine allgemeingültige Visualisierungs-Methode, weswegen ein Vergleich schwierig ist. Nach bisheriger Recherche existieren wenige Versuche, die ergründen, welche Art der Metaphern für das Verständnis von Software sinnvoller ist [1].
Das Verwenden der „real life“ Metaphern versucht, abstrakte Modelle auf bekannte Gegebenheiten abzubilden und zu verdeutlichen. In der Literatur [39, 40, 41, 42, 43] werden für Software Architektur Visualisierung vermehrt die Sonnensystem- (vgl. Kapitel 3) oder die City-Metapher (vgl. Abschnitt 2.6.2) verwendet.

	Level	Focus	Section	Visualization Technique	Representation	References	Year
Time T Visualization	Line	Line properties	2	Seesoft	2D colored pixel	[1], [2]	1992
				Sv3d	3D colored cuboid	[3], [4]	2003
	Class	Functioning, Metrics	3	Class BluePrint	2D layers and graph	[5], [6], [7]	1999
	Architecture	Organization	4.1	Treemap	2D/3D colored nested boxes	[8], [9], [10]	1991
				Circular Treemap	2D/3D colored nested circles	[8], [11]	1991
				City/Cities	3D city metaphor	[12], [13], [14], [15]	1993
				Sunburst	2D colored radial display	[16], [17], [18]	1998
				Solar System	3D solar system metaphor	[19], [20]	2003
				Voronoi Treemap	2D colored irregular shapes	[21]	2005
		Relationships	4.2	Dependency Structure Matrix	2D table	[22], [23], [24]	1981
				UML	2D diagrams	[25]	1996
				Geon	3D geon diagrams	[26], [27], [28]	1998
				Solar System	3D solar system metaphor	[19], [20]	2003
				Landscape	3D landscape metaphor	[29], [30]	2004
				Hierarchical Edge Bundles	2D graph with bundled edges	[31]	2006
				City/Cities	3D city metaphor with edges	[32], [33], [34]	2007
				3D Clustered Graph	3D clustered graph	[35]	2007
		Metrics	4.3	Polymetric views	2D graph	[5], [36], [37]	1999
				Solar System	3D Solar system metaphor with edges	[19], [20]	2003
				UML MetricView	2D UML diagrams with charts on top	[38]	2005
				Treemap metrics	2D nested boxes with color and texture	[39]	2005
				City	3D City metaphor	[40], [41], [42], [43]	2005
				UML Area Of Interest	2D diagrams with area of interest	[44], [45]	2006
Visualizing Evolution	Line	Changes	5.1	Code Flow	cable-and-plug wiring metaphor	[46], [47]	2007
	Class		5.2	TimeLine	3D building metaphor	[48]	2008
	Archi.	Organizational Changes	5.3.1	Hierarchical Edge Bundles	2D graph with bundled edges	[49]	2008
		Metrics Evolution		Evolution Matrix	2D matrix	[50], [51]	2001
			5.3.2	RelVis	2D Kiviat diagrams and graph	[52]	2005
				City/Cities	3D city metaphor with animation	[48], [53]	2008

Abbildung 4: Übersicht verschiedener Visualisierungsansätze aus [1], Stand 2011. In gelb sind die Ansätze zur City-Metaphe vertreten, grün hingegen sind weitere 3D spezifische Metaphern markiert.

2.6.2 City-Metapher

Im Jahr 1993 entwickelte Dieberger [44] die erste Visualisierung mit der sogenannten *City-Metapher*, allerdings nicht im Kontext der Software Architektur Visualisierung. Es handelt sich um ein Versuch, das Darstellen von Hypertext und das damit einhergehende Navigationsproblem innerhalb der Visualisierung zu lösen.
Die erste Software Architektur Visualisierung mit Anlehnung an die City-Metapher entstand 1999 [45, 46]. Visualisierungen, basierend auf dem menschlichen Verständnis von physikalischen Gegebenheiten, verbessern das Verständnis und die Navigation [1]. Daher liegt es nahe, vertraute Konzepte zu verwenden. Eine Unterteilung der Erde in Kontinente, Kontinente in Länder und diese wiederum in Städte ist eine allgemein bekannte Struktur. Städte können des Weiteren in Orte unterteilt werden, welche Straßen, Häuser und Gärten enthalten. Die verschiedenen Granulat-Level können auch für Software Architektur Visualisierung verwendet werden, da diese Struktur allgemein bekannt ist und keinerlei weiterer Erklärung bedarf. Beispielsweise könnte ein mögliches Mapping die Erde als komplettes Software Projekt sehen: Länder repräsentieren Packages, Städte Files, Orte Klassen und Häuser Methoden.
In der Literatur setzt sich diese Metapher als moderner Ansatz für 3D-Anwendungen durch. Abbildung 4 aus dem Jahre 2011 zeigt in gelb die Ansätze zur City-Metapher, welche sich für viele Anwendungsfälle der Software Visualisierungen eignen. In Grün sind weitere 3D spezifische Metaphern markiert.

3 Verwandte Arbeiten

Quellcode als solcher wurde in verschiedenen Arbeiten direkt in eine Visualisierung gemapped. Das bekanntestes Beispiel entstand 1992 unter dem Namen *SeeSoft*[47]. Eine Miniaturansicht der Code Zeilen wird dargestellt. Die Weiterentwicklung zu einer 3D-Darstellung *sv3D* [31] erfolgte 2003. Laut Recherche aus [1] war dies auch das letzte Jahr, in dem Visualisierung auf Source Code Ebene entwickelt wurde. Die anschließenden Techniken bezogen sich auf die Repräsentation höherer Abstraktionsebenen des Projektes, wie hierarchische Strukturen, Beziehungen einzelner Komponenten untereinander und Metrik-basierender Visualisierung, bei der die Qualität der Software dargestellt werden soll.
Einen guten Überblick über existierende Arbeiten im Bereich der Software Architektur Visualisierung (Stand 2011) geben Caserta und Zendra [1] sowie Merino et al. [28].
Panas et al. [48] entwickelten eine sehr realistische Darstellung einer Stadt mit vielen Details, wie Bäumen, Straßen und Straßenlaternen. Der Nutzer kann verschiedene Zoom Level auswählen und frei navigieren. Es wird davon ausgegangen, dass detailgetreue Darstellung nicht zum Verständnis vorausgesetzt wird. Kleine Abweichungen stören den Nutzer nicht bei der Interpretation. Es wird in [1] sogar davon ausgegangen, dass Vereinfachung und weniger Detailtreue zum schnelleren Verständnis der Metapher führen kann und dabei nicht von den wichtigen Informationen abgelenkt wird.
Als weitere *real life* Metapher existiert die Sonnensystem-Metapher. Dabei wird die Software als virtuelle Galaxie, bestehend aus vielen Sonnensystemen, dargestellt. Der zentrale Stern symbolisiert das gesamte Package, während Planeten in Orbits die Klassen innerhalb des Packages darstellen. Je weiter entfernt der Planet im Orbit dargestellt ist, desto tiefer ist er in der Baum-Hierarchie zu finden. Farben symbolisieren die Unterschiede zwischen Interfaces und Klassen. Ein interessantes Feature ist die Möglichkeit des Verschiebens einzelner Sterne. Dadurch ermöglichen sich weitere Ansichten für den Nutzer. Allerdings kann die Sonnensystem-Metapher weniger Perspektiven zur Darstellung der Organisation des Source Codes aufweisen. Eine Erweiterung auf Galaxie Ebene würde funktionieren, allerdings nicht als intuitiv verstanden werden [43, 42].
Die statische Struktur eines Softwareprojekts darzustellen ist einfacher, als alle Beziehungen der Komponenten verständlich abzubilden [1]. Die Anzahl an verschiedenen Beziehungen übersteigt die der statischen Elemente. Vornehmlich werden graph-basierte Visualisierungstools [49] verwendet, bei denen die Komponenten als Knotenpunkte und Beziehungen mittels Verbindungs-Kanten dargestellt werden. Allerdings können bei größeren Projekten schnell unübersichtliche und schlecht zu interpretierende Graphen entstehen, bei denen sich beispielsweise Kanten kreuzen oder Kom-

ponenten überlappen [5, 36]. Graphen in 3D können besser mit Verdeckung umgehen und lassen sich leichter navigieren. Allerdings ist die Navigation selbst schwierig und der Nutzer kann leicht die Orientierung verlieren, weswegen ein Zurücksetzen zur Ausgangssicht meist unumgänglich ist [30].
Dank der kommerziellen Zugänglichkeit der HMDs rücken diese Brillen in den Fokus der Wissenschaft [50]. Anwendungen in AR oder VR auf immersiven Headsets sind von 2D und 3D Visualisierungen zu unterscheiden. Relative Größen innerhalb der Visualisierung können besser mit AR/VR verstanden werden. Bei einer Bildschirmrepräsentation muss das komplette Modell verschoben werden, um die relativen Größen wahrzunehmen. Im Vergleich dazu kann der Nutzer sich in AR oder VR um das Modell bewegen [51]: In der virtuellen Welt herrscht oft ein Tradeoff zwischen dem Level-of-Detail (Auflösung und Genauigkeit der Darstellung), der Repräsentation und der Navigation. Auch die Möglichkeit der Zusammenarbeit (Collaboration verschiedener Nutzer) ist derzeit noch etwas eingeschränkt [51].
Software City-Metaphern werden daher auch schon in virtueller und überlagerter Realität untersucht.

3.1 Arbeiten in Virtual Reality

In [50] wird eine VR Anwendung mit Gestenerkennung (Oculus Rift[3] in Kombination mit Kinect[4]) vorgestellt. Eine City-Metapher basierte Software Architektur Visualisierung für Java wird vorgestellt. Abbildung 5 (Links) zeigt die Darstellung der Package Hierarchy in dieser Anwendung. Anschließend wurden insgesamt elf Testpersonen nach den entwickelten Gesten befragt. Der Fokus dieser Befragung liegt auf der Durchführbarkeit der entwickelten Gesten in 3D-Anwendungen. Die Gesten werden hier mit der Leap Motion erkannt und mit der linken oder rechten Hand des Nutzers ausgeführt. Die Gesten zur Translation, Rotation und Selektion schneiden in der Bewertung der Probanden gut ab, die Zoom Geste wird als nicht intuitiv wahrgenommen.

In [52] wird ein Software-Visualisierungssystem präsentiert, das objektorientierte Software in einer Virtual-Reality-Umgebung darstellt. Die Arbeit wird durch einige der jüngsten Fortschritte auf dem Gebiet der Informationsvisualisierung (2001) motiviert. Es wird eine eigens entwickelte Mapping-Sprache für Software vorgestellt. Klassen werden beispielsweise als Plattformen dargestellt und die Anzahl an Methoden bestimmt die Größe dieser. Eingefärbte Balken geben Auskunft über vorhandene Konstruktoren, Funktionen und Attribute. Die Anwendung wird in einer CAVE [53]

[3] www.oculus.com
[4] https://www.xbox.com/en-US/xbox-one/accessories/kinect

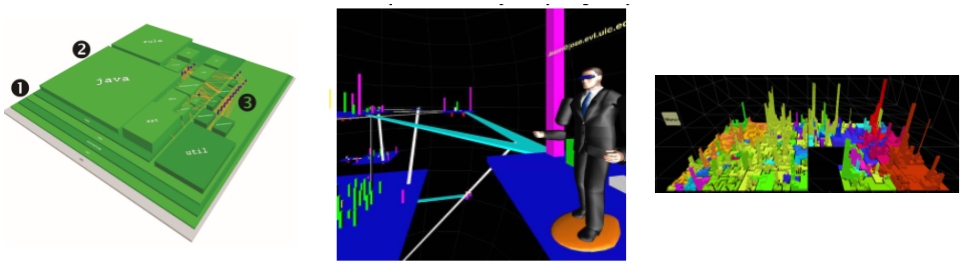


Abbildung 5: a) An die City-Metapher angelehnte Visualisierung einer Java Software Architektur Visualisierung in VR Umgebung [31].
b) UML Visualsisierung für VR Umgebung [51].
c) Code History Visualisierung mit farbiger City Kulisse für VR Anwendungen [30].

umgesetzt. Die gleichen Autoren publizieren eine weitere Arbeit [51], bei der es um die gleiche Umsetzung der UML-Diagramme geht. Das Ziel ist dabei nicht die Neuentwicklung einer Interaktion, sondern den sinnvollen Einsatz bisheriger Techniken in diesem Kontext zu finden. Basierend auf den Designprinzipien einer guten Navigation von [54] wird eine futuristische Darstellung gezeigt (Abgebildung 5b). Bei dieser Darstellung wird eine Teleporting Option des Standpunktes ermöglicht. Die Möglichkeit das System zu erkunden wird als positiv vermerkt und die Nutzer sind interessierter bei der Sache. Besonders hervorzuheben ist die mögliche eigene Gestaltung der Visualisierung. Der Nutzer kann die Darstellung anpassen und somit individuell gestalten. Laut [55] ist diese Anpassungsfähigkeit der Schlüssel zum besseren Verständnis. Der Nutzer kann sich besser orientieren und Gegebenheiten einprägen.
Eine Stadt mit Verbindungslinien zwischen den Gebäuden um Beziehungen darzustellen, findet in [56] Anwendung. Die 3D-Desktop Anwendungen dient der Darstellung von Trace eines Softwaresystems. Dafür wird eine Tages- und eine Nachtsicht der Stadt generiert. Verschiedene Perioden der Events werden zusammengefasst und durch Leuchten der einzelnen Elemente die Veränderung an den jeweiligen Gebäuden visualisiert.
In [30] wird eine auf der City-Metapher basierende VR-Anwendung vorgestellt. Dabei werden neue Layout Algorithmen für die Gebäude aufgezeigt. In dieser Visualisierung können Software Aspekte erforscht und zusätzlich auch reiner Quellcode betrachtet werden. Interaktion erfolgt über zwei Controller. In zukünftigen Arbeiten sollen stabilere Berechnungen zur Darstellung der Stadt Anwendung finden und eine Git-Hub[5] Erweiterung zur Identifizierung von schlechtem oder ineffizienten Code erfolgen. Die Anwendung soll diese Gebiete aufzeigen und markieren. Eine farbige Repräsentation der Software-Stadt ist in Abbildung 5c) zu sehen. Dadurch

[5] `https://github.com/`

können Systeme optimiert werden. Zusätzlich ist eine genauere Analyse mit eye-tracking fähigen HMDs geplant. Die Adaption dieser Anwendung für Augmented-Reality Brillen ist ein weiterer zukünftiger Schritt dieser Forschungsgruppe.
Mit *CityVR* [57] existiert ein interaktives Visualisierungstool, welches die Stadtmetapher mit Hilfe von Virtual Reality in einer immersiven 3D- Umgebung implementiert. Durch die spielerische Umsetzung soll das Interesse und die Begeisterung der Entwickler für Software-Verständnisaufgaben erhöht werden. Die Nutzer sind neugierig, aufgeregt, eingebunden und im positiven Sinne herausgefordert. Die Probanden zeigen sich gewillt länger mit der Anwendung und den Aufgaben zu beschäftigen.
Mit *IslandViz* [4] existiert eine VR-Visualisierung für modulbasierte Software (wie beispielsweise OSGi). Es bietet durch verschiedene Ansichten auf Detaillevel höheres Verständnispotential im Vergleich zu klassischen 3D-Visualisierungen. Es wird eine Weiterentwicklung der Stadt-Metapher verwendet: Die Inselmetapher. Die Softwaremodule werden als Inseln auf einer Wasseroberfläche visualisiert. Das Inselsystem wird im Rahmen eines virtuellen Tischs angezeigt. Der Benutzer kann die Software-Visualisierung auf mehreren Granularitäts-Ebenen mittels intuitiver Navigation erkunden. Die Inseln werden anhand ihrer Import- und Export-Häufigkeiten ausgerichtet. Dadurch werden wichtige Bundles direkt in der Mitte platziert und in den Fokus gelenkt.

3.2 Arbeiten in Augmented Reality

Im Vergleich zu VR Software Visualisierungen ist die Augmented Reality Welt weniger in der Software Architektur Visualisierung vertreten. Folgende Ansätze verwenden AR, um Software darzustellen:
In *SkyscrapAR* [58] wird die erste AR Software Visualisierung (bisweilen nur für Java), als Open Source Projekt, vorgestellt. Hierbei findet auch die City-Metapher Anwendung und es wird die Entwicklung der Software dargestellt. Technisch wird nur ein Computer, ein gedruckter Marker und eine preisgünstige Kamera für ein markerbasiertes Tracking verwendet. Aufgezeigt wird auch hier das Problem, dass der Computerbildschirm zweidimensional ist und die Visualisierung in 3D abgebildet wird. Eine ständige Manipulation der Darstellung ist notwendig. Über gängige Eingaben, wie beispielsweise der Computermaus sind nicht alle sechs Freiheitsgrade (drei auf der Translations-Achse und drei auf der Rotations-Achse) erreichbar bzw. anwendbar. Die Lösung dieses Problems mit Augmented Reality [9] wurde schon 1997 vorgeschlagen. In Abbildung 6a) ist der Marker mit überlagerter Software-City zu sehen. Wird der Marker rotiert oder verschoben, so passt sich die Darstellung an. Durch die Verwendung eines Markers gelingt eine bessere Manipulation der Sichtweise, jedoch werden zur Interaktion mit der präsentierten Stadt immer noch Maus und Bildschirm als

Eingabe verwendet. Hovert der Nutzer über die einzelnen Gebäude, werden Zusatzinformationen angezeigt. In zukünftiger Arbeit sollen diese Eingabemechanismen ersetzt werden und zudem die Anwendung für andere Programmiersprachen erweitert werden. Zusätzlich zum Informationsgehalt wird auch die Positionierung der Maus auf dem Modell erleichtert. Die Höhe der Gebäude wird durch die Anzahl der Änderungen im Code bestimmt. Dadurch fallen direkt die höchsten Gebäude in der Stadt auf, welche durch die Anzahl der Veränderungen am meisten Beachtung finden. Werden mehrere Hochhäuser in einer unmittelbaren Nachbarschaft entdeckt, so kann auch von einem City Center gesprochen werden und diese Klassen bilden den Kern der Anwendung.
VisAR [59] erforscht die Einsetzbarkeit der AR in statischen Datenvisualisierungen. Graphen, Diagramme und verwandte statische Daten profitieren von Interaktion. Als Anwendungsfall wird eine Präsentation genannt, bei welcher die Teilnehmer die Möglichkeit erhalten die gezeigten Daten zu filtern und mit ihnen zu interagieren. AR ist bereits ein beliebtes Mittel, um Daten interessanter zu vermitteln und auch für Entertainment oder in der Lehre einzusetzen. Die Motivation des Nutzers soll durch diese neue Technik verstärkt werden. Statische Daten werden bei *VisAR* als Hologramm mit der *HoloLens* erzeugt und mit Filtermechanismen und ToolTips belegt. Die Verknüpfung verschiedener Darstellungen (z.B. Balkendiagramm vs. Graph) führen laut [59] zu höherem Interesse des Dargestellten. Die Interaktion mit statischen Darstellungen wird als vielversprechend gesehen. Durch die zunehmende Verfügbarkeit der MR-Devices wird die Anwendung und das Konzept als vielversprechend angesehen. Trotzdem müsse noch weitere Grundlagenforschung betrieben werden, um den Mehrwert herauszufiltern.

Abbildung 6: a) Markerbasierte AR Software Visualisierung aus [31].
b) UML Visualsisierung für eine MR Umgebung [60], Verbindungen der Klassen (als Rechtecke dargestellt), werden mittels Verbindungsrohr visualisiert.
c) Multi-Display Darstellung eines Debugging Tools [61].

Die Interaktion und Darstellung dreidimensionaler UML-Diagramme wird in [60] untersucht. Es wird sich mit der Frage beschäftigt, wie sinnvoll eine Mixed Reality Anwendung für das Verständnis des Nutzers ist. Es wird ein Prototyp für die Microsoft *HoloLens* entwickelt. Dabei wird erforscht, ob die Relationen, die verschiedenen Perspektiven und die Manipulation mit den Entitäten intuitiv ist. Der Nutzer soll die Objekte bewegen, markieren und verändern können. In Abbildung 6b) ist eine Darstellung eines einfachen UML Diagrammes abgebildet. Rechtecke sind Klassen und Verbindungen, die in 2D mit Pfeilen dargestellt sind, werden nun im dreidimensionalen Raum über Verbindungsrohre visualisiert. Die Visualisierung verwendet keine Metapher, um die Architektur zu verdeutlichen. Es werden lediglich die Flächen zu dreidimensionalen Formen gemapped. Es wird eine einfache Darstellungsweise verwendet, um verschiedene Layer der Architektur zu zeigen. Die Entwicklung des User Interfaces wird als schwierig empfunden, da es keine vergleichbaren Arbeiten gibt. Es wird sich an das „Natural User Interface“ aus [62, 63] orientiert. Weitere Arbeiten sollen den Unterschied der Exploration in 2D und 3D untersuchen.

DebugAR [61] stellt einen Multi-Display Ansatz vor, bei welchem Bildschirm, Tablet und *HoloLens* zusammen verwendet werden. Thematisch handelt es sich um eine Repräsentierung des aktuellen Standes eines verteilten Software Systems. Der Nachrichtenfluss wird aufgrund der Verwendung von AR expliziter dargestellt. In Abbildung 6c) ist die überlagerte Sicht durch die *HoloLens* abgebildet. Der hybride Ansatz bietet ein leichtes Verständnis der Vorgänge eines verteilten Systems. Der klassische Computermonitor zeigt die Text-Log Ausgabe, vergleichbar mit herkömmlichen Debugging Tools. Der zweite Screen besteht aus einem Multi-Touch Display und zeigt eine 3D-Visualisierung des Systems über die Zeit. Nodes, Zeitstempel und Color-Codes werden verwendet, um den Nachrichtenfluss abzubilden. Die Kommunikation aller verwendeten Devices wurde durch eine Client-Server Architektur hergestellt. Bisher ist nur eine minimale Interaktion umgesetzt, weswegen weitere Interaktions-Möglichkeiten und Layout Ansichten in der Zukunft eingebaut werden sollen.

4 Mehrwert und Besonderheiten der *HoloLens*

Der derzeit (2018) stolze Preis für die *HoloLens* limitiert das Wachstum der Marktanteile dieses MR-Devices. Während die Industrie im mobilen und portablen Bereich durch erschwingliche Preise immer weiter wächst, sind AR und VR Devices noch nicht komplett massentauglich. Der Umschwung ist im VR Bereich bereits erkennbar, da immer billigere Produkte auf dem Markt erscheinen. Windows plant, alle zukünftigen Geräte mit Windows Holographic auszustatten [64]. Die Statistiken[6] und Investitionen der großen Technikvertriebe deuten auf ein Umdenken hin und es wird ein rapider Anstieg des Umsatzes in diesem Feld prognostiziert. Nicht nur in der Spiele-Industrie wird der Fokus auf AR/VR gelegt. Klassische Anwendungsfälle für AR sind in der Industrie und Medizin angesiedelt. Größere emotionale Kunden-Erfahrung wird in den sozialen Medien immer präsenter, wodurch die Kommunikationsrate und der Bekanntheitsgrad steigt. Dadurch werden auch Sparten, wie Tourismus, Kultur, der Bereich der Inneneinrichtung und der Börsenmarkt in naher Zukunft mehr auf neue Technologien setzen, um den *Hype* für sich zu nutzen [64].

Da HMDs derzeit im Trend liegen und der Kaufpreis mittlerweile für kleinere Firmen und teilweise auch für Privatpersonen immer erschwinglicher wird, existieren vermehrt Ansätze, diese Brillen in den Arbeitsalltag zu integrieren. Sie sind leicht und benötigen weniger Platz als VR-Anwendungen mit externen Tracking-Stationen, wie beispielsweise die HTC Vive[7]. Es ist kein extra Raum zur Darstellung notwendig. Der Nutzer kann jederzeit die Anwendung starten, welche sich der Umgebung anpasst. Mittels Spatial Mapping (siehe Abshcnitt 4.5) können Hindernisse erkannt werden und die Hologramme realistisch positioniert und manipuliert werden.
Dank AR kann ein Gefühl der Immersion geschaffen werden. Virtuelle Objekte und reale Gegenstände existieren zugleich. Darüber hinaus können Bewegungshinweise in Kombination mit Stereo-Sehen erheblich zur wahrgenommenen Größe und Struktur der Objekte beitragen [30]. Auch verbessert sich das Verständnis, wenn der Benutzer die Gegenstände verändern kann [65].

Visualisierung für Software Projekte im 2D-Raum wurde bereits aktiv erforscht. Viele Techniken zur Erzeugung von Diagrammen, Grafiken und Mapping-Informationen wurden ebenfalls ausgiebig studiert [37]. In 2D existieren viele graphbasierte Methoden, welche mit Skalierbarkeit, Layout und Mapping Problemen zu kämpfen haben [31]. Die Darstellung von Daten in 3D erleichtern für Nutzer die Verständlichkeit und unterstützen sie besser bei Merkprozessen [66]. Die Identifizierung von dreidimensio-

[6] https://www.statista.com/statistics/591181/global-augmented-virtual-reality-market-size/

[7] https://www.vive.com/eu/

nalen Objekten fällt Probanden leichter als in 2D [67, 68]. Obwohl die Frage, welche genauen Vorteile 3D-Darstellungen gegenüber 2D bieten, noch immer beantwortet werden muss, haben einige Experimente vielversprechende Ergebnisse gezeigt. Diese Erkenntnisse motivieren zur Weiterarbeit im 3D-Raum und zur Exploration neuer Darstellungsmedien [37]. Ein großer Nachteil an bisherigen 3D-Darstellungen ist das intrinsische 3D-Navigationssystem [36]. Es ist schwierig, einen 3D-Raum mit einem 2D-Eingabegerät, wie beispielsweise eine Computermaus, zu manipulieren. Dadurch können Nutzer mit einer höheren Wahrscheinlichkeit die Orientierung verlieren [50, 36]. Vielversprechend sind daher virtuelle und Augmented Reality Visualisierungen mit anderen Eingabemethoden, welche möglichst der natürlichen menschlichen Bewegung nahekommen [30].

In der Forschung und Industrie existieren bereits einige Anwendungen und konkrete Umsetzungen für AR HMDs. Jedoch existieren im Kontext der Software Architektur Visualisierung wenige Ansätze, die den Mehrwert herausarbeiten:

Dank der Erinnerungsfähigkeit der Position, der Hologramme oder Datenfenster ist es problemlos möglich das Gerät auszuschalten und beim erneuten Anschalten die Objekte an derselben Stelle vorzufinden. Dies ist für eine Konsistenz der Arbeitsumgebung sehr wichtig und auch für das Zeigen eines gewissen Zustandes.
Die eingebauten Kamera des Devices, ermöglicht das Teilen und Zeigen eines Zustandes. Eine Live Verfolgung auf einem separaten Bildschirm oder das Aufnehmen von Screenshots oder Videos können bereit gestellt werden. Durch mehrere virtuelle Bildschirme können verschiedene Aspekte begutachtet werden, ohne eine Ansicht zu überschreiben (vergleichbar mehrerer Tabs im Browser oder verschiedener geöffneter Anwendungen auf einem Bildschirm).
Zusätzlich spielt die Gesundheitskomponente eine Rolle: Die Arbeit wird auch im Stehen ermöglicht. Es findet mehr Bewegung im Arbeitsalltag statt. In [57] wird kritisiert, dass Entwickler stundenlang nur an ihrem Bildschirm sitzen. Es wird herausgefunden, dass neue Techniken mit Bewegungsmöglichkeiten die Begeisterung für die Bewältigung softwaretechnischer Aufgaben steigert [69]. Auch können die Daten-Fenster in einer angenehmen Höhe im Raum angebracht werden. Dadurch ist die Anwendung nicht an große Büros gebunden. Der Nutzer kann durch das Skalieren des Modells oder der Fenster im Sitzen arbeiten und seinen eigenen Workspace (sprich Schreibtisch + Freiflächen) optimal ausnutzen.
Eingebaute Tiefen-Sensoren im Gerät ermöglichen die Gestenerkennung. Allerdings ist diese Feld, in dem die Gestern erkannt werden können (noch) sehr klein (Veranschaulicht durch das eingezeichnete Rechteck in Abbildung 7 mittig). VR Anwendungen koppeln oftmals andere Sensoren oder Geräte, um die tatsächlichen Hände der Nutzer zur Eingabe verwenden zu können. Bei der *HoloLens* wird all dies mit einem einzigen HMD zur Verfü-

gung gestellt.
Die Erweiterung der dritten Dimension kann als Vorteil und auch als Nachteil angesehen werden. Durch fehlende Begrenzung in der Darstellung werden neue Funktionalitäten zugänglich, andere aber erschwert. Meist geht dies mit einem Verlust der einfachen Koordination einher [50]. Dadurch ist es notwendig das Zurücksetzen des Ausgangszustandes zu ermöglichen. Eine kleine 2D-Landkarte aus Sicht der Vogelperspektive zur Orientierung anzubieten, lässt sich leicht einbinden, um den Nutzer zu unterstützen.

Dank neuster Technologie verfügt die Microsoft *HoloLens* über drei verschiedene Eingabemechanismen: Gaze, Gestenerkennung und Spracheingabe (vgl. Abbildung 7). Wichtig ist bei der Gestenerkennung, dass die Geste im sichtbaren Feld der integrierten Tiefen-Kamera ausgeführt wird.

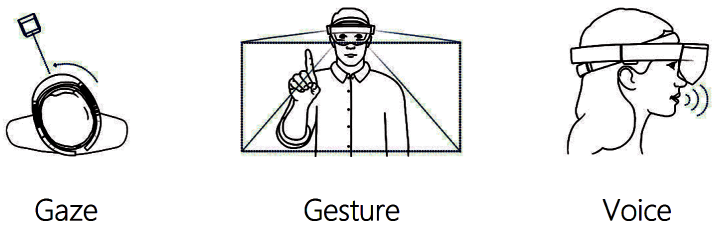


Abbildung 7: Mögliche Eingabemechanismen der Microsoft *HoloLens* [23].

4.1 Gaze Input

Der Blickpunkt, oder auch *gaze* zeigt an, wohin der Benutzer in der Welt schaut und lässt seine Absicht bestimmen. In der realen Welt werden normalerweise Objekte betrachtet, mit denen interagiert werden möchte. Das Gleiche Prinzip findet hier Anwendung. Mixed Reality Headsets verwenden die Position und Ausrichtung des Kopfes um dessen Blickvektor zu bestimmen (Abbildung 7 links). Dabei handelt es sich also nicht wirklich um den realen Blickpunkt der Nutzer. Der Vektor kann als ein Laser-Pointer interpretiert werden, welcher zwischen den Augen des Benutzers liegt. Während der Benutzer sich im Raum umsieht, kann die Anwendung diesen Strahl verfolgen, um zu bestimmen, welches virtuelle oder reale Objekt vom Benutzer betrachtet wird.

4.2 Gestenerkennung

Der Einsatz von Handgesten als Mittel zur Kommunikation und Steuerung der von Augmented-Reality-Systemen bereitgestellten Informationen bietet eine attraktive Alternative zu umständlichen Schnittstellengeräten

für die Mensch-Computer-Interaktion. Handgesten können dabei helfen, Leichtigkeit und Natürlichkeit zu erreichen [12].

Abbildung 8: Ausführung der *Air-Tap-Geste*, links die *ready-pose* und rechts die *Tap-Geste*

Dank der Werkzeuge im MR-Toolkit [25] kann wenn der Blick auf ein holografisches Objekt gerichtet ist, dieses Objekt durch eine Handbewegung des Benutzers beeinflusst werden. Um die Gesten-Steuerung zu aktivieren, muss sich die Hand des Anwenders innerhalb des Sichtfelds der Tiefen-Kamera befinden (Abbildung 7 mittig). Die *Air-Tap-Geste* ist die am häufigsten verwendete. Dabei befindet sich der Zeigefinder zunächst nach oben ausgestreckt (vgl. Abbilgung 8 links), welche *Ready Pose* genannt wird. Das Beugen des Zeigefingers zum Daumen (vgl. Abbilgung 8 rechts) und das anschließende Zurückkehren in die Ready-Pose führt einen sogenannten Air-Tap aus. Dieser Bewegungsablauf ist dem Mausklick nachempfunden. Er führt eine Auswahloperation durch, die das holographische Objekt dazu veranlasst, jede Aktion auszuführen, für die es programmiert wurde. Andere Gesten ermöglichen es dem Benutzer ein holografisches Objekt zu halten, es zu manipulieren oder zu bewegen [22]. Weitere gängige Gesten sind für das Scrollen einer holografischen 2D-Seite, das Drehen eines Hologramms oder das Zeichnen im Raum belegt [22]. Das Gestenvokabular ist ziemlich begrenzt, so dass für deren Verwendungen Designentscheidungen getroffen werden müssen. Mit der *Air-Tap-Geste* können bestimmte Objekte angewählt werden und durch das Halten des Zeigefingers in der Tap-Position und anschließendes Bewegen der Hand verschoben werden.

Speziell angepasste Gesten zu definieren und zu Erkennen ist von Microsoft nicht gewünscht und stellt somit einige Hürden dar. Es sollen lediglich einfache und nachvollziehbare Gesten verwendet werden, welche in jeder Applikation gleich belegt sein sollen. Das Positions-Tracking der Hände und Finger ist nicht ganz präzise, um eigene explizite Gesten verlässlich zu implementieren. Daher sollen in der Anwendung nur die vorinstallierten Gesten benutzt werden.

4.3 Spracheingabe

Cortana[8], ein virtueller Assistent zur Spracherkennung und Verarbeitung, ist automatisch auf der *HoloLens* installiert. Die vorgefertigten Befehle (wie das Starten und Enden einer Applikation, Lautstärke-Einstellungen oder Befehle zum Aktivitätsstatus des Devices) sind zu jeder Zeit verfügbar [19] (Abbildung 7 rechts).

Die Keywords für die Spracheingabe sollten eindeutig und verschieden sein. Am besten werden Begriffe verwendet, die in keiner anderen Anwendung mit einer anderer Funktionalität belegt sind. Falls doppelte Zuweisungen sinnvoll sind, so sollen diese einheitlich sein und nicht in unterschiedlichen Anwendungen andere Bedeutungen besitzen. Schlüsselworte, wie beispielsweise „Select“ sollten dementsprechend nur für das „Auswählen“ verwendet werden [22]. Gleichbedeutend zu diesem Schlüsselwort kann eine *Air-Tap-Geste* ausgeführt werden.

Nichtsdestotrotz wollen Nutzer ihre Augmented Reality Brille vielleicht nicht mit der Stimme aktivieren oder steuern. Hörbare Befehle sind zu öffentlich und ablenkend, weswegen sie als sozial schwierig betitelt werden. Ähnlich wie die Telefonie mit einem Headset. Unbekannte Dritte nehmen die sprechende Person als befremdlich wahr, da diese scheinbar mit Unsichtbaren oder sich selbst spricht [12]. Daher sollten alle möglichen Sprachbefehle auch über andere Eingabemechanismen erfolgen können.

4.4 Spatial Sound

Der folgende Abschnitt orientiert sich stark an der Definition und Erklärung aus [22]: In der realen Welt ist der Mensch in der Lage den Ursprung des Klangs zu orten. Der Klang, der aus dem Lautsprecher auf der linken Seite kommt, trifft zuerst das linke Ohr, bevor das rechte Ohr einen Ton wahrnimmt. Da das linke Ohr näher an der Tonquelle ist, ist der Ton dort lauter wahrzunehmen, als im rechten Ohr. Entsprechend trifft der Klang, der aus dem rechten Lautsprecher kommt, eher auf das rechtes Ohr und ist lauter als das, was das linke Ohr hört. Beim Stereo-Hören befindet sich der beste Platz zwischen den zwei Lautsprechern. Hingegen kann sich beim räumlichen Klang in einem Raum bewegt werden. Der Klang scheint immer von den Orten zu kommen, an welchen die Hologramme positioniert sind. Wird sich einer Schallquelle genähert, wird das Gehörte lauter. Da ein Hologramm nicht anfassbar ist, können Audio-Hinweise eingesetzt werden, um Information und Rückmeldungen an den Nutzer zu senden. Sinnvoll ist dies beispielsweise bei der Auswahl eines holografisches Objekts oder wenn eine Geste erkannt wurde.

Die Menschen sind daran gewöhnt, dass ihre Aufmerksamkeit vom Klang angezogen wird - wir blicken instinktiv auf ein Objekt, das wir um

[8] `https://www.microsoft.com/en-us/cortana`

uns herum hören. Wenn der Blick des Benutzers auf einen bestimmten Ort oder Objekt gelenkt werden soll, kann mit Sound gearbeitet werden [23]. Herkömmliche visuelle Reize, wie Pfeile können unterstützend verwendet werden. Das Anordnen eines Tons an diesem Ort ist eine sehr natürliche und schnelle Art, den Nutzer zum Objekt zu führen. Bei den meisten herkömmlichen interaktiven Erlebnissen werden Interaktion-Sounds wie UI-Klangeffekte in Standard-Mono oder Stereo wider gegeben. Da aber alles in der gemischten Realität im 3D-Raum existiert - einschließlich der Benutzeroberfläche - profitieren diese Objekte von räumlichen Klängen. Wenn wir einen Knopf in der realen Welt drücken, kommt der Ton, den wir hören, von diesem Knopf. Durch realistisches Positionieren der Soundeffekte kann eine natürlichere und realistischere Benutzererfahrung gegeben werden [15, 22].

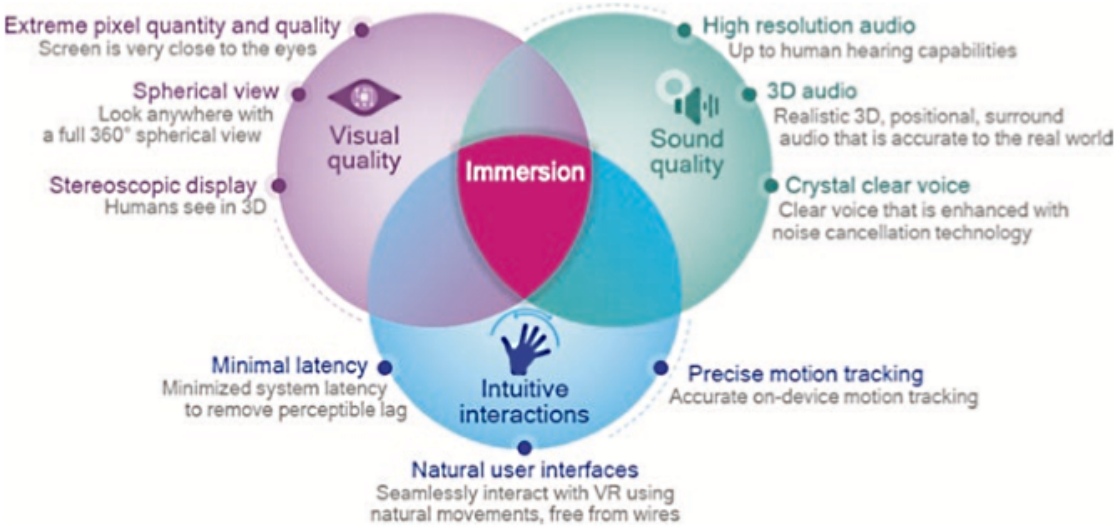
4.5 Spatial Mapping

Spatial Mapping [22] ist die Technologie, die es der *HoloLens* ermöglicht, holografische Objekte in den Kontext der realen Welt einzubetten. Mit einem Scan der Umgebung lernt die *HoloLens* die Position der vorhandenen Wände, des Bodens, der Decke, sowie aller Möbel oder anderen Gegenständen, die sich im Raum befinden. Für eine gute Illusion sind zwei Informationen unumgänglich: Zum einen die exakte Position des zu projizierenden Objekts und zum anderen die 3D-Position des Nutzers im selben 3D-Raum. Diese Informationen zusammen reichen aus, um eine exakte perspektivische Projektion für AR zu generieren [12]. Räumliches Mapping ist eine der Schlüsseltechnologien, die eine gute Illusion erzeugt, so dass holographische Objekte tatsächlich in der realen Welt zu interpretieren sind. Es kann gesteuert werden, wie detailliert ein Scan der Umgebung sein soll. Je detaillierter der Scan, desto genauer die Platzierung der Hologramme und desto überzeugender wird die Illusion. Je detaillierter der Scan, desto mehr Leistung und Zeit wird beansprucht. Ist ein Raum eingescannt, wird dieser abgespeichert und muss bei einem erneuten Betreten nicht neu abgetastet werden. Lediglich Objekte, welche sich verändert haben, werden aktualisiert [12, 22].
Mit den gesammelten Daten der Umgebung kann ein Hologramm auf Kollision reagieren. Überdeckungen und Überlagerungen können anhand dieser Daten berechnet werden. Werden Hologramme hinter einem realen Objekt platziert, so wird das Hologramm nicht sichtbar sein. Wird es vor diesem positioniert, wird es wahrnehmbar gezeigt. Für diesen Effekt müssen Hologramme wissen, wie weit sie vom tatsächlichen Objekte entfernt liegen und wo sich diese genau befinden [22].

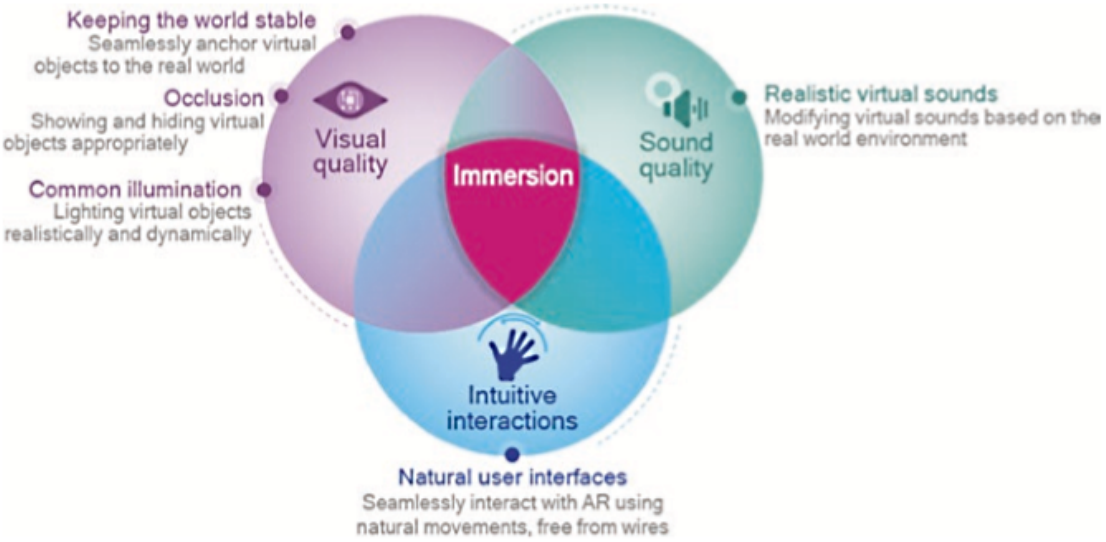
5 Immersive Anwendungen in Augmented Reality

Die Interaktion mit grafischen Benutzeroberflächen stellt in Augmented Reality Systemen einer der Hauptentwicklungsbereiche dar [12]. 2D-User Interfaces und grafische Oberflächen sind gut erforscht und dokumentiert. MR hingegen befindet sich am Anfang der Entwicklung, weswegen es noch keine ausreichende Forschung zu bewährten User Interfaces (UI) gibt [60]. Es existieren demzufolge auch keinerlei Standards, obwohl einige gängige Konzepte aus der mobilen Industrie bereits erfolgreich adaptiert werden (bspw. Swipen). Aufgrund der Einzigartigkeit der verschiedenen Anwendungen ist es unwahrscheinlich, dass ein einziges gemeinsames Konzept erstellt werden kann. Genauso unterscheiden sich grafische Benutzeroberflächen (GUIs) von PCs, Smartphones und Tablets von Betriebssystem, Plattform und Browser-Anbietern [12].
Natural User Interface Guidelines (NUI) [63] beschreiben im Allgemeinen Anforderungen an bedienbare User Interfaces. Es soll ein Interface spezifisch für das Darstellungsmedium entworfen werden. Es sollen keine bereits existierenden Mechanismen grundlos kopiert werden. Diese müssen begründet eingesetzt werden. Der Kontext der Darstellung soll beachtet und die Visualisierung immer dem Nutzen angepasst werden. Steuerungsmöglichkeiten sollten immer intuitiv sein und ein Feedback für den Benutzer bereit halten. Auch ist die Kombination aus physischen Geräten zur Eingabe denkbar. Konsistenz in der Benutzung ist sehr wichtig. Eine Geste darf beispielsweise nicht in einer Ansicht für andere Aktionen umfunktioniert werden [63]. Gute Interaktions-Möglichkeiten mit der gerenderten Repräsentation der Software führt zu einer brauchbaren Visualisierung. Fehlen sinnige Interaktionen, mindert es die Aussagekraft der Darstellung [3].

Ein gutes MR-Erlebnis basiert auf zwei Komponenten: Komfort und Immersion [18]. Flache Visualisierungen limitieren das menschliche Verstehen und das Erfahren der Objekte, die wir ergründen wollen [21]. In Abbildung 9a sind die Komponenten: Visuelle Qualität, guter Einsatz von Sound und intuitive Steuerung zu sehen, die in der Schnittmenge eine gute Immersion erzeugen. Stabile Welten, realistische Lichteinwirkungen auf Objekte und die Verdeckung sind genauso wichtig, wie eine gute Sound-Kulisse [12]. Für Augmented Reality Anwendungen gelten diese Richtlinien verschärft. Zu erkennen ist in Abbildung 9b, dass zu einer realistischen Sound-Kulisse auch hoch aufgelöste Audio-Dateien und klar verständliche Spracherkennung von großer Bedeutung sind. Auch werden immense Anforderungen an die Eingabe gestellt. Eine sehr geringe Latenz und hohe Genauigkeit der Gestenerkennung ist Pflicht. Zu niedrige Auflösungen des Dargestellten fallen direkt auf, da das Display meist sehr nah an den Augen platziert ist [12]. Die Anzeige- und Sensorverzögerung müssen kürzer als $10ms$ sein und die 3D-Welt muss mittels Tiefenmapping eingebunden



(a)



(b)

Abbildung 9: Grafische Darstellung der Komponenten, die für eine gute Immersion bei Anwendungen mitwirken [12]: (a) Elemente für eine gute immersive Anwendung im Allgemeinen (b) Besonderheiten für gute immersive Anwendung in Augmented Reality

werden [18].
MacKinlay [37] definierte zwei Kriterien, um die Zuordnung von Daten zu einer visuellen Metapher auszuwerten: Ausdruckskraft und Effektivität. Diese Kriterien wurden für 2D-Mappings entworfen, können aber auch für 3D-Mappings angewendet werden. Effektivität bezieht sich auf die Fähigkeit der Metapher, visuell alle Informationen darzustellen, die wir visualisieren wollen. Wenn beispielsweise die Anzahl der visuellen Parameter, die in der Metapher zur Anzeige von Informationen verfügbar sind, geringer ist als die Anzahl der Datenwerte, die wir visualisieren möchten, kann die Metapher das Ausdrucks Kriterium nicht erfüllen [37]. Das zweite Kriterium, die Effektivität, bezieht sich auf die Wirksamkeit der Metapher als ein Mittel der Darstellung der Informationen. Diese lässt sich weiter hinsichtlich ästhetischer Belange, wie auch der Optimierung, (z.B. die Anzahl der Polygone, die zum Rendern der Ansicht erforderlich sind) unterteilen [37]. Es sollten höchsten zwei bis drei verschiedene Formen verwendet werden, da das menschliche Gehirn mit zu vielen Formen Probleme während der Verarbeitung bekommt [31].

6 Konzeption

Werden die Anforderungen aus Kapitel 5 betrachtet, wird deutlich, dass die Konzeption einer AR-Anwendung eine große Herausforderungen darstellt. Gängige Konzepte können unter Umständen nicht wirkungsvoll sein und müssen speziell und spezifisch für das verwendete Gerät angepasst werden.
Zuerst wird die Zielgruppe bestimmt, anschließend die Metapher und die Darstellungsform der Software kritisch betrachtet und festgelegt. Die Begutachtung der möglichen Interaktion (vgl. Abschnitt 4) mit der *HoloLens* bieten die Grundlage für überlegte (und umsetzbare) Funktionalitäten der Anwendung. Anhand des theoretischen Konzepts soll später die Umsetzung (Kapitel 7) erfolgen und mögliche Limitationen (vgl. Abschnitt 7.1) aufgezeigt werden können. Tendenzen zu der Frage, wie sinnvoll AR im Bereich der Software Architektur Visualisierung ist, sollen gegeben werden können.

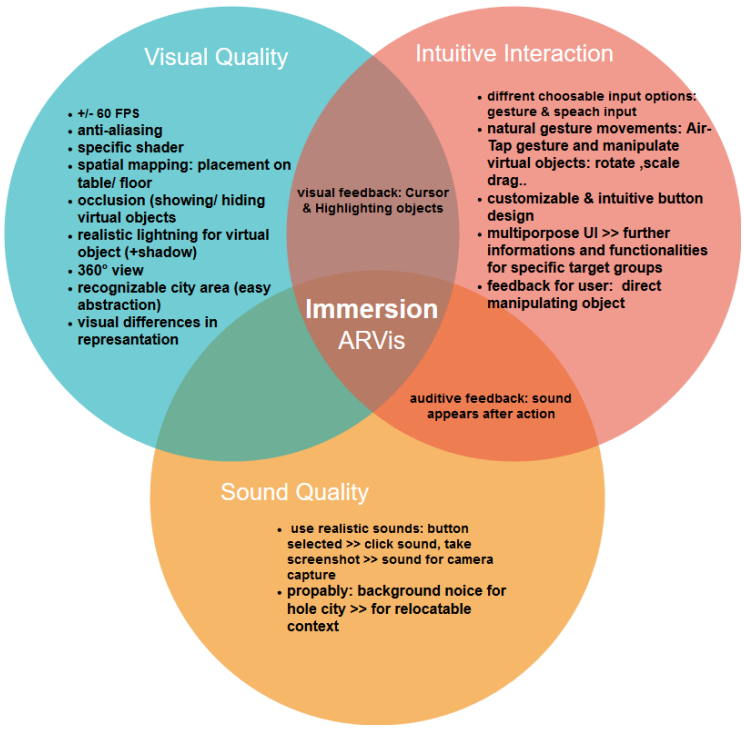


Abbildung 10: In Anlehnung an Abschnitt 5, welcher die Anforderungen an immersive Anwendungen auflistet, wird hier ein für *ARVis* angepasstes Venn Diagramm gezeigt.

6.1 Zielgruppe

In der Literatur zur Software Architektur Visualisierung wird die Definition der Zielgruppe nicht konkret geklärt. Oftmals wird lediglich der Benutzer oder User als Adressat genannt. In Figur 11 wird eine Kategorisierung aus 65 IEEE (Internationa Electrical and Electronics Engineers) SOFTVIS/ VISSOFT[9] Veröffentlichungen visualisiert. Daraus geht hervor, dass keine allgemeingültige spezifische Zielgruppe für Visualisierungen existiert. Für jeden Adressat bestehen besondere Bedürfnisse, wobei sich einige Anliegen überschneiden:

- **Software Entwickler & Ingenieure**
  Im Allgemeinen sind Software-Entwickler an einer Repräsentation der Software interessiert, um einen groben Gesamtüberblick zu erhalten. Das Einarbeiten in neue (große) Software Projekte kann erleichtert werden:
  - **Software Architekten** Software Entwürfe entsprechen sehr häufig nicht dem tatsächlichen Zustand der Software. Eine Visualisierung, um eventuelle Abweichungen im Software Design zu erkennen, wäre für diese Zielgruppe sinnvoll. Interessant sind hierbei auch Fehlkonstruktionen zu erkennen. Schleifen in Abhängigkeiten lassen sich gut visuell ergründen.
  - **Programmierer, Designer, Tester** Für diese Zielgruppe werden detailliertere Informationen benötigt. Quellcode, Dokumentationen oder detaillierte Abstraktions-Level sind wichtig.
- **Lehre (Studenten & Dozenten)** Zur Lehre und Erklärung von Software Komponenten ist nicht nur die Darstellungen der Software wichtig. Die Brücke zwischen Abstraktion und echtem Quellcode ist sinnvoll. Auch wird in diesem Kontext die Bedeutung einer guten Dokumentation deutlich. Wird die Vorstellungskraft und die Bedeutung der Verknüpfungen einzelner Komponenten deutlich, so können in Zukunft bessere Lehr-Ergebnisse erzielt werden.
- **Projekt Leiter/ Manager** Der Projektzustand sollte vermittelt werden können, auch ist es wichtig Fortschritte aufzuzeigen. Je nach Vorkenntnissen, ist diese Zielgruppe mehr oder weniger an technischen Informationen interessiert.
- **Kunden und Präsentation nach Außen** Diese Zielgruppe möchte auch den Projektstand erfassen. Im Gegensatz zu Software-Entwicklern zeigen Kunden insbesondere im Hinblick auf technische Fähigkeiten und Kenntnisse starke Unterschiede. Eine hübsche und einprägende

[9] `http://www.vissoft.info/` zuletzt eingesehen am 15.01.2018

Darstellung ist hierbei wichtiger, als der tatsächliche Nutzen dieser. Die Kombination aus interessanter Visualisierung und neuen Technologien wird als besonders wirksam gekennzeichnet. Quellcode als solcher wird dieser Zielgruppe wenig Mehrwert bieten. Die Komplexität einer Software allerdings mit vielen Verbindungen zu verdeutlichen erzeugt mehr Aufmerksamkeit als reine Zahlen oder Fakten.

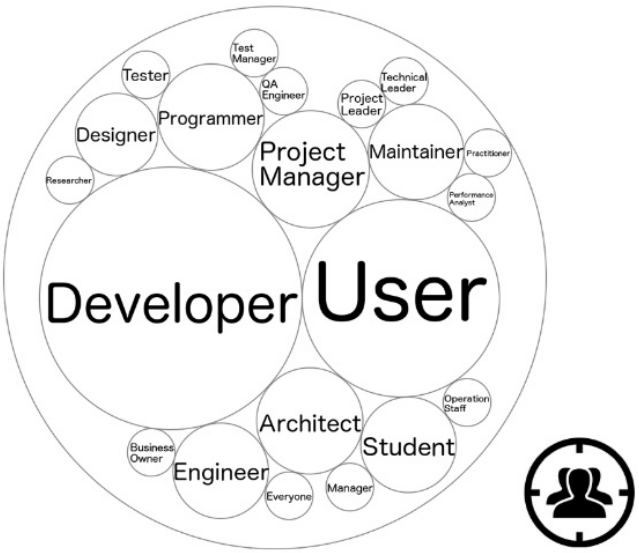


Abbildung 11: Zielgruppenkategorisierung anhand des IEEE SOftVis Analyse Papers [28].

Diese Arbeit dient als exemplarische Ideengebung der Visualisierung im AR Kontext. Daher ist eine genaue Zielgruppenbestimmung schwierig. Es sollen möglichst viele Anwendungsfälle abdeckt werden. Es werden Möglichkeiten aufgezeigt, die Anwendung auf die speziellen Anforderungen anzupassen. Aus Kapitel 2.6 geht hervor, dass eine gute Visualisierung den Nutzer nicht mit zu vielen Informationen überfordern soll. Daher ist die Abgrenzung der Bedürfnisse der jeweiligen Zielgruppe von größter Bedeutung. Derzeit erscheint die Anwendung zur Präsentation für Kunden/Projektleiter nach außen oder für neue Mitarbeiter am wahrscheinlichsten. Um den Rahmen der Arbeit nicht zu sprengen, wird sich allerdings eher auf die Bedürfnisse eines neuen Entwicklers/Mitarbeiters fokussiert, welcher sich in ein großes OSGi-Projekt einarbeiten soll. Daher wurden neue Mitarbeiter und duale Studenten am DLR befragt, wie sie sich in die Systematik eingearbeitet haben und welche Probleme aufgetreten sind. Die Befragten (insgesamt 6 Probanden) haben hauptsächlich Literatur und Source Code zur Einarbeitung verwendet. Unterstützung erhielten sie durch Kollegen, die Einstiegspunkte und kleine Beispiele zeigten. Probleme traten durch die Größe des Projektes und die Besonderhei-

ten OSGis auf. Die Verknüpfung der einzelnen Komponenten und deren Funktionen wurden unter anderem als schwierige Faktoren genannt. Eine Visualisierung wurde in nur einem Fall verwendet, welche allerdings nicht zum Verständnis beitrug und definitiv nicht wieder verwendet wird. Alle Teilnehmer wünschten sich eine Visualisierung oder konnten sich zumindest vorstellen eine zu verwenden.

Da es sehr schwierig ist, eine Statistik zur Nutzung von Software Architektur Visualisierung in Unternehmen zu erstellen, fehlen diese Zahlen zur Begründung der Notwendigkeit guter Software Architektur Visualisierung [70].

6.2 Darstellung der Komponenten

Ursprünglich sollte die Darstellungsvariante der VR-Anwendung *IslandViz* [4] als Metapher für diese Arbeit dienen. Das ganze Software System wird als Ozean mit Inseln repräsentiert. Jede Insel repräsentiert dabei ein OSGi Bundle und ist in mehrere Regionen unterteilt. Diese Regionen repräsentieren Java Packages und beinhalten mehrere Gebäude, welche die individuellen Klassen aufzeigen. Jede Insel verfügt über einen Import- und Export Hafen. Diese handhaben die Verbindungen zwischen den individuellen Bundles. Die Verbindungen sind auf verschiedenen Höhen darstellt, um zwischen Service Interface oder Service Komponente zu unterscheiden. Dadurch wird die visuelle Komplexität verringert und eröffnet dem Nutzer weitere Selektionsmöglichkeiten. Abschnitt 6.3 beschreibt die Problematik mit der Portierung der verwendeten Inseln. Da die Insel-Metapher aber sehr stark an der City-Metapher orientiert ist und eine Stadt schneller zur prototypischen Entwicklung verwendbar ist, wird als Alternative eine Stadt dargestellt. In zukünftiger Arbeit kann die Umsetzung der Inseln betrachtet und dieselbe Interaktion angewendet werden.

Die City-Metapher findet außerdem Anwendung, da sie sich in der zahlreichen Literatur bewährt hat [3, 30, 44, 57, 70]. Eine Metapher muss klar und einfach die Abstraktion der Inhalte vermitteln. Der Kern dieser Arbeit bezieht sich auf OSGi basierte Software Architektur Visualisierung. Dabei ist zu beachten, dass spezielle Funktionalitäten abzubilden sind. Vor allem das Modul Layer und die verschiedenen Services sind zu berücksichtigen. Die Methode muss stark genug sein, um folgende Komponenten abzubilden:

1. Klassen und deren Typen (Klassen, Interfaces, Enums)
2. Bundles (Module)
3. Packages
4. Import- und Exportrelationen zwischen Bundles

5. Service Komponenten und deren Beziehungen

6. Service Interfaces

Das Darstellen einer Software Architektur Visualisierung in einem feineren Detaillevel als der Klassenstruktur ist ein nettes Feature, welches oftmals wenig Mehrwert zur Analyse der Architektur beiträgt [71]. Die Metapher sollte den Fokus auf die modularen Layer legen, da sie die zentralen Aspekte abbilden. Basierend auf einer Landscape Metapher soll das City Konzept angewendet werden, welches sich gut zur Darstellung von OSGi-Projekten eignet. Jedes Element soll als 3D-Objekt auf einer 2D-Platte angelegt sein. Dieses Vorgehen verhindert einen Orientierungsverlust des Nutzers. Allerdings ist bei dieser Methode oftmals die Dimension der Höhe zur Darstellung weitestgehend unbeachtet. Dieser Platz kann optimal zur Darstellung der Services verwendet werden, ohne die Existenz der anderen Objekte zu beeinträchtigen.

Das ganze Software System soll als Miniatur-Stadt dargestellt werden. Jede Platte soll dabei ein OSGi Bundle repräsentieren und in mehrere farbige Regionen unterteilt sein. Diese Regionen sollen Java Packages verkörpern und beinhalten mehrere Gebäude, welche die individuellen Klassen aufzeigen sollen. Jede Region bietet genug Fläche, um die Gebäude gut positionieren zu können. Die Gesamtgröße der Packages wird anhand ihrer beinhalteten Komponenten angepasst, weswegen es zu keiner Überlappung kommen sollte. Jedes Bundle soll über jeweils ein Import- und Export „Paketzentrum“ verfügen. Werden andere Bundles verwendet oder stellt dieses Services bereit, werden die Paketzentren verwendet. Angedacht sind grüne für den Import und rote für den Export. Diese Zentren sollen die Verbindungen zwischen den individuellen Bundles handhaben. Vergleichbar mit der Post, kann sich ein Austausch der Bundles vorgestellt werden. Diese Verknüpfungen sollen auf verschiedenen Höhen dargestellt sein, um zwischen Service Interface oder Service Komponente unterscheiden zu können. Dadurch soll die visuelle Komplexität verringert und dem Nutzer weitere Selektionsmöglichkeiten ermöglicht werden. Das Einsetzen von Transparenzen kann Filterung verdeutlichen [72].

Die Metapher soll eine hierarchische Struktur mit drei Abstraktions-Level (Vorort, Region, Gebäude) ermöglichen. Die Navigation zwischen diesen Leveln erfolgt mit einem natürlichen Verständnis der Verknüpfungen. Dieses Vorgehen bietet Erweiterungen für weitere Detail-Level. Beispielsweise könnte eine Gruppierung von Städten als Land oder Kontinent interpretiert werden.

Die Positionierung des Hologramms auf einem Tisch bietet einen analytischen Blickwinkel auf die Daten. Vergleichbar ist diese Positionierung mit der Idee des strategischen Planungstisch für Generäle. Eine 2D-Sicht mit echten 3D-Figuren bietet einen guten Überblick. Jede andere Sicht kann natürlich angenommen und beliebig angepasst werden.

Eine Möglichkeit der dynamischen Darstellung wäre ein Auto- oder Fahrzeug-Verkehr (z.B. Postauto) zwischen den Paket-Zentren. Dies würde allerdings ein immenses Chaos hervorrufen, wenn nicht explizit ein Modul ausgewählt wäre.

Abgesehen von der Darstellung der eigentlichen Software Komponenten, müssen Zusatzinformationen benutzerfreundlich zugänglich gemacht werden. Konzipiert wird zunächst ein einfaches Hauptmenü zur Organisation des Kontextes. Anforderungen an Menüs in 2D unterscheiden sich sehr von denen an AR/VR Anwendungen. In [73] werden die Anforderungen für VR exakt aufgelistet. Einige Punkte lassen sich gut für augmentierte Applikationen übertragen: So ist die Erreichbarkeit der Menü-Elemente in 2D kontinuierlich gegeben. In Anwendungen mit neuen Techniken, muss dies nicht der Fall sein, da der Nutzer sich meist frei auf einer Freifläche bewegen kann. In bisher veröffentlichten Arbeiten finden sich oftmals in den 3D-Umgebungen 2D-Metaphern. Beispielsweise durch ein in der Hand gehaltenes Gerät, welches einen Bildschirm in 2D abbildet (vergleichbar mit einem Tablet). Dadurch können traditionelle Menüs in 3D verwendet werden und auf altbekannte Mittel zurück gegriffen werden [73]. Die Größe, Platzierung, Darstellungsweise (Text und/oder Icons) spielen eine große Rolle bei der Benutzbarkeit [73]. Zuerst sollte ein fixes Hauptmenü im Sichtfeld des Nutzers platziert werden. Die Microsoft Design Guidelines[10] für HoloLens Applikationen raten von dieser Anordnung ab. Daher soll als Alternative das Hauptmenü zu jeder Zeit mit einem Sprachbefehl aufgerufen werden können. Anschließend sollte dieses im mittleren Sichtfeld des Nutzers platziert werden. Das Menü folgt dem Gaze-Punkt, um immer im Sichtfeld des Nutzers zu bleiben, bis es explizit abgewählt wird. Dadurch erfolgt kein Verlust des auswählbaren Menüs im Raum. Sehr wichtig ist das visuelle Feedback für den Nutzer, um zu wissen, ob das Gewählte tatsächlich ausgewählt wurde. Schön sind, zusätzlich zu visuellen Reizen, auch akustische Bestätigungen [22, 73]. Für detaillierte Beschreibungen der Mächtigkeit des Sounds wird auf 4.4 verwiesen.

6.3 Portierung IslandViz

Ursprünglich sollte die Virtual Reality Visualisierung *IslandViz* [4] in Augmented Reality portiert werden. OSGi-basierte Software wird mittels einer Insel-Metapher dargestellt. In Abbildung 12 sind Screenshots aus der Anwendung zu sehen. Verwendet wurde die HTC Vive inklusive beider speziellen Eingabecontroller. In frühen Entwicklungsphasen dieser Anwendung befand sich der Inhalt schwebend im virtuellen Raum. Dies führte zu erhöhtem Aufkommen von Cyber Sickness, weswegen die Anwendung mittlerweile einen virtuellen Raum und einen großen Tisch in dessen Mitte

[10] `https://docs.microsoft.com/en-us/windows/mixed-reality/design`

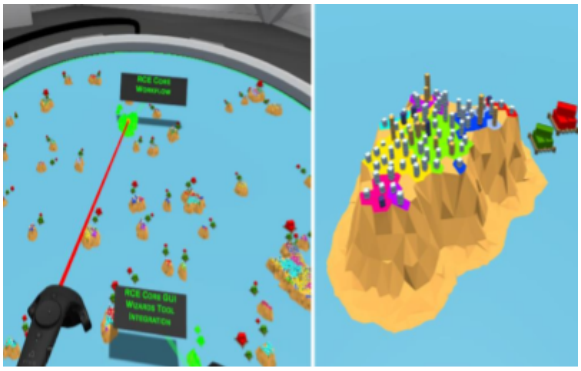


Abbildung 12: Abbildung der Komponenten aus *IslandViz* [4]

zeigt. Auf dem runden Tisch wird durch blaue Farbe das Meer abstrahiert angedeutet und die Inseln (also die Software Komponenten) darauf positioniert. Durch diese Ankerpunkte einer bekannten Umgebung wurde das Risiko der Cyber Sickness erheblich vermindert. Diese Table-Top Ansicht spricht eigentlich eher für eine Augmented Reality Anwendung, da meistens ein echter Tisch in Büros zugänglich ist. Daher entstand die Idee, die Anwendung für die Microsoft *HoloLens* zu entwickeln.

Die Entwicklung für Virtual Reality Anwendungen ist mit OpenVR[11] sehr gut unterstützt. Viele der verfügbaren Libraries sind (noch) nicht kompatibel für Universal Windows Plattform (UWP)[12] [74]. Daher ergeben sich sehr viele Probleme mit doppelten Verweisen für Dependencies und Metadaten. Auch wird Threading nicht im selben Maße unterstützt. Dementsprechend entstehen viele Fehler und es kann keine ausführbare Datei gebaut werden. Eine Möglichkeit wäre das Abkapseln dieser benötigten Libraries und diese einzeln extern zu bauen. Anschließend können sie wieder in das Projekt eingefügt werden. In Unity können mittels der „Do not process“-Funktion Libraries vom Bildungsprozess ausgeschlossen werden. Die Darstellung der Inseln und deren Positionierung mittels eines spezifischen Layout-Algorithmus aus IslandViz sollten adaptiert werden und mit der *HoloLens* in AR/MR übertragen werden. Die Erstellung der Inseln verwendet einige externe Libraries und da diese teilweise auch abgeändert wurden, war der Aufwand für den Nutzen der Darstellung nicht im zeitlichen Rahmen dieser Arbeit ausführbar. Jede einzelne Library hätte ersetzt werden müssen.
Die entwickelten Mechanismen lassen sich leicht auf andere Objekte übertragen. Eine Neu-Implementierung der Inselerstellung mittels explizit unterstützter Libraries ist sehr ratsam, um anschließend diese Darstellungsweise mit den Interaktionen in AR zu verknüpfen.

[11] `https://github.com/ValveSoftware/openvr`
[12] `https://docs.microsoft.com/en-us/windows/uwp/`

6.4 Interaktion

Ergebnisse einer Befragung aus [50] zeigen, dass Nutzer eine direkte Manipulation mit den eigenen Händen bevorzugen. Eine Interaktion des Objektes mittels einer Handbewegung wird als intuitiv wahrgenommen. Ein weiterer Ansatz, bei welchem der Nutzer eine Bewegung so lange ausführen und halten muss bis die gewünschte Aktion geschehen ist, wird als unbrauchbar bewertet. Die Testpersonen, welche diese Methode verwenden sollten, versuchten sie wie die direkte Manipulation anzuwenden. Die ausgeführten Gesten sollten sowohl für Rechtshänder als auch für Linkshänder ausführbar sein. Es sollten keine Unterschiede in der bevorzugten Hand existieren [37]. Die HoloLens ist als Computer konzipiert, nicht als Spielkonsole oder Fernseher. Microsoft entwickelt die Hard- und Software speziell, um die HoloLens zu einem funktionstüchtigen PC zu generieren. Physikalische Tastaturen können mit der HoloLens gekoppelt werden, oder der Benutzer kann sprechen, um Text einzugeben. Cortana[13] ist bereits in das Betriebssystem integriert und voll funktionsfähig [19].

Die Microsoft *HoloLens* bietet 3 Möglichkeiten der Eingabe: Gestenerkennung, Spracheingabe und Blickpunktberechnungen (vgl. Abschnitt 4). Die entwickelte Anwendung soll verschiedene Eingabemechanismen miteinander kombinieren. Der Prototyp soll bei den meisten Interaktionen zwei verschiedene Varianten (Sprachbefehle und Gesten-Steuerung) ermöglichen. Die Gaze-Verfolgung soll lediglich als Orientierungspunkt und visuelles Feedback der betrachteten Objekte dienen. Dadurch kann der Nutzer selbst entscheiden, welche Methode er verwenden möchte. Ein Sprachbefehl zur Erstellung eines Screenshots ist vermutlich schneller und angenehmer, als diese Option in einem Menü auszuwählen. Anschließend müsste zur gewünschten Position navigiert und dann der Screenshot aktiviert werden.

Dank der eingebauten Lautsprecher der *HoloLens* können Spatial Sound Effekte verwendet werden. *ARVis* soll diese Eigenschaft nutzen, um Feedback an den Nutzer zurückgeben zu können. Ein gutes Feedback könnte ein Foto-Geräusch sein, welches abgespielt wird, wenn ein Screenshot aufgenommen wird. Der Klick-Sound beim Auswählen eines Buttons soll aus der korrekten Richtung erscheinen und von der Software-Stadt könnten typische City-Geräusche (Straßenverkehr, Vögel, Menschenmassen etc.) erschallen. Dadurch würde die Metapher echter wirken. Stadt-Geräusche können das Orten der Stadt erleichtern, aber bieten vermutlich einen Reizüberflutung für den Nutzer.

6.5 Funktionalitäten

Alle Funktionalitäten sollen dazu dienen ein multifunktionales Umfeld für den Nutzer zu erstellen. Eine speziell anpassbare Darstellung soll möglich

[13] `https://www.microsoft.com/en-us/cortana`

sein. Je nach Zielgruppe können Informationen ein und ausgeblendet werden. Quellcode beispielsweise bringt für Nicht-Entwickler keinerlei Mehrwert, wohingegen ein Entwickler teilweise auch an spezifischen Methoden und genauen Inhalten interessiert sein kann. Über einfache Menüs können die Aktionen ein und ausgeschaltet werden. In Abbildung 13 ist ein früher Entwurf des User Interface gegeben. Der Nutzer sieht mittig platziert das auf die City-Metapher gemappte Software Projekt. Verschiedene Optionen geben Zusatzinformationen über die ausgewählte Komponente. Die Verwendung mehrerer Fenster (vergleichbar mit mehreren Bildschirmen oder offenen Tabs) dient als Inspiration. Das Erstellen eines individuellen Arbeitsplatzes soll ohne viel Aufwand möglich sein.

6.5.1 Information Panel

In [31] ist eine Weiterentwicklung der *SeeSoft* [47] Visualisierung zu einem Framework zu finden. Jede Code-Zeile wird als Zylinder repräsentiert. Die Farbe wird als Stilmittel für die verschachtelten Level verwendet. Über jedem Zylinder kann der Name des betroffenen Ordners angezeigt werden. Die Schrift wird immer zum Benutzer lesbar dargestellt. Die Namen können von dem Benutzer ein- oder ausgeblendet werden. In Abbildung 14 ist eine Repräsentation dieser Visualisierung zu sehen. Als weiteres Stilmittel werden verschiedene Transparenzen als Unterscheidungsmerkmal verwendet. Der Nutzer kann unterschiedliche Transparenz-Level einstellen und dadurch unwichtige Informationen herausfiltern. Dies umgeht zudem Überdeckungen. Allerdings sind die verschiedenen Opazitäten sehr gering und können in dieser Anwendung von den Nutzern nicht eindeutig zugewiesen werden [31].

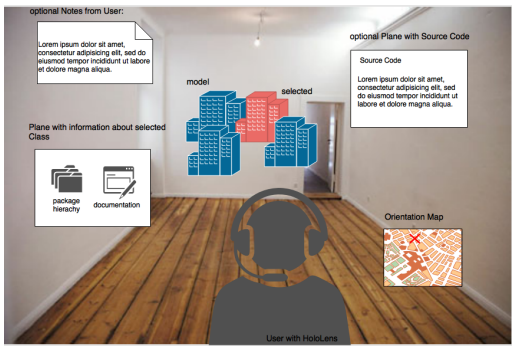


Abbildung 13: Erster User Interface Entwurf für *ARVis*: Das Software-Modell befindet sich in der Mitte mit hervorgehobenen angewählten Objekt. Verschiedene Fenster, vergleichbar mit Bildschirmen in der Umgebung sind in der Umgebung angebracht, welche Zusatzinformationen bereit stellen. Eine Orientierungs-Karte unten rechts gibt Auskunft über die Position des Nutzers innerhalb des Modells.

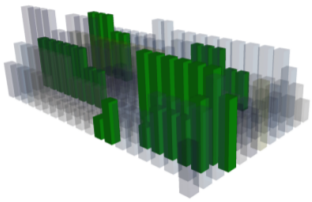


Abbildung 14: Mapping von einzelnen Code-Zeilen auf Zylinder, Farbe und Transparenz als Unterscheidungsmerkmale [31].

Das Information Panel wird konzipiert, um Zusatz-Informationen für den Nutzer bereitzustellen. Ähnlich wie in *SeeSoft* [47] soll die Hover Funktion Anwendung finden. Trifft der Gaze-Punkt auf ein Gebäude oder ein Paketzentrum, sollen diese optisch hervorgehoben und bei Gebäuden der Name angezeigt werden. Da der Name allein meist zu wenige Informationen bereit stellt, soll es möglich sein weitere Eigenschaften einzusehen. Wird ein Gebäude explizit ausgewählt (durch einmaliges Anklicken: Gaze-Fokussierung und Air-Tap-Geste) soll mittels optischen Feedback das Anwählen verdeutlicht werden. Geplant ist eine Bounding Box um das Objekt und ein relativ zum Sichtfeld eingeblendetes Information Panel. Die Bounding Box soll somit das ausgewählte Objekt markieren. Inkludiert sind standardmäßig der Name, der Package-Pfad im Projekt und der Typ der Komponente. Erste Entwürfe verwendeten ein Fenster, welches explizit an eine Position geschoben wurde. Frühe Feedbacks von Nutzern zeigten, dass das ständige Hin- und Herbewegen des Kopfes zum einen unangenehm ist und zum anderen die Übersicht des ausgewählten Objekts in Komposition nicht gewährleistet. Daher sollte das Panel relativ zur Bounding Box ausgerichtet werden. Auch wurden zu viele verschiedene Fenster nicht gut aufgenommen. Der Verlust der Orientierung, wo welche Information zu finden war, gilt als großer Nachteil. Daher soll in der Umsetzung ein multimodales Panel erstellt werden, dass alle benötigten Informationen bereit stellt und zudem im Sichtfeld des gewählten Objekts zu finden ist.

Aufgrund einer breiten Verteilung der Zielgruppen soll als Zusatzmenü *Developer Options* bereitgestellt werden. Über diese Einstellungen können sich die Dokumentation, oder aber auch der komplette Source Code des ausgewählten Gebäudes angezeigt werden. Zudem soll die Möglichkeit existieren sich Fakten anzeigen zu lassen: Die Anzahl der Zeilen Code, der Methoden, der verwendeten oder bereitgestellten Services. Da ein Nutzer, ohne Basiswissen der Informatik kaum etwas mit reinem Source Code anfangen kann, wird diese Information nicht zum besseren Verständnis beitragen. Wohingegen ein Entwickler durchaus auch an echtem Code hinter der Abstraktion interessiert sein wird. Durch die mögliche Anpassung der dargestellten Informationen kann der Nutzer selbst seinen Arbeitsplatz gestalten.

6.5.2 Import- und Export Verknüpfungen

Im- und Export-Verknüpfungen zwischen den Bundles sollen leicht verständlich dargestellt werden: Für Konsistenz innerhalb der verwendeten Metapher, können sich Post-Lager vorgestellt werden. Jeder Stadtteil (Bundle) sollte über zwei Lagerhallen verfügen. Diese Lagerhallen verbildlichen das System der Import- und Export Eigenschaften der Bundles. Jedes Bundle kann andere verwenden oder aber auch eigene Funktionalitäten zur Verfügung stellen. Visuell sollten beide Lagerstätten durch unterschiedliche Farben gekennzeichnet sein. Rote verwalten die Export-Schnittstellen und grüne beispielsweise die Imports.
Fixiert der User eine Lagerhalle, sollten die Verknüpfungen mittels eingefärbter (Rohre) dargestellt werden. Verliert sich der Fokus sollten sie wieder ausgeblendet werden. Wird eine Lagerhalle explizit ausgewählt (Fokussierung des Gaze + *Air-Tap* Geste oder mittels des Sprachbefehls „Select“) können Verbindungen konsistent eingeblendet bleiben. Ausgeschaltet könnten diese dann durch abermaliges Auswählen oder über den Befehl „Deselect Dependencies“.

6.5.3 Suchfunktion

Das Durchsuchen des Projekts nach einem expliziten Namen soll ermöglicht werden. Anschließend soll die eingeblendete Markierung den Nutzer zur richtigen Klasse hinführen. Das gesuchte Objekt soll in den Status *Aktiv* gesetzt werden und das Information Panel weitere Informationen anzeigen. Anhand des visuellen Feedbacks der Bounding Box, soll der Nutzer die Klasse orten können. Unterstützt wird er mit einer Orientierungs-Karte (falls diese vorher aktiviert wurde). Siehe hierzu Abschnitt 6.5.5. Dort wird die entsprechende Klasse markiert. Als weiteres Feauture soll der komplette Quellcode durchsucht werden können: Speziell für Entwickler kann diese Funktionalität sehr interessant sein: Wird nach einer Methode gesucht, kann nicht nur die Definition (bzw. die Klasse, die diese beinhaltet) gesucht werden, sondern auch alle Parteien, welche diese verwenden. Ähnlich wie in [30] können über ein Suchfeld explizit Klassen und deren Code angezeigt werden.

6.5.4 Annotationen hinzufügen

Um Gedanken oder Zusatzinformationen speziell für bestimmte Komponenten zugänglich zu machen, sollen Annotationen hinzugefügt werden können. Das Aufnehmen sollte sowohl über Tastatur Eingabe als auch Spracherkennung diktiert werden können. Die *HoloLens* stellt mit dem Mixed Reality Toolkit schon Grundfunktionalitäten für diese Aktionen bereit. Da es vielen Nutzern unangenehm sein könnte laut Texte zu diktieren sollen

andere herkömmliche Eingabemethoden Anwendung finden. Eine physikalische Bluetooth- Tastatur wäre eine denkbare Lösung für längere Texte oder eine komfortable Eingabe.

6.5.5 Orientierungskarte

In [50, 36] ist der Verlust der Orientierung als großer Nachteil der 3D-Visualisierungen mit immersiven Headsets genannt, daher ist es wichtig diesem gezielt entgegen zu wirken. Große Explorations-Flächen leiden oft am intrinsischem Navigationssystem und sind meist schwer zu navigieren. Laut [75] ist in den meisten Fällen jede Karte besser, als keine Karte. Die gilt besonders für sehr große Virtual Environments. Eine Karte kann den Nutzer gezielt unterstützen. Angelehnt an Straßenkarten wie Google Maps[14], soll eine Karte aus Sicht der Vogelperspektive auf die Stadt der Orientierung helfen. Auch das Zurücksetzen der Stadt zur Original Position kann bei einem Verlust der Orientierung hilfreich sein.

6.6 Ziele

Ziel ist es, dass ein Benutzer OSGi-basierte Software Architekturen schneller kognitiv erfassen kann. Dabei soll ein Überblick über das gesamte System vermittelt werden. Dadurch wird vor allem die Komplexität der Software direkt deutlich. Die Darstellungen von Abhängigkeiten sollen vereinfacht werden.

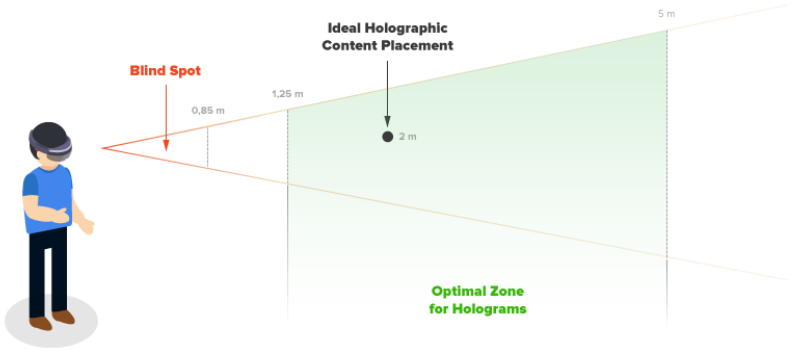


Abbildung 15: Optimaler Abstand für die Platzierung der Hologramme [76].

Die Anwendung soll einen explorativen Ansatz verkörpern. Auch soll die Interaktion interessant gestaltet sein, damit die Aufmerksamkeit des Nutzers gegeben ist. Ein gewisser Spaßfaktor ist bei diesem Vorhaben nicht außer Acht zu lassen [60].

[14] `https://www.google.com/maps`

VR ermöglicht die Gestaltung von allen denkbaren verschiedenen Räumen. Allerdings ist ein VR Ansatz mit benötigten Tracking-Stationen für einen Konzern beispielsweise nur bedingt sinnvoll. Die Portabilität und die Möglichkeit reale Objekte in die Darstellung einzubinden sprechen für MR. In jedem Büro kann die Applikation verwendet und auch echte Dokumente noch unterstützend hinzugezogen werden [60]. Daher soll ein interaktiver benutzerfreundlicher Prototyp entwickelt werden, der sich spezifische Vorteile von MR zu nutze macht. Eingabemechanismen wie Sprachkommandos und Gestenerkennung sollen Anwendung finden, um ein immersives Erlebnis zu gestalten [60]. Das Ziel ist es, ähnlich wie in [51], nicht die Neuentwicklung der Interaktion, sondern den sinnvollen Einsatz bisheriger Techniken in diesen Kontext zu finden.

Kurz zusammengefasst und auf Abschnitt 2.6 beziehend:

Task Komplexität großer OSGi-Projekte vereinfacht darstellen. Neue Mitarbeiter können so einen leichteren und spielerischen Überblick über das System erhalten. Die Exploration der Daten steht im Vordergrund.

Zielgruppe neue Entwickler/Mitarbeiter & Studenten

Target OSGi-spezifische Komponenten: Bundles, Services, Klassen & Dependencies

Representation Verwendung der City-Metapher

Medium Darstellung mit der Microsoft *HoloLens*

7 Umsetzung

Bevor eine Repräsentation der Visualisierung angezeigt werden kann, müssen die einzelnen Komponenten des Software Projekts extrahiert werden und einige Konstruktionsschritte durchgeführt werden. Da die Konstruktion sehr Performance intensiv sein kann, sollte diese vor der eigentlichen Darstellung erstellt werden. In Abbildung 17 ist der grobe Ablauf der angestrebten Informationsextraktion und der Erstellung der Visualisierung abgebildet. Da die Portierung der Insel-Darstellung in dem gesetzten Zeitrahmen nicht erfolgreich war, wird lediglich ein Prototyp implementiert und dargestellt. In geplanter künftiger Arbeit wird dieser Schritt des Einlesens und der Automatisierung angegangen und überarbeitet werden: Als Eingabe fungiert dann das Softwareprojekt, anschließend wird mit Hilfe eines von Tobias Marquardt [5] entwickelten Verfahrens ein JSON mit den extrahierten Informationen erstellt. Dabei handelt es sich um das Separieren der einzelnen Bestandteile, wie Bundles, Klassen oder Services. Diese JSON-Datei wird ausgelesen und die einzelnen Komponenten werden an die Metaphern angepasst und auf Objekte gemapped [4].

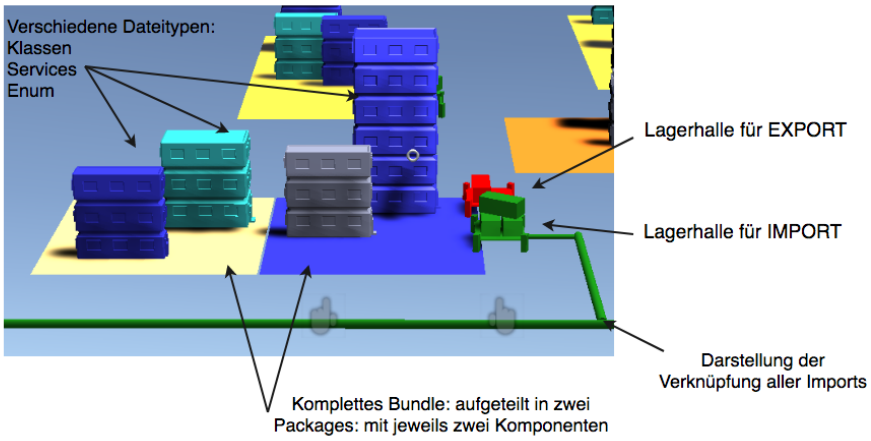


Abbildung 16: Darstellung eines Bundles mit zwei Packages (Screenshot: Unity Editor). Die unterschiedlichen Bodenplatten zeigen die Zugehörigkeit der darauf positionierten Dateitypen: Links befinden sich auf hellem Grund in Package 1 eine Klasse und ein Service-Interface. Das rechte Package (lila) enthält eine Klasse und ein Enum. Auf der rechten Seite des Bundles sind in grün und rot Lagerhallen platziert. Das grüne Rohr verdeutlicht die Imports dieses Bundles.

Die JSON Datei, beinhaltet folgende hierarchischen Informationen und internen Datenstrukturen :

- **Bundle:** Eine Liste mit allen vorkommenden OSGi Bundles

Abbildung 17: Informationsgewinnungsprozess der Anwendung

  - **Package:** Eine Liste mit allen Pakages, die im Bundle vorkommen.
    * **Complication Unit:** Eine Liste aller öffentlichen Klassen, enthalten in diesem Package. Eine Complication Unit korrespondiert zu einem einzelnen Java File, welcher eine Klasse kompiliert. Diese Units können entweder ein Interface, ein Enum oder eine Klasse sein.

- **Service:** Eine Liste mit allen vorkommenden Services.
  - **Service Component:** Jeder Service speichert eine Liste mit den bereitstellenden und verwendeten Service Komponenten.

- **Bidirectional Graph:** Ein gerichteter Graph zur Speicherung der Abhängigkeiten der individuellen Bundles.

Während der Erstellung der Objekte, wird auch die Funktionalität mitgegeben. Anschließend wird die komplette Software Architektur Visualisierung gerendert und das Projekt auf die *HoloLens* übertragen.
Der entwickelte Prototyp stellt bisweilen kein funktionsfähiges Projekt dar, sondern verbildlicht nur die mögliche Visualisierung/Umsetzung und deren Interaktionsmöglichkeiten.

Layout Algorithmen für die Positionierung der Gebäude und Regionen der City-Metapher sind spärlich erforscht. Hierbei existieren nur wenige Ansätze, welche sich mit einer intelligenten Anordnung beschäftigen. Das traditionell verschachtelte, das Straßen Layout und das Diagramm basierte Layout sind in der Literatur weit verbreitet [56, 77, 78]. All diese repräsentieren Klassen mit gleichen Formen und ignorieren bei der Positionierung die Beziehungen untereinander [30]. *IslandViz* stellt einen Layout Algorithmus vor, welcher die Bundles anhand der Anzahl Ihrer Verbindungen ausrichtet. Fehlende Evaluierungen bezüglich des Nutzens dieser Ausrichtungen bieten eine weitere spannende Frage für zukünftige Arbeiten im Bereich der Software Architektur Visualisierung.

Die erste Positionierung des Dargestellten wird in *ARVis* in einem Abstand von zwei Metern platziert. Laut [76] ist dies die optimale Entfernung für eine gute Interaktions-Möglichkeit. Der Nutzer kann diese Platzierung allerdings über die Menü-Einstellungen abändern. In Abbildung 15 ist der

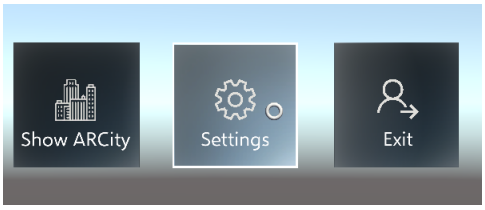


Abbildung 18: Hauptmenü *ARVis*: Der mittlere Button befindet sich im Blickpunkt, weswegen dieser visuell hervorgehoben wird. Zusätzlich verändert sich der vorherige runde Gaze-Cursor zu einem „Donout“. Mittels dieser beiden visuellen Feedbacks wird dem Nutzer signalisiert, dass es sich um ein anklickbares Objekt handelt.

blinde Bereich vor dem Nutzer und die optimale Darstellungsfläche visualisiert. Eine Framerate von circa 60 Bildern pro Sekunde wird als Richtwert für ein flüssiges User Erlebnis angestrebt. Eine zu geringe Framerate kann zu Cyber Sickness führen. Eine zu hoch gewählte Rate kann hingegen nicht schnell genug von der *HoloLens* berechnet werden. Dies liegt an der eigenständigen und daher limitierten HPU des Geräts [23].

Die Umsetzung des Prototys *ARVis* erfolgte unter Unity, Visual Studio und mit Hilfe des MR-Toolkits. Die verwendteten 3D-Objekte wurden entweder von [4] übernommen, oder mit Blender[15] erstellt. In Abbildung 16 sind die Grundkomponenten der Stadt abgebildet. *ARVis* ist in verschiedene Skripte oder Klassen aufgeteilt, die jeweils Unity-Objekten angehörig sind. Die meisten Skripte sind vom *Monobehavior* abgeleitet und besitzen daher eine Awake() oder Start() Funktion, sowie eine Update() Methode. Awake() wird direkt zur Initialisierung aufgerufen, falls das Skript aktiv ist, wird danach einmalig Start() ausgeführt. Update() wird solange das Skript aktiv ist für jeden Frame aufgerufen. Das HoloToolkit (vgl. Abschnitt 2.5) ist eine Sammlung an Skripten und Komponenten, die die Entwicklung für die *HoloLens* unterstützen. Die wichtigsten verwendeten Skripte dieser Anwendung sind für die Interaktion zuständig: wie der *InputManager*, *GazeManager* oder *VoiceManager*/*SpeachInputManager*. Der *GazeManager* ermittelt durch Raycasts Objekte, welche vom Benutzer anvisiert werden. Die Daten werden vom Zeiger (Cursor) benutzt, um ihn an entsprechender Stelle anzuzeigen und ggf. anschließend an den *InputManager* weiter zu leiten. Der *InputManager* überwacht alle Eingabequellen (z.B. verschiedene Gesten, Sprachbefehle, Controller) und sendet Eingabe-Events an das gerade fokussierte Objekt. Dieses kann dann entsprechend auf den Input reagieren und die angehängten Methoden und Skripte ausführen. Der *SpeachInputManager* verknüpft Sprachbefehle mit Events. Wenn ein Schlüsselwort erkannt wird, ruft er die entsprechende Methode auf, die ausgeführt werden soll.

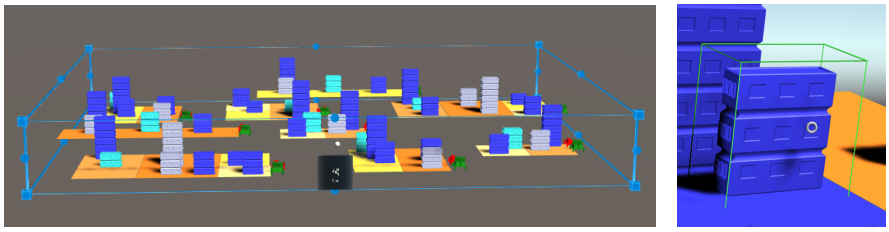
[15] `https://www.blender.org/`

Menü Alle implementierten Menüs verfügen über Holografische Buttons. Alle Buttons wurden als Kombination aus Schrift und Icon entworfen. Dadurch kann die Funktionalität leichter erkannt werden. Die Reduzierung der wichtigsten Einstellungen führt zu einer optisch einfach zu interpretierenden Oberfläche. Möchte der Nutzer zusätzliche Optionen auswählen, werden die jeweiligen Funktionalitäten gestaffelt zur Verfügung gestellt.
Zuerst wird ein Hauptmenü mit drei verschiedenen Buttons angezeigt, welches in Abbildung 18 zu finden ist. Dieses Menü ist mittels der Billboard-Funktion immer zum Nutzer ausgerichtet und so jederzeit lesbar. Zudem wird über das *TagAlong*-Skript das Menü immer in der Bildschirmmitte des Nutzers platziert, bis eine Funktionalität ausgewählt wurde. Ein Button ist für die Beendigung der Applikation zuständig, eine weiterer kann das Settings-Menü aufrufen und der dritte ist für das Laden der Software-Visualisierung zuständig.
Die Schriftgröße wurde an die spezifischen Anforderungen für *HoloLens* -Applikationen angepasst. Derzeit existieren keine offiziellen Guidelines, die sich bewährt haben. Auch die verwendeten Parameter in dem offiziellen *HoloLens* -Menü sind nicht öffentlich. Microsoft Mitarbeiter[16] weisen darauf hin, dass mit den Parametern so lange gespielt werden sollte, bis es dem eigenen Empfinden nach stimmig ist. Die Geschwindigkeiten des *TagAlong*-Skripts können derzeit nur durch Ausprobieren bestimmt werden.
Zusatz-Funktionen können zum einen über das Haupt-Menü und dem anschließenden Auswählen des Settings-Menü (vlg. Abbildung 23) eingeblendet werden oder aber durch das Menü innerhalb des Information Panels. Abschnitt 6.5.1 gibt detailliertere Informationen über das Information Panel.

Positionieren der Stadt Die komplette Software-Stadt kann beliebig positioniert und skaliert werden. Dank des *ManipulationManagers* wird das BoundinBoxRig-Skript aktiviert und ein AppBar-Prefab für die Stadt erstellt. Dieses ermöglicht die Skalierung der gesamten City (vgl. Abbildung 19a).

Import- & Export Verknüpfungen Jedes Bundle (Stadt-Region) verfügt über zwei Post-Lager. Diese Lagerhallen verbildlichen das System der Import- und Export Eigenschaften der Bundles. Fixiert der Nutzer ein Lager, so werden rohrförmige Verknüpfungen angezeigt. Verliert sich der Fokus werden diese ausgeblendet. Ein Lager kann explizit ausgewählt werden (Fokussierung des Gaze + *Air-Tap*-Geste oder mittels des Sprachbefehls „Select“) wodurch die Verbindungen konsistent angezeigt werden. Derzeit sind diese Verbindungen noch nicht au-

[16] `https://forums.hololens.com/`



(a) Adjust-Modus der Software City **(b)**

Abbildung 19: a) Wird über die App Bar der Adjust Modus ausgewählt, so wird die Software-Stadt von einer blauen Bounding Box umgeben. Die Formen an den Ecken (Kugeln und Quader) können ausgewählt werden und durch das Halten der *Air-Tap Geste* die Stadt skalieren oder rotieren. Ist dieser Modus aktiviert, kann die gesamte Stadt auch verschoben und dadurch neu positioniert werden.
b) Als visuelles Feedback wird ein grüner Wireframe um die angewählte Komponente gezeichnet.

tomatisch generiert. Ausgeschaltet werden diese durch abermaliges Auswählen oder über den Befehl „Deselect Dependencies“.

User Feedback Visuelles Feedback ist für den Nutzer ungemein wichtig. Rückmeldungen, ob anvisierte Objekte tatsächlich aktiv sind oder angeklickt wurden erleichtern das Verständnis der Applikation. Der Gaze-Punkt ist permanent in *ARVis* zu sehen, dadurch gewinnt der Nutzer einen Orientierungspunkt. Vergleichbar mit dem Computer-Maus-Zeiger ist eine Navigation möglich. Wird über ein anklickbares Objekt oder auch ein Button gehovert, wird dieser farblich hervorgehoben. Ein leichtes „Glühen“ der Objekte wird umgesetzt. Dabei wird das Material etwas heller dargestellt (vgl. Abbildung 18). Um ausgewählte Komponenten von anderen zu unterscheiden, wird eine Boundingbox erstellt, die sich an das Objekt anschmiegt. Ein grüner Wire-Frame markiert die Komponente (vgl. Abbildung 19b).

Zusätzlich zu visuellen Feedback sollten auch akustische Reize eingesetzt werden. Wurde ein Element ausgewählt ist dies durch ein „Auswahl-Geräusch“, wie beispielsweise ein Klick, eine gute Rückmeldung an den Nutzer. Das Erstellen eines Screenshots wird beispielsweise durch das Abspielen eines Photo-Capture-Sound untermalt. Der Sound wird zum einen über das MR-Toolkit [23] und von SoundCloud[17] zur Verfügung gestellt.

Information Panel Das Information Panel wird über das Anklicken eines Hauses aktiviert. Es enthält den Namen, Typ, Package-Pfad und falls vorhanden auch Annotationen über das ausgewählte Objekt. Wird

[17] `https://soundcloud.com/`

ein anderes Gebäude ausgewählt, verschiebt sich das Panel relativ zum Mittelpunkt des gewünschten Objekts. Aus diesem Grund und wegen der Billboard- Funktion ist das Panel immer zum Benutzer gerichtet und dadurch lesbar. Diese Positionierung kann zu ungewollten Verhalten führen. Überlagerungen mit anderen Gebäuden oder Verdeckung der Information durch andere Objekte können auftreten. In Abbildung 20 sind Kollision und Verdeckung zu sehen. Ein implementierter Shader, der das Panel immer auf einem Layer vor der eigentlichen Darstellung rendert, erzeugte andere unangenehme (Probleme mit Cursor, Buttons, Highlight-Möglichkeiten) Effekte. Wird eine Kollision mit einem Objekt erkannt, wir die Position um einen Offset nach vorne geschoben und erneut auf Kollision geprüft. Da dieses Verfahren aber mögliche Überdeckung nicht verhindert, wird das komplette Panel durch ein angehängtes Skript verschiebbar. Ein Auswählen und halten der Tap-Geste ermöglicht die Positionierung. Generell bleibt das Panel so lange eingeblendet, bis der Nutzer es explizit abwählt, ins Leere klickt, oder ein anderes Gebäude ausgewählt wird.
Innerhalb des Information Panels befindet sich ein Settings-Menü (vgl. Abbildung 23). Dieses ermöglicht die weitere Anzeige spezifischer Informationen. Wird diese Option ausgewählt, können Annotationen angefügt werden oder die spezifischen Developer Optionen angezeigt werden.

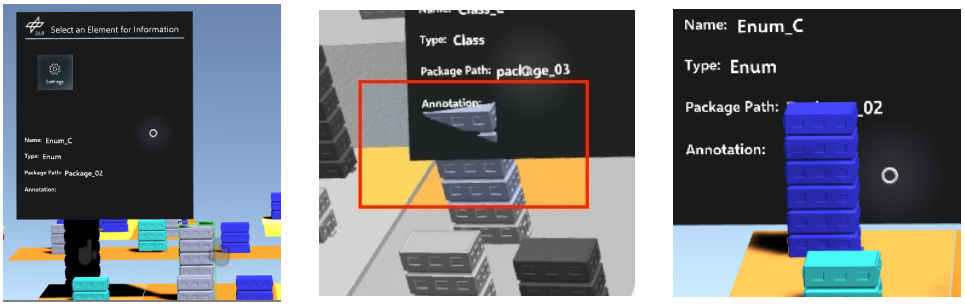
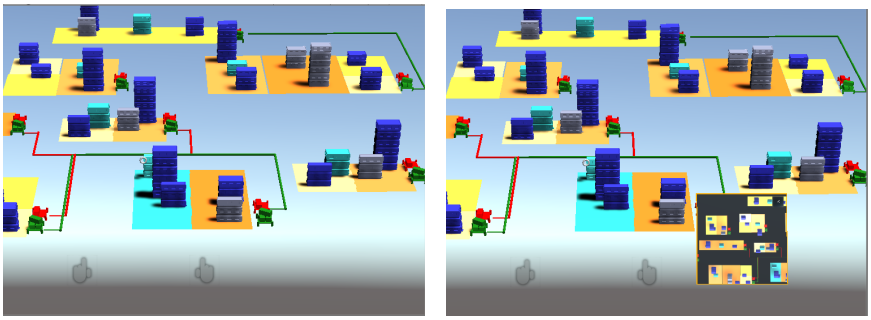


Abbildung 20: Abbildung des Information Panels (links) und ungewolltes Verhalten: (mitte) Kollision: Ein Gebäude schneidet das Information Panel. (rechts) Verdeckung: Ein Gebäude verdeckt Informationen.

Annotationen hinzufügen Wird eine Komponente ausgewählt, erscheint das Information Panel mit den Grundinformationen, wie Name, Package und Typ des Objektes. Ist bereits eine zusätzliche Information vorhanden, erweitert sich das Panel um diese Nachricht. Erstellt werden kann diese Information auf zwei Wegen: Ist das gewünschte Objekt ausgewählt erscheint im Information Panel ein Button, für *Annotatikeon hinzufügen*. Wird dieser ausgewählt, erscheint ein Textfeld. Ist



(a) **(b)**

Abbildung 21: Virtuelle Stadt mit und ohne Orientierungskarte:
a) Screenshot: Unity Editor der City ohne Orientierungs-Karte
b) Screenshot: Unity Editor mit Orientierungs-Karte im rechten unteren Bildabschnitt.

dieses aktiv (wird angeklickt), so wird eine virtuelle Tastatur eingeblendet (Abbildung 22) Anschließend kann der Text direkt über diese Tastatur eingegeben, oder das Mikrophon Symbol ausgewählt werden. Dieses aktiviert die Spracherkennung und den Diktierungs-Modus. Gesagtes wird aufgenommen und zu Text verarbeitet. Wird anschließend der Button „Save Annotation“ ausgewählt, wird die Information explizit für dieses Objekt gespeichert. Bis diese Notiz wieder gelöscht wird, erscheint sie als Zusatzinformation auf dem Information Panel. Beispielsweise können Auffälligkeiten, Kritik, Gedanken oder Hinweise zum Verständnis aufgenommen werden und für spätere Verwendungen zugänglich gemacht werden.

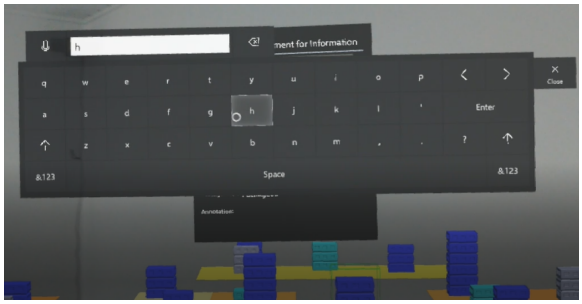


Abbildung 22: Die virtuelle Tastatur verfügt über zwei Eingabemechanismen: Jeder Buchstabe fungiert als einzelner Button. Zusätzlich kann über das Mikrofon-Icon (oben links) der Dirktierungs-Modus aktiviert werden und Sprache zu Text umgeformt werden.

Verwendet werden KeyboardInputFields und die vordefinierte virtuelle Tastatur. Die Grundfunktionalität wird schon über das MR-Toolkit [23] bereit gestellt. Allerdings existieren in der jetzigen Versi-

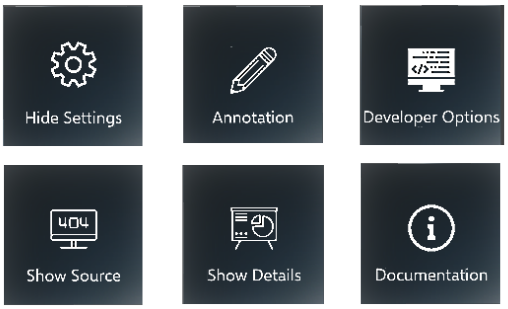


Abbildung 23: Abbildung des Zusatzmenüs im Information Panel. Erst wird nur der obere linke Button angezeigt. Wird dieser Aktiviert erscheint die komplette obere Reihe. Werden die Developer Optionen ausgewählt, erscheinen das komplette hier abgebildete Menü.

on einige Verhaltensmuster oder Bugs, die für die Anwendung behoben werden müssen. Die Tastatur besitzt kein korrektes Verhalten für Kollisionen mit Objekten. Auch müssen einige Erweiterungen vorgenommen werden, um sicherzustellen, dass die Tastatur nur dann eingeblendet wird, wenn es wirklich gewünscht ist. Der eingebende Text soll zudem nicht bei anderen ausgewählten Objekten im Textfeld erscheinen. Der Diktierungs-Assistent ist über ein Button auf der Tastatur aktivierbar. Er bleibt solange aktiv, bis er wieder abgeschaltet wird, oder die maximale Eingabelänge erreicht ist. Derzeit funktioniert nur die englische Eingabe, welche definitiv noch ausbaubar ist. Hier sind ganz klar Limitationen zu erkennen. Die Kombination einer physikalischen Bluetooth-Tastatur könnte eine Lösung sein.

Orientierungskarte Eine Karte zur Orientierung ist ein wichtiges Feature, da gerade AR-Anwendungen oftmals am Problem der Orientierungslosigkeit leiden. Eine Karte aus Sicht der Vogelperspektive, kann ähnlich wie bei der Benutzung von Landkarten den Nutzer bei der Navigation unterstützen. Verwendet wird eine zweite Kamera, die über der Stadt angebracht ist. Mittels orthografischer Projektion ist die typische Landkartensicht herstellbar (vgl. Abbildung 21b). Die Kamera für die Orientierungskarte wird mit dem Player (bei *HoloLens* - Anwendungen ist dies die Hauptkamera) verknüpft und folgt somit immer dem Nutzer.
Durch das Zurücksetzen der Stadt zur Urspungsposition ist selbst bei einem Orientierungsverlust ein Reset möglich. Diese Aktion kann über die Spracheingabe „Reset City“ erfolgen.

Screenshot Um den aktuellen Stand und Gegebenheiten für später festzuhalten, können Sceenshots aufgenommen werden. Der Sprachbefehle „Take Picture“ & „Screenshot“ nehmen mit Hilfe der integrier-

ten Frontkamera der *HoloLens* ein Screenshot auf. Es wird ein Sound abgespielt, um ein akustisches Feedback an den User zu senden. Explizit angepasst werden muss, dass auch die Hologramme mit auf dem Bild gespeichert werden, da ansonsten nur die realen Objekte im Sichtfeld auf dem Bild abgelichtet würden. Das Bild wird als *.jpg* gespeichert und kann später weiter verarbeitet werden.

Sprachbefehle In *ARVis* ist die Spracheingabe ein Instrument zur Steuerung. Verschiedene integrierte Sprachbefehle sind in der folgenden Tabelle abgebildet:

SELECT	Gleichbedeutend mit Air-Tap-Geste
SHOW CITY	Zeige ARVis-Stadt
RESET CITY	Zurücksetzen auf Anfangsposition
DESELECT DEPENDENCIES	Deaktiviere alle Dependencies aus
SCREENSHOT	Erstelle Screenshot
MAIN MENU	Zeige Main Menu
SETTINGS MENU	Zeige Settings Menu
MAP	Blende die Orientierungskarte ein

Die folgenden Funktionen in diesem Kapitel waren zur Implementierung geplant, wurden allerdings (noch) nicht umgesetzt:

Suchfunktion Das Suchen nach echten Klassennamen soll über ein einfaches User Interface erfolgen. Ähnlich zur Erstellung einer Annotation soll ein Suchfeld erstellt werden und anschließend nach der Eingabe (über Spracherkennung oder virtueller Tastatur) gesucht werden. Existiert ein Objekt mit diesem Namen, soll es *Aktiv* gesetzt werden. Dadurch erhält es das visuelle Feedback der Boundingbox und das Information Panel wird aufgerufen.

Services In OSGi wird zwischen Service Schnittstellen und Service Komponenten unterschieden. Während Service Interfaces direkt einer Java-Klasse oder einem Interface entsprechen, existieren Service Komponenten nur als OSGi spezifische Deklaration. Diese Deklaration muss jedoch eine Referenz auf eine Java-Klasse enthalten, die sie implementiert. Der Prototyp zeigt derzeit nur die einfachere Form der Service Schnittstellen. Erweitert werden muss die Darstellung noch für die Service-Komponenten. *IslandVis* [4] verwendet die Höhe als Darstellungsplattform mit unterschiedlichen Verbindungsknoten. Eine Erweiterung um diese Funktion ist in zukünftiger Arbeit mit hoher Priorität anzusiedeln.

Quellcode & Dokumentation Im Prototyp *ARVis* werden Informationen über das Information Panel angezeigt. Die Erweiterung um sichtbaren Quellcode und Dokumentationen der Klassen ist ein gutes Feature für Entwickler. Mittels eines kleinen Pfeils zur Navigation kann

auf dem Panel geblättert werden. Die Option der Auslagerung der Ansichten soll in Zukunft umgesetzt werden. Dadurch steigert sich die Komplexität der Darstellung, bei Bedarf können jedoch alle Informationen gleichzeitig erfasst werden.

Das MR-Toolkit bietet einige vorgefertigte Skripte und Funktionalitäten. Aufgrund der Aktualität und der permanenten Weiterentwicklung werden Skripte stetig verbessert und verändert. Dies führt zu neuen nützlichen Features. Allerdings werden dadurch vorherige Implementierungen teilweise hinfällig und ein komplettes Umdenken/Umschreiben notwendig. Einige Skripte benötigen nach der Aktualisierung andere Abhängigkeiten und unterschiedliche oder neue Parameter. Während der Entwicklung von *ARVis* wurden zuerst eigene Skripts zur Skalierung, Rotation und Positionierung geschrieben. Probleme entstanden dabei durch ruckelnde Anpassungen der Objekte. Dank des *ManipulationManagers* können diese Features mittlerweile leichter für Objekte verwendet werden. Andere Funktionalitäten haben an dieser Stelle nicht mehr mit der neuen Toolkit Version funktioniert und mussten erneuert werden.

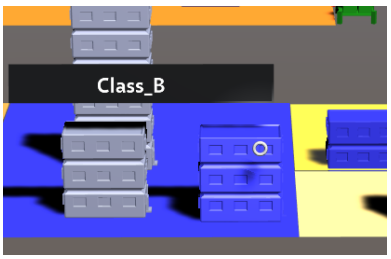


Abbildung 24: Wird eine Komponente anvisiert, erscheint ein 3D-Schriftzug als Tooltip.

Zusätzlich sind einige Probleme im Unity Editor aufgetreten, welche teilweise anschließend nicht reproduzierbar waren. Die entwickelten ToolTips (vgl. Abbildung 24) der Komponenten wurden sowohl im Editor, als auch im *HoloLens* Emulator gerade ausgerichtet. Nach der Portierung des Projekts auf das eigentliche Endgerät waren diese willkürlich rotiert. (Obwohl die gesetzten Positionen und Parameter der Rotationsachsen korrekt gesetzt wurden. Die Koordinaten wurden mittels einer Debug-Funktion der *HoloLens* überprüft.) Generell ist der Emulator eine gute Möglichkeit auch ohne Gerät zu programmieren. Allerdings existieren noch einige Bugs, die unvorhersehbare Hürden verursachen. Selbst wenn im Unity Editor und auf dem Emulator alles planmäßig funktioniert, können Unstimmigkeiten auf dem Gerät auftauchen. Der Emulator ist nur unter Windows 10 (Pro oder Education) verfügbar, was die Entwicklung weiterhin einschränkt. Die Dokumentation des Toolkits liegt meistens hinter dem aktuellsten Stand der Skripte, weswegen teilweise wichtige Informationen über Parameter oder Verknüpfungen fehlen und ein immenser Zeitaufwand benötigt wird, um die richtigen Einstellungen zu finden.
Erste Tests der Applikation erreichten gerade einmal 36 Frames pro Sekun-

de (FPS). Laut Microsoft [23] ist eine Framerate von ca. 60 FPS anzustreben, weswegen die Quality-Settings des gesamten Projekts überarbeitet wurden und letztendlich schafft es die Anwendung auf etwa 55 Frames. Da derzeit nur wenige Objekte angezeigt werden, müssen weitere Beschleunigungen in der Berechnung erfolgen, um ein flüssiges User Erlebnis garantieren zu können: Aufgrund des Aliasing-Effekts besitzen Kanten keinen geradlinigen Verlauf. Mittels Antialiasing, einer Technik die Tiefpassfilterung anwendet, können unschöne Kanten verbessert werden. Über den Parameter der *Eye Texture Resolution Scale* kann der Faktor angepasst werden. Laut [23] soll dieser Parameter zwischen $1,4$ und 2 liegen. Allerdings wird eine stabile Repräsentation mit einem Scale von $1,0$ erzielt. Das viermalige Super-Sampling der Anwendung sorgt für eine höhere Framerate.
Bei der Microsoft *HoloLens* handelt es sich um eine Entwicklerversion, weswegen einige Funktionalitäten noch weiterentwickelt werden müssen. Die Erweiterung des Toolkits ist als sehr schnelllebig und positiv zu vermerken. Jedoch wird dadurch auch die eigentliche Entwicklung neuer Anwendungen erschwert.

7.1 Limitationen

Die Microsoft *HoloLens* ist ein neues Gerät in der ersten Generation. Dadurch ergeben sich einige Limitationen: Die HPU der *HoloLens* ist mit ihren 2GB RAM nicht so leistungsfähig [23]. Mehr als 60 Frames pro Sekunde können nicht umgesetzt werden, weswegen zu viele Polygone, verschiedene Materialien und auffällige Shader nicht schnell berechnet werden können. Dadurch kommt es unter anderem zu einem Flackern der Anwendung und hohen Latenzen während Interaktionen.
Das Sichtfeld der *HoloLens* ist sehr klein, weswegen Anwendungen stark durch dieses eingeschränkt sind. Optimal wäre auch für *ARVis* ein größeres Sichtfeld, um die komplette Stadt auch in einer größeren Skalierung betrachten zu können. Auch ist die Region, welche die Gesten erkennt, noch relativ klein. Wünschenswert wäre ein größerer Bereich, um den Arm nicht immer so weit oben zu halten. Die Erweiterung mit dem sogenannten *Clicker* ist eine Möglichkeit dies zu umgehen, allerdings profitiert die *HoloLens* eher von der eigentlichen Gesten-Eingabe.
Als weitere Limitierung ist die Akkulaufzeit [23] zu nennen: Diese liegt zwischen 2-3 Stunden bei aktiver Nutzung. Als Nebeneffekt können bei stark beanspruchten Geräten kleine visuelle Störungen auftreten.
Die Spracheingabe funktioniert bei kurzen Befehlen ziemlich gut, allerdings ist das Diktieren von Text mit einem Glücksspiel vergleichbar. Nur die englische Sprache wird einigermaßen korrekt erkannt. Die aktive Forschung für Sprachsteuerung wird in Zukunft hoffentlich bessere Ergebnisse vorweisen. Die Nutzung von Sprache zur Eingabe ist ungewohnt. Viele Nutzer möchten keine lauten Befehle als Eingabe verwenden [12]. Das tragbare HMD ist nach längerer Benutzung schwer und unangenehm. Vor allem für Brillenträger ist es nicht optimal.

Der vorgestellte Prototyp könnte auch in VR umgesetzt werden. Lediglich das Anschmiegen der Stadt an glatte Oberflächen (z.B. eines Tisches) nutzt die Überlagerung und Verschmelzung realer und virtueller Objekte. Dank der Flexibilität der Größe kann auch ein kleiner Arbeitsbereich auf einem Schreibtisch als Darstellungsplatz fungieren. Eine Kombination aus physikalischen Geräten und der Anwendung ist geplant. Eingaben sollen über eine Bluetooth-Tastatur erfolgen können.

8 User Study

Obwohl beeindruckende Bilder zur Visualisierung erzeugt werden können, muss jedes Design für die realen Aufgaben getestet werden. In der Literatur existieren viele Ansätze der Software Architektur Visualisierung, allerdings sind diese schwer zu evaluieren [28, 79, 80]. Es muss festgestellt werden, wie nützlich die Darstellung für das menschliche Verständnis ist [81]. Jede Nutzerstudie sollte den Fokus sowohl auf die Bewertung der visuellen Darstellung, als auch auf Interaktionstechniken legen, da die Exploration mehr Informationen bereitstellen kann, als aneinandergereihte Bilder [82, 83, 84]. In den letzten Jahren ist das Interesse an Evaluierungen und Usability-Tests immens gestiegen. Das Hauptaugenmerk liegt auf der Human-Computer-Interaktion [82, 85]. Die meisten Methoden zur Evaluierung wurden für grafische Benutzeroberflächen entwickelt und für die Informationsvisualisierung angepasst [36]. Um einen Test für Usability durchführen zu können, sollte eine kurze Definition gegeben werden [86]: Wenn ein Produkt oder eine Dienstleistung wirklich brauchbar ist, kann der Benutzer das tun, was er oder sie will, so wie er oder sie es erwartet, ohne Hindernisse, Zögern oder Fragen. Um nutzbar zu sein, sollte ein Produkt oder eine Dienstleistung nützlich sein, effizient, effektiv, befriedigend, erlernbar und zugänglich. Die genauen Wort-Definitionen können in [86] nachgeschlagen werden.

Bewährte Verfahren zur Evaluation [36] sind:

analytische Methoden Expertisen und kognitive Walktroughs
empirische Methoden kontrollierte Experimente, Fragebögen, Interviews und Fokusgruppen.

Beide Ansätze erfahren in der Literatur viel Zuspruch. Sie sind allerdings kritisch zu betrachten: Viele der empirischen Studien bestehen in der Regel nur aus einfachen Aufgaben, welche fern von der tatsächlichen späteren Anwendung ausfallen [80]. Trotzdem findet der empirische Ansatz am meisten Verwendung [87]. Dazu ist es schwierig Tools mit analytischen Methoden zu vergleichen, da unterschiedliche Analyseaufgaben verschiedene Visualisierungstechniken erfordern [83]. Bewährt haben sich Kombinationen aus beiden Verfahren [87]. Analysen speziell für 3D-Software- Visualisierung sind noch rar, weswegen es bei neuen Technologien zu Herausforderungen der Evaluierung führen kann [88]. Empirische Bewertungsmethoden sammeln Usability-Daten, indem sie die Aktivitäten von Endanwendern beobachten oder messen, die mit einem Prototyp oder einer tatsächlichen Implementierung des Systems interagieren. Eine Nutzer Studie zu einem 3D-Visualisierungssystems namens *sv3D* [88, 89] zeigt, dass trotz eines positiven Gesamteindrucks der Anwender, es neue Schwierigkeiten mit der Interpretation und Anpassung an 3D-Darstellungen gibt.

In Abbildung 25 sind die Ergebnisse eines Literaturpapers [28] abgebildet. Die Mehrheit der publizierten Werke verwenden zur empirischen Evaluierung der Software Visualisierungen Beispiele und Experimente mit Nutzern. Auffällig ist (in der linken Tabelle) die hohe Zahl der Literatur ohne Evaluierung. Auch [79] untersucht 65 Veröffentlichungen für Informationsvisualisierungen und stellt fest, dass weniger als 20% der Autoren eine Evaluierung vorstellen oder diese als Aufgabe in zukünftiger Arbeit sehen (2006). Lediglich zwei dieser Publikationen eignen sich für eine sinnvolle Analyse, um Erfahrungen und Vorgehensweisen für zukünftige, erfolgreiche Evaluierungen herauszuarbeiten. Laut [70] könnte ein Grund für das Fehlen der Evaluierungen auch an der Akzeptanz von Software Visualisierungen sowohl in der freien Wirtschaft, als auch in der Forschung liegen.
Die mittlere Tabelle (Abbildung 25) zeigt die Methodik der Untersuchungen auf. Das häufigste verwendete Mittel ist der Fragebogen. Auch wird die *Think-Aloud* Technik [90] vermehrt angewendet. Dabei werden Probanden während gestellter Aufgaben gefilmt und alles was sie sagen aufgezeichnet. Die Testpersonen sollen wortwörtlich „laut Denken". Die aufgenommenen Daten werden für die Analyse des Designprozesses verwendet. Dadurch kann Wissen über die menschlichen Problemlösungen erforscht werden. Während diese Methode sehr gut für subjektive Empfindungen funktioniert, gibt sie keinerlei Rückmeldung über funktionelle Gegebenheiten. Diese Methode wird vermehrt für Design Studien während des Entwicklungsprozesses verwendet, um frühzeitig Probleme oder nicht intuitive Gegebenheiten zu eliminieren. Für Probanden ist es zumeist ungewohnt laut zu denken und dadurch können unangenehme Gefühle während der Befragung das Ergebnis beeinflussen.
Interessant sind die Indikatoren, die eine gute Visualisierung ausmachen. Auf der rechten Seite der Abbildung 25 befinden sich die abhängigen Variablen, die als Bewertungskategorien Anwendung finden. Oftmals leiden Evaluierungen unter zwei großen Problemen [79]: Wie definiert sich die Messbarkeit des Erfolgs bei Visualisierungs-Techniken und worin besteht der Zweck der Befragung. Als Mess-Variablen haben sich in der Literatur die Korrektheit der ausgeführten Tasks und die Dauer bewährt [70]. Neben der messbaren Performance anhand korrekter Antworten stehen auch Empfindungen der Testpersonen im Fokus. Dafür werden in [33] subjektive Skalen verwendet: sehr gut, gut, weniger gut, gar nicht gut und nicht messbar sind dabei Optionen. Effektive Vermittlung der Daten mit intuitiver Steuerung werden als Hauptkriterien untersucht.
In [57] wird mittels einer informellen Befragung der Testpersonen die Anwendung auf drei verschiedene Kategorien untersucht: Interaktion (d.h. Bewegung), Gefühle und Zeitwahrnehmung. Die Probanden hatten bereits Erfahrung mit Software Visualisierungs Tools, jedoch nicht mit dem verwendeten VR Device. Basierend auf den Ergebnissen von [89] wird die Begeisterungsfähigkeit und Immersion als positiv erachtet und mit Bewer-

Category	Strategy	#
Theoretical		2
No Explicit Evaluation		24
Empirical	Survey	4
	Anecdotal Evidence	6
	Case Study	12
	Experiment	53
	Example	83

Method	#
Questionnaire	37
Think-Aloud	17
Interview	12
Video Recording	9
Sketch Drawing	3
Others	3

Dependent Variable		#
User Performance	Not Explicit	2
	Time	15
	Correctness	29
	Effectiveness	17
	Completion	2
	Recollection	2
	Others	3
User Experience	Not Explicit	3
	Usability	14
	Engagement	2
	Understandability	2
	Feeling	5
	Others	3

Abbildung 25: Evaluationsverfahren für Software Visualisierung aus veröffentlichten IEEE SoftVis Publikationen (Stand 2016) [28].
Links ist die verwendete Strategie kategorisiert. In der Mitte werden die expliziten Methoden aufgezeigt und die Tabelle auf der rechten Seite zeigt die Kriterien und abhängigen Variablen der Evaluierung.

tungsskalen messbar gemacht. Probanden wollten mehr Zeit mit der Anwendung verbringen und weitere Aufgaben erfüllen. Der *Spaß*-Faktor bei Visualisierungen ist demnach nicht zu unterschätzen.

Auch ist bisher kaum erforscht wie viel Einfluss das Repräsentations- Medium auf das Ergebnis nimmt [31]. In [91] wurden eine Desktop Anwendung, ein 3D-Printmodell und eine VR-Anwendung miteinander verglichen. Es handelte sich jeweils um eine auf der City-Metapher basierende Visualisierung. Um gleiche Bedingungen der Test-Medien zu schaffen, erfolgte der Test ohne Hover-Funktion der Desktop- oder VR-Anwendung. Um die Vergleichbarkeit zu generieren werden die expliziten Eigenschaften der Test-Medien (leider) nicht genutzt.

Bei Evaluierungen sollten immer vergleichbare Aussagen als Ergebnis bestehen können. Die reine Auflistung der benötigten Zeit gibt keine aussagekräftige Ergebnisse [92]. Was bedeutet es beispielsweise, wenn eine Aufgabe innerhalb von 70 Sekunden ausgeführt werden kann? Daher sind Vergleiche sinnvoll. Ein Vergleich mit der VR-Darstellung [4] war zu Beginn der Thesis angedacht. Da die Portierung der Metapher nicht stattgefunden hat, ist ein Vergleich unbedeutend. Die Gemeinsamkeiten der Anwendungen wären nur minimal. Die Durchführung würde kaum Mehrwert erzeugen. Ein anderer Ansatz wäre der Vergleich von bisher gängigen Tools, die Software Entwickler verwenden. Angelehnt an [79], wo eine Stadt Metapher basierte Visualisierung mit Eclipse (eine beliebte Entwicklerumgebung) verglichen wird, wäre eine vorstellbare Evaluierung. Da allerdings bisher nur ein Prototyp entwickelt wurde und zudem auch Probanden am

Test teilnehmen, die keinerlei Programmier- Erfahrung haben, kann keine IDE zum Vergleich herangezogen werden. Daher wird sich auf kleine und einfach ausführbare Tasks beschränkt, die miteinander in Relation gesetzt werden können. Dadurch soll verhindert werden, dass sich die Probanden überfordert fühlen [88]. Für die weiteren entwickelten Mechanismen der Eingabe und des UI Designs werden lediglich Ideen vorgestellt, die in zukünftiger Arbeit genauer evaluiert werden sollen.

8.1 Testpersonen

Abhängig von der Evaluierung und den Zielen sollten die Testpersonen ausgesucht werden. In [83, 80, 15] werden Experten als die repräsentativsten Probanden beschrieben. Die Argumentation besteht darin, dass die tatsächliche Zielgruppe mit Ihrem speziellen Wissen Rückmeldung geben kann. Die Befragung solcher Experten gestaltet sich zumeist schwierig, da sie oftmals nur in geringer Anzahl zur Verfügung stehen [79].

Bei den Testpersonen handelt es sich um Mitarbeiter des DLRs. Sieben Mitarbeiter (4 weiblich, 3 männlich) nahmen an der Usability-Studie teil. Sechs davon haben Erfahrungen in der Informatik und verfügen über einen Abschluss (3 Bachelor, 1 Master, 1 Diplom und 1 PhD), mit durchschnittlich 6-9 Jahren Programmiererfahrung. Nur eine Testperson ist mit dem jeweiligen Thema nicht vertraut. Fünf Personen sind Rechtshänder und zwei Linkshänder. Es konnten keine Unterschiede in der Bedienbarkeit feststellt werden. Die Erfahrung mit Software-Architektur lag zwischen Anfänger und Fortgeschrittenen. Lediglich drei Probanden hatten die *HoloLens* schon einmal ausprobiert. Allerdings beschränkte sich die Erfahrung auch bei diesen auf wenige Minuten.

8.2 Testentwurf

Alle Testpersonen verwenden die gleiche *HoloLens* und befinden sich in der selben Umgebung (VR-Labor des DLR am Standort Köln Porz, dabei handelt es sich um einen Raum mit viel Freifläche auf dem Boden). Vor der jeweiligen Befragung findet eine kleine Einführung statt. Das Thema der Arbeit wird erläutert und anhand einer Abbildung die Darstellung der Komponenten erklärt (Bundles, Packages, Klassen, Services und Import-Export-Lagerhallen). Anschließend wird die *HoloLens* vorgestellt und die Möglichen Eingabemechanismen erklärt und vorgeführt. Die Testpersonen werden darauf hingeweisen die Gesten sehr nah am Sichtfeld auszuführen, da die Tiefen-Kamera diese sonst nicht registriert. Jede Testperson kann drei Minuten selbstständig bei einer Testanwendung die Gesten und Manipulationen ausprobieren. In der Literatur hat es sich bewährt ein kleines Tutorial vor dem eigentlich Testdurchlauf auszuführen [70, 15, 92]. Bei der Testanwendung handelt es sich um auswählbare Objekte, um die Air-Tap

Geste zu üben. Die Testpersonen können jederzeit Fragen stellen und Hilfestellungen zur Bedienung erhalten. Die Probanden führen den Test im Stehen durch. Vermeintlich schlecht geprobte oder noch nicht entdeckte Gesten werden vom Instruktor noch einmal erklärt und vorgeführt. Wenn die Testpersonen sich wohler im Umgang mit der Anwendung fühlen, erhalten sie einen Task nach dem anderen. Jeder Task wird sowohl verbal formuliert, als auch unterstützend in einem Video vorgeführt. Dabei sollten die Tasks zum einen nah an echten Anwendungsfällen liegen und zudem über eine zeitliche Begrenzung verfügen, ab wann sie als nicht ausgeführt gelten [70]. Auch sollte die Anzahl und der Aufwand der Aufgaben den Probanden nicht überfordern, da dies zu verfälschten Daten führen könnte [15, 79]. Nach jeder Aufgabe werden die Probanden zur Bedienbarkeit befragt. Verwendet wird ein Fragebogen mit Fragen zu geistigen, körperlichen und zeitlichem Anforderungen. Auch sollen die Personen Angaben über ihr Frustrations-Niveau, den Aufwand zur Bewältigung (Anstrengung) und Leistung geben. Verwendet wird der *NASA-Task Load Index* [93, 94]. Dabei können Indikatoren für Beanspruchung herausgefunden werden. Es handelt sich um ein Rating-Verfahren, welches die subjektive Wahrnehmung misst [95] (vgl. Abbildung 26). Nach jedem Task werden die Nutzer zu ihrer empfundenen Belastung befragt. Die Antworten werden anonym gespeichert, um den sozialen Druck zu mindern. Die Testpersonen nehmen alle freiwillig am Test teil. Anschließend werden allgemeine Fragen gestellt, damit die Testpersonen nicht vor dem Test ermüden.
Außer den messbaren Zeiten sowie den Fragebögen wurden die Teilnehmer während der Studie beobachtet und Kommentare aufgeschrieben.
Die aufgestellten Hypothesen lauten:

- Die Bedienmethoden mit Buttons und die Kombination aus Spracheingabe und Gesten-Steuerung wird als einfach und intuitiv wahrgenommen.
- Die Probanden sind interessiert an der Anwendung und würden sie gerne länger ausprobieren.
- Der Prototyp fördert das Verständnis von Software Architektur Visualisierung.

In [50] dienen einige kleine Aufgaben zur Evaluierung der dort entwickelten Anwendung und deren Interaktion. (Da die qualitative Analyse im Fokus steht, ist die Korrelation der Fragen nicht zu beachten.) Aufgrund dieser Fragen ergaben sich die folgenden angepassten Tasks für die befragten Testpersonen:

Task 1:
1. Platzieren Sie die Stadt so im Raum, wie es für sie angenehm ist.
2. Finden und Selektieren Sie das Gebäude *Class A*.

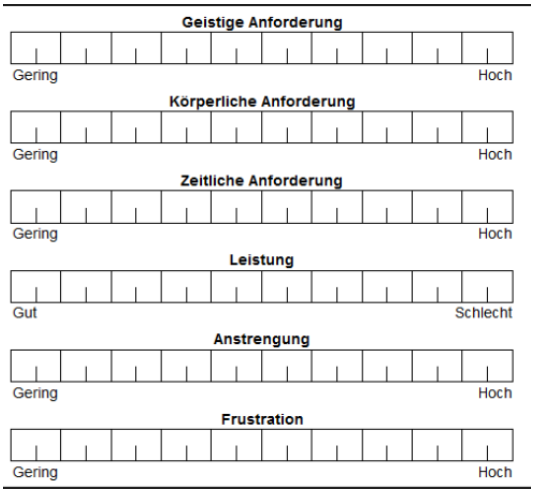


Abbildung 26: Skala des Nasa-Task Load Index [93, 95] zur Bestimmung der subjektiven Belastung der Probanden. Wortdefinitionen können im Anhang nachgelesen werden.

3. Finden und Selektieren Sie Gebäude *Service B*.

Task 2:
1. Selektieren Sie *Class D*
2. Verschieben Sie das Information Panel, falls es für Sie nicht gut positioniert ist.
3. Lesen Sie den zugehörigen Package-Pfad vor.

Task 3:
1. Aktivieren Sie die Import- und Export Stationen dieses Bundles.
2. Erstellen Sie über den Sprachbefehl „Screenshot“ einen Screenshot.

Task 4:
1. Öffnen Sie das Annotation Window
2. Erstellen Sie eine Annotation mit dem Inhalt „important note“ zu dieser Klasse (Dabei steht es Ihnen frei die Spracheingabe oder die virtuelle Tastatur zu verwenden)

8.3 Testergebnisse

Insgesamt nahmen sieben Probanden an der Nutzerstudie teil. Als wissenschaftlicher/ technischer Mitarbeiter oder studentische Hilfskraft, verfügen fast alle über Programmiererfahrungen (6 von 7). Die Erfahrungen liegen zwischen 6 und 9 Jahren. Lediglich vier haben bereits mit OSGi-Projekten gearbeitet, zwei davon länger als 3 Jahre. Zwei Probanden sind Linkshänder, der Rest Rechtshänder. Die Probanden hatten vor der Studie

wenig bis gar keine Berührung mit der *HoloLens* . Lediglich drei hatten bereits eine Beispielanwendung für wenige Minuten ausprobiert.
Für fünf Personen war generell das Tragen der *HoloLens* angenehm bzw. generell als nicht störend wahrgenommen worden. Zwei empfanden die Brille als unangenehm. Während der Durchführung der Aufgaben sind von drei verschiedenen Probanden Aussagen zum Komfort der Brille gefallen: „Die wird aber ganz schön schwer mit der Zeit“ (nach Task 2), „Das merke ich jetzt schon im Nacken“ (nach Task 2, jedoch wurde von dieser Person die Brille nicht zum Ausfüllen der empirischen Fragen abgesetzt), „Die Haptik ist unangenehm zu tragen“ (während Task 3). Insgesamt wurde die Brille zwischen 6 und 13 Minuten getragen. Zwei Testpersonen beschwerten sich über das dauernde Arm hochhalten und das dadurch begründete Ermüden der Arme. Eine Testperson wechselte sogar von der rechten Hand auf die Linke, um der Ermüdung vorzubeugen.

Drei würden die Anwendung definitiv wieder verwenden wollen, die restlichen erst nach eine Überarbeitung bzw. Verbesserung der Applikation. Auf die Frage, was verbessert werden könnte wurden folgende Punkte genannt: Das Sichtfeld sei zu klein und das hoch halten des Armes sei anstrengend. Gewünscht wurde zudem eine Suchfunktion für leichteres Auffinden von Klassen. Gebäude sollten sich nicht nur in ihrer Farbe und Höhe unterscheiden, sondern auch in der Gebäudeart. Zwei mal wurde bemängelt, dass die Selektion teilweise noch ungenau sei. Positiv erwähnt wurde, dass die Steuerung über das Menü sehr intuitiv sei.

Die Verwendung der Sprachbefehle ist für die meisten Probanden unangenehm. Vier empfinden die Verwendung als nicht unangenehm, die restlichen drei dafür umso mehr: „Wie ungewohnt so eine Eingabe zu tätigen“ & „Das wäre in einer anderen Umgebung nicht anwendbar“ sind Aussagen, die diese Meinung unterstreichen. Erkenntnis zur Verbesserung: Bei der Spracheingabe kommt es gelegentlich zu Irritationen, wenn der Befehl nicht erkannt wird. Eine Rückmeldung, wie „Sprachbefehl nicht erkannt!“ wäre sinnvoll. Das visuelle und akustische Feedback bei der Tätigung des Screenshots wurde gut aufgenommen. „Oh ich hab was gehört“ und „Das ging aber schnell“ bestätigen dies.

Insgesamt empfanden alle Teilnehmer die Aufgaben als leicht und werteten auch den Aufwand als sehr gering. In Tabelle 2 sind die durchschnittlichen Bewertungen jeder Kategorie für alle Tasks zusammengefasst. Die genauen Ergebnisse können im Anhang eingesehen werden. Lediglich bei Task 4, sind einige Komplikationen aufgetreten. Die virtuelle Tastatur hat teilweise nicht korrekt funktioniert, welches teilweise zu einem höheren Frustrations- Level der Probanden führte und die Ergebnisse etwas nach unten zieht.
Task 4 konnte mittels der virtuellen Tastatur gelöst werden, oder über die Spracheingabe. Drei Personen wählten die Spracheingabe und waren mit $\varnothing 1,14 min$ bedeutend schneller, als die Probanden, welche die Tastatur ver-

Geistige Anforderung	$1,96$
Körperliche Anstrengung	$2,14$
Zeitliche Anstrengung	$2,39$
Leistung	$2,32$
Anstrengung	$2,04$
Frustration	$2,46$
Gesamtwertung	$2,22$

Tabelle 2: Durchschnittliche Bewertung der Kategorien bei allen Tasks (7 ist das höchste und 1 die niedrigste Möglichkeit).

wendeten ($\varnothing 2,58min$). Zwei Probanden wechselten währen der Aufgabe die Optionen, da vorherige falsche Eingaben getätigt wurden. Dadurch verloren sie viel Zeit und benötigten im Schnitt ($\varnothing 4,29min$). Aufgrund dieser Tatsache erhielt dieser Task mit Abstand die höchste Bewertung für Frustration. Die Probanden, welche die Möglichkeit des Diktierens wahrgenommen haben, empfanden keinerlei Frustration, Anstrengung oder geistige bzw. körperliche Anstrengung. Die Teilnehmer hingegen, welche die einzelnen Buchstaben eingaben, bewerteten sich sehr schlecht in der Leistung, dem zeitlichen Aufwand und wiesen eine hohe Frustration auf. Hierbei ist die Skala zu relativieren, da die ersten Aufgaben wirklich sehr schnell ausgeführt werden konnten, gehen wir davon aus, dass die Probanden zusätzlich den zeitlichen Aufwand als immens wahrgenommen haben.

Bezogen auf die Hypothesen aus Abschnitt 8.2 lässt sich folgendes feststellen: Der User Interface Entwurf mit holografischen Buttons wird als intuitiv wahrgenommen. Auch wenn einige Nutzer die Sprach-Steuerung als kritisch ansahen, fanden andere diese als durchaus sinnvoll eingesetzt. Beide Eingabemethoden wurden implementiert, um den Nutzer die freie Entscheidung des Inputs zu ermöglichen.
Alle Probanden sind sehr interessiert an der Anwendung und würden diese gerne noch einmal testen. Einige allerdings erst nach einer Verbesserung der Darstellungen. Unterstützend ist hierzu zu erwähnen, dass fast alle Testpersonen im Anschluss des offiziellen Nutzertests die Anwendung ausprobierten. Nicht getestete Funktionen wurden ausprobiert und erhielten positives Feedback. Die Orientierungs-Karte wurde als sehr hilfreich betitelt.
Das Verständnis der Metapher ist schwer zu evaluieren. Eine repräsentative Studie müsste dafür durchgeführt werden. Insgesamt empfanden alle Probanden die City-Metapher als eine sinnvolle Repräsentation für Software Architektur Visualisierung (eine Person grenzte diese Aussage weiter ein: das kommt auf die Größe des Projektes an). Eine Testperson gab an, dies nicht beurteilen zu können. Als Anwendungsfälle sehen die Proban-

den hauptsächlich: Den Einsatz in der Lehre, Gefühl/Überblick für Software erhalten und dem Erkennen der Dependencies. Die Einarbeitung neuer Mitarbeiter oder zur Repräsentation eines Projektes nach Außen wurde nur mit einer Tendenz zum Erfolg eingeschätzt. Das Überprüfen der Struktur und zu Testzwecken wurde fast voll allen Probanden als nicht zielführend bewertet.

9 Fazit und Ausblick

Ziel der Software-Visualisierung ist es nicht, beeindruckende Bilder zu erzeugen, sondern Abbildungen, die zum besseren Verständnis der Software dienen [2]. Auf diese Weise können Ingenieure eine erste Vorstellung davon bekommen, wie Software strukturiert ist, die Softwarelogik verstehen, die Entwicklung erklären und kommunizieren. Software-Visualisierung kombiniert Techniken aus verschiedenen Bereichen wie Software-Engineering, Data Mining, Computergrafik, Informations-Visualisierung und Mensch-Computer-Interaktion [36].
OSGi-basierte Software ist zumeist groß und komplex. Gerade bei der Einarbeitung eines Entwicklers, erschweren diese Faktoren die Arbeit mit OSGi-basierter Software. Weswegen ein AR-Ansatz zur Exploration der Software Architektur Visualisierung basierend auf der 3D-Stadtmetapher unter Verwendung der Microsoft HoloLens vorgestellt wurde, um die Arbeit mit umfangreicher komplexer Software zu erleichtern. Import- und Export-Verknüpfungen zwischen den Bundles werden über verschiedenfarbige Lagerhallen dargestellt. Das Konzept ist übertragbar auf andere Darstellungsformen, wie beispielsweise die Insel-Metapher. Der entwickelte Prototyp kann über verschiedene Interaktions-Möglichkeiten gesteuert werden:
Spracheingabe, Blickpunkt- und Gesten-Steuerung. Der Aufbau der Software kann mittels der Explorations-Möglichkeit vom Nutzer besser ergründet und behalten werden, als reiner Quellcode.
Zudem wurde ein multifunktionales User Interface für Software Visualisierungen entworfen, welches für verschiedene Zielgruppen anpassbar ist. Die Flexibilität ermöglicht es neue Informationsquellen zu integrieren. Da in vielen entwickelten Visualisierungen die Zielgruppendefinition nicht eindeutig geklärt ist, ist ein anpassbares User Interface eine anzustrebende Lösung. Beispielsweise reicht einem Manager das Darstellen der Software-Stadt, um die Komplexität zu verstehen. Ein Entwickler hingegen kann sich auch für spezifischere Informationen interessieren, sei es für die Dokumentation, Kommentare oder sogar für den eigentlichen Quellcode. In der bisherigen Literatur wir der Fokus nicht auf anpassbare User Interfaces gelegt, geschweige denn auf benötigte Zusatzinformationen für einen tatsächlichen Mehrwert. Oftmals werden nur die Klassen-Namen der Komponenten angezeigt.
User Interfaces für Mixed Reality Anwendungen sind noch nicht ausreichen erforscht, weswegen keine genauen Guidelines oder Standards existieren. Viele Ideen müssen erst auf ihre Nutzbarkeit mit dem entsprechendem Darstellungs-Medium getestet werden. Aus frühen Phasen der Konzeption geht hervor, dass Informationen immer unmittelbar an dem betroffenen Objekt anzubringen sind. Das ständige Kopf-Drehen des Nutzers ist unangenehm und stört die Auffassungsgabe der Zusammengehörigkeit dieser Komponenten.

Generell ist die quantitative Analyse für Software Visualisierung schwierig [31]. Um die Bedienbarkeit der Anwendung *ARVis* zu überprüfen, wurde eine kleine Nutzer-Studie durchgeführt. Dabei wurden den Probanden vier Aufgaben zum Programmverständnis gestellt. Diese sollten sie lösen und anschließend deren benötigten Aufwand bewerten. Die Testergebnisse sind durchweg positiv. Die subjektiven Empfindungen der Probanden waren zumeist sehr gut, welches ein Indiz für ein intuitives Bedienerlebnis aufweist. Lediglich die Eingabe der Annotationen mittels der virtuellen Tastatur oder Spracheingabe wurden als schwierig und frustrierend befunden. In Abschnitt 2.6 sind die Kriterien einer guten Software Architektur Visualisierung aufgelistet und beschrieben. Im folgenden Abschnitt findet eine Bewertung der Umsetzung in *ARVis* statt:

- ✓ **Einfache Navigation:**
  Die Testpersonen konnten in geringer Zeit alle gestellten Aufgaben lösen. Dank der freien Bewegungsmöglichkeit der Probanden konnte sich gut in der Stadt zurecht gefunden werden. Die implementierte Orientierungs-Karte unterstützt den Nutzer bei der Navigation. Falls es jedoch einmal zu einer Orientierungslosigkeit kommen sollte, kann durch das Zurücksetzen der Stadt der Anfangszustand wiederhergestellt werden.

- ✓ **Hoher Informationsgehalt:**
  ARVis bietet in den Grundeinstellungen nur die nötigsten Informationen für den Nutzer. Die Visualisierung zeigt die Software Architektur Visualisierung und über das Fokussieren und Anwählen der Komponenten werden Zusatzinformationen bereit gestellt. Alle diese Informationen können zudem beliebig abgewählt werden. Ist ein Nutzer an weiteren Daten interessiert, kann er diese hinzuwählen. In der Nutzer-Studie (vgl. Kapitel 8) zeigte sich, dass verschiedene Nutzer nach unterschiedlichen Zusatzinformationen fragten. Erfahrene Programmierer fanden das Einblenden der Dokumentation oder mögliche Notizen zu den Klassen sinnvoll. Die Anwendung bietet viele Informationen, wenn diese gewünscht sind. Ansonsten wird sich nur auf das essentielle beschränkt, um den Nutzer nicht zu überfordern.

- ✓× **Einfache Darstellungsart der Visualisierung:**
  Die Visualisierung beschränkt sich auf abstrakte Komponenten. Die Gebäude unterscheiden sich derzeit nur in Höhe und Farbe. Zwei Probanden wünschten sich mehr Variation, um unterschiedliche Komponenten schneller zu erkennen. Dank der einfachen Darstellung wird nicht von dem eigentlichen Zweck der Anwendung abgelenkt.

- × **Mehrere Detaillevel:**
  ArVis bietet keine unterschiedlichen Detail-Level. Über Skalierung,

Zoom und Rotation können andere Blickpunkte generiert werden. Der Nutzer kann zudem frei um die Hologramme herum gehen. Ein weiteres Abstraktionslevel ist geplant: Die einfache Darstellung der Bundles mittels Quadern und deren Import- und Export-Lagerhallen.

✓× **Stabilität gegenüber geringen Änderungen:**
Da es sich um einen Prototyp handelt und (noch) kein echtes Projekt, ist diese Anforderung schwer zu bewerten. Tendenziell soll in weiterer Forschung der Layout-Algorithmus aus [4] Anwendung finden. Dieser ist bei geringen Änderungen robust. Die Positionierung der Bundles erfolgt anhand ihrer Anzahl der Verknüpfungen, weswegen bei kleinen Änderungen keine große Umstrukturierung notwendig ist.

✓ **Visuelle Metaphern:**
Als visuelle Metapher wird die City-Metapher vorgestellt. Diese hat sich in vorheriger Literatur etabliert und alle Testpersonen empfanden sie als wirksam. Eine detaillierte Nutzer-Studie über den Nutzen visueller Metaphern und verschiedener Varianten fehlt derzeit. Weitere Arbeiten sollten nicht nur den Nutzen von Software Architektur Visualisierung untersuchen, sondern auch den Effekt der gewählten Metapher.

✓ **Anpassbares User Interface:**
Aufgrund der spezifischen Anpassbarkeit des User Interfaces an verschiedene Zielgruppen, wird mit *ARVis* auch ein anpassbares User Interface vorgestellt:

- ✓ Navigation: Rotation, Verschiebung, Zoom und das Wechseln von Ansichten ist enthalten.
- ✓ Filterung: Das Reduzieren der angezeigten Daten wird ermöglicht.
- ✓ Selektion: Das Auswählen von Elementen, mit besonderer Interessen, wird unterstützt.
- × Kodierung: Anpassung von grafischen Variablen ermöglichen.

✓ **Integration von anderen Informationsquellen:**
Durch die geplante Funktion der integrierten Dokumentation und des Quellcodes ist diese Forderung erfüllt. Das Aufnehmen einer Annotation zu einer Klasse speichert das Gedankengut eines oder mehrer Entwickler in Kombination mit den existierenden Fakten (vorhandene Klassen und Hierarchien).

× **Anpassbare Automatisierung:**
Derzeit ist nur ein Prototyp entwickelt, welcher die Visualisierung

aus Projekten nicht dynamisch erstellt. Das Erzeugen aus Projekten muss in zukünftiger Arbeit umgesetzt werden.

Viele Anforderungen an eine gute Visualisierung wurden bereits umgesetzt. Dennoch sind noch nicht alle Anforderungen erfüllt und der Prototyp muss Verbesserung erfahren. Zukünftige Arbeit wird darin bestehen, reale Softwareprojekte automatisch zu importieren und die Grafiken der Präsentation zu optimieren. Weitere Funktionen wie z.B. die Suche oder das Einbinden von Quellcode und Dokumentationen sollen implementiert werden. Interessant sind auch Team-Sharing Möglichkeiten. Gemeinsames Betrachten und Arbeiten mit der Stadt ist ein sinnvolles Anwendungsgebiet. Eine Optimierung der Übertragungsgeschwindigkeit der angebrachten Kamera für einen Bildschirm wäre ein weiterer Punkt, um das Erlebnis zu teilen.
Zur besseren Auswertung der Ergebnisse sollte auch eine repräsentative Nutzerstudie durchgeführt werden. Das Bewerten mit richtigen Anwendern ist im kompletten Forschungsbereich noch zu wenig umgesetzt worden. Es existieren keine bis kaum Statistiken über den eigentlich Nutzen von Software Architektur Visualisierungen. Eine Erforschung des Mehrwertes wäre vorteilig, aber schwer umzusetzen. Die Anwendungstauglichkeit in der freien Wirtschaft und Lehre sollten untersucht werden. Die meisten Visualisierungen sind in akademischen Forschungsprojekten entstanden und praktikabel. Eine Integration in Arbeitsumgebungen, wie beispielsweise Eclipse wäre ein denkbar hilfreicher Schritt für Entwickler [36].
Das vorgestellte UI Prinzip könnte auch für andere Hardware oder Metaphern angewendet werden. Bisweilen kann die Stadt auf glatten Oberflächen angebracht werden und somit beispielsweise eine Tischplatte überlagern. Es besteht kaum ein Unterschied zu einer VR Anwendung. Im Gegensatz zu VR Anwendungen ist die *HoloLens* allerdings nicht auf Räumlichkeiten beschränkt und kann auch beispielsweise mit physischen Devices gekoppelt werden. Sinnvoll wären Bildschirm und Tastatur, da die Auflösung und Lesbarkeit bei herkömmlichen Bildschirmen besser ist und auch die virtuelle Tastatur oder die Sprach- Diktierung noch ausbaufähig sind.

Der entwickelte Prototyp ist ein Tool, welches sich gut für die Repräsentation eignet. Auch die Vorstellung des Projekts für Manager oder Projektleiter (fachfremd) lässt sich umsetzen. Es kann verdeutlicht werden, wie komplex und zusammenhängend Aufgaben sind und wie ein Fortschritt zustande gekommen ist. In der Lehre kann der Prototyp für kleine Beispielprojekte verwendet werden. Dank der Annotationen und Screenshot Funktion können Gruppen Zustände teilen und diskutieren. Die Arbeit eines Entwicklers wird mit der Anwendung weniger unterstützt. Das Lesen von Quellcode lässt sich besser am Computerbildschirm umsetzten. Derzeit eignet sich der Ansatz nicht für große Software-Projekte. Der gewollte Überblick der Software wird anhand der Datenmassen nicht erreicht.

Trotzdem sollte weiter an Software Architektur Visualisierung in Augmented Reality gearbeitet werden: Im Vergleich zu 2D-Visualisierungen ist die dritte Dimension besser nutzbar und die Navigation ist intuitiver. Die derzeitigen Limitationen der *HoloLens* erschweren momentan die Vorstellung, das Gerät in den Arbeitsalltag einzubauen. Aufgrund der geringen Rechenleistung und des beschränkten Bildschirms, ist die Anwendung derzeit nicht für den realen Einsatz bereit. Die vielversprechenden Ankündigungen über die neue HoloLens und das größere Sichtfeld lassen allerdings auf bessere Ergebnisse hoffen. Zusätzlich ist davon auszugehen, dass neue Funktionalitäten im MR-Toolkit die Entwicklung immens vereinfachen werden. Auch soll der Arbeitsspeicher der eingebauten HPU erweitert werden, wodurch grafische Verbesserungen möglich wären.

Literatur

[1] Pierre Caserta and Olivier Zendra. Visualization of the static aspects of software: A survey. *IEEE transactions on visualization and computer graphics*, 17(7):913–933, 2011.

[2] Stephan Diehl. *Software visualization: visualizing the structure, behaviour, and evolution of software*. Springer Science & Business Media, 2007.

[3] Sheelagh Carpendale and Yaser Ghanam. A survey paper on software architecture visualization. Technical report, University of Calgary, 2008.

[4] Martin Misiak. Immersive Exploration of OSGi Based Software Architectures in Virtual Reality. Master's thesis, Technische Hochschule Köln, 2017.

[5] Tobias Marquardt. Extraktion und Visualisierung von Beziehungen und Abhängigkeiten zwischen Komponenten großer Softwareprojekte. Master's thesis, Technische Universität Dortmund, 2016.

[6] Andre LC Tavares and Marco Tulio Valente. A gentle introduction to OSGi. *ACM SIGSOFT Software Engineering Notes*, 33(5):8, 2008.

[7] Bernd Johannes Kolb, Nils Hartmann, Matthias Lübken, and Gerd Wütherich. *Die OSGi Service Platform: Eine Einführung mit Eclipse Equinox*. dpunkt. verlag, 2015.

[8] Doreen Seider, Andreas Schreiber, Tobias Marquardt, and Marlene Brüggemann. Visualizing modules and dependencies of osgi-based applications. In *Software Visualization (VISSOFT), 2016 IEEE Working Conference on*, pages 96–100. IEEE, 2016.

[9] Ronald T Azuma. A survey of augmented reality. *Presence: Teleoperators and virtual environments*, 6(4):355–385, 1997.

[10] Christopher Rohs. *Die erweiterte Realität: Einsatzgebiete und Potential von Augmented Reality*. diplom.de, 2015.

[11] Paul Milgram and Fumio Kishino. A taxonomy of mixed reality visual displays. *IEICE TRANSACTIONS on Information and Systems*, 77(12):1321–1329, 1994.

[12] Jon Peddie. *Augmented reality: Where we will all live*. Springer, 2017.

[13] Ivan E Sutherland. The ultimate display. *Multimedia: From Wagner to virtual reality*, pages 506–508, 1965.

[14] Ivan E Sutherland. A head-mounted three dimensional display. In *Proceedings of the December 9-11, 1968, fall joint computer conference, part I*, pages 757–764. ACM, 1968.

[15] Ralf Dörner, Wolfgang Broll, Paul Grimm, and Bernhard Jung. *Virtual und augmented reality (VR/AR): Grundlagen und Methoden der Virtuellen und Augmentierten Realität*. Springer-Verlag, 2014.

[16] Matthias Wille, Britta Grauel, and Lars Adolph. Head-Mounted Displays–Bedingungen des sicheren und beanspruchungsoptimalen Einsatzes. In *Datenbrillen–aktueller Stand von Forschung und Umsetzung sowie zukünftiger Entwicklungsrichtungen. Tagungsdokumentation anlässlich des Workshops „Datenbrillen–aktueller Stand von Forschung und Umsetzung sowie zukünftiger Entwicklungsrichtungen "im Rahmen des Forschungsprojektes F*, volume 2288, pages 6–11, 2012.

[17] Jonathan C Roberts, Panagiotis D Ritsos, Sriram Karthik Badam, Dominique Brodbeck, Jessie Kennedy, and Niklas Elmqvist. Visualization beyond the desktop–the next big thing. *IEEE Computer Graphics and Applications*, 34(6):26–34, 2014.

[18] Bernard C Kress and William J Cummings. Optical architecture of hololens mixed reality headset. In *Digital Optical Technologies 2017*, volume 10335, page 103350K. International Society for Optics and Photonics, 2017.

[19] Laura Toler. Holographic rovers: Augmented reality and the microsoft hololens. 2017.

[20] A Narasima Venkatesh. Connecting the dots: Internet of things and human resource management. 2017.

[21] Mark A Hoffman. The future of three-dimensional thinking. *Science*, 353(6302):876–876, 2016.

[22] Allen G Taylor. *Develop Microsoft HoloLens Apps Now*. Springer, 2016.

[23] Microsoft:https://docs.microsoft.com/de-de/windows/mixed-reality. Eingesehen am: 08.04.2018.

[24] Unity Game Engine: https://unity3d.com/de/.

[25] Mixed reality toolkit: https://github.com/microsoft/mixedrealitytoolkit-unity. Eingesehen am: 04.04.2018.

[26] Thomas Ball and Stephen G Eick. Software visualization in the large. *Computer*, 29(4):33–43, 1996.

[27] John Grundy and John Hosking. High-level static and dynamic visualisation of software architectures. In *Visual Languages, 2000. Proceedings. 2000 IEEE International Symposium on*, pages 5–12. IEEE, 2000.

[28] Leonel Merino, Mohammad Ghafari, and Oscar Nierstrasz. Towards actionable visualisation in software development. In *Software Visualization (VISSOFT), 2016 IEEE Working Conference on*, pages 61–70. IEEE, 2016.

[29] Kang Zhang. *Software Visualization: From Theory to Practice*, volume 734. Springer Science & Business Media, 2012.

[30] Juraj Vincur, Pavol Navrat, and Ivan Polášek. Vr city: Software analysis in virtual reality environment. In *Software Quality, Reliability and Security Companion (QRS-C), 2017 IEEE International Conference on*, pages 509–516. IEEE, 2017.

[31] Andrian Marcus, Louis Feng, and Jonathan I Maletic. 3D representations for software visualization. pages 27–ff, 2003.

[32] Jonathan I Maletic, Andrian Marcus, and Michael L Collard. A task oriented view of software visualization. In *Visualizing Software for Understanding and Analysis, 2002. Proceedings. First International Workshop on*, pages 32–40. IEEE, 2002.

[33] Peter Young and Malcolm Munro. Visualising software in virtual reality. In *Program Comprehension, 1998. IWPC'98. Proceedings., 6th International Workshop on*, pages 19–26. IEEE, 1998.

[34] Matthew O Ward, Georges Grinstein, and Daniel Keim. *Interactive data visualization: foundations, techniques, and applications*. CRC Press, 2010.

[35] Ji Soo Yi, Youn ah Kang, and John Stasko. Toward a deeper understanding of the role of interaction in information visualization. *IEEE transactions on visualization and computer graphics*, 13(6):1224–1231, 2007.

[36] Alfredo R Teyseyre and Marcelo R Campo. An overview of 3D software visualization. *IEEE transactions on visualization and computer graphics*, 15(1):87–105, 2009.

[37] Jock D. Mackinlay. Automating the Design of Graphical Presentations of Relational Information. *ACM Trans. Graph.*, 5:110–141, 1986.

[38] Doreen Seider, Andreas Schreiber, Tobias Marquardt, and Marlene Brüggemann. Visualizing modules and dependencies of osgi-based applications. In *Software Visualization (VISSOFT), 2016 IEEE Working Conference on*, pages 96–100. IEEE, 2016.

[39] Richard Wettel and Michele Lanza. Program comprehension through software habitability. In *Program Comprehension, 2007. ICPC'07. 15th IEEE International Conference on*, pages 231–240. IEEE, 2007.

[40] Thomas Panas, Thomas Epperly, Daniel Quinlan, Andreas Saebjornsen, and Richard Vuduc. Communicating software architecture using a unified single-view visualization. In *Engineering Complex Computer Systems, 2007. 12th IEEE International Conference on*, pages 217–228. IEEE, 2007.

[41] Sazzadul Alam and Philippe Dugerdil. EvoSpaces: 3D Visualization of Software Architecture. In *SEKE*, volume 7, pages 500–505, 2007.

[42] H Yang and H Graham. Software metrics and visualisation. *Univ. of Auckland, Tech. Rep*, 2003.

[43] Hamish Graham, Hong Yul Yang, and Rebecca Berrigan. A solar system metaphor for 3D visualisation of object oriented software metrics. In *Proceedings of the 2004 Australasian symposium on Information Visualisation-Volume 35*, pages 53–59. Australian Computer Society, Inc., 2004.

[44] Andreas Dieberger and Jolanda G Tromp. The information city project-a virtual reality user interface for navigation in information spaces. *Proceedings of the Virtual Reality Vienna*, 93, 1993.

[45] Claire Knight and Malcolm Munro. Comprehension with [in] virtual environment visualisations. In *Program Comprehension, 1999. Proceedings. Seventh International Workshop on*, pages 4–11. IEEE, 1999.

[46] Claire Knight and Malcolm Munro. Virtual but visible software. In *Information Visualization, 2000. Proceedings. IEEE International Conference on*, pages 198–205. IEEE, 2000.

[47] SC Eick, Joseph L Steffen, and Eric E Sumner. Seesoft-a tool for visualizing line oriented software statistics. *IEEE Transactions on Software Engineering*, 18(11):957–968, 1992.

[48] Thomas Panas, Rebecca Berrigan, and John Grundy. A 3D metaphor for software production visualization. In *Information Visualization, 2003. IV 2003. Proceedings. Seventh International Conference on*, pages 314–319. IEEE, 2003.

[49] Clinton Jeffery and Jafar Al-Gharaibeh. *Writing Virtual Environments for Software Visualization*. Springer, 2014.

[50] Florian Fittkau, Alexander Krause, and Wilhelm Hasselbring. Exploring software cities in virtual reality. In *Software Visualization (VISSOFT), 2015 IEEE 3rd Working Conference on*, pages 130–134. IEEE, 2015.

[51] Jonathan I Maletic, Jason Leigh, and Andrian Marcus. Visualizing software in an immersive virtual reality environment. In *Proceedings of ICSE*, volume 1, pages 12–13, 2001.

[52] Jonathan I Maletic, Jason Leigh, Andrian Marcus, and Greg Dunlap. Visualizing object-oriented software in virtual reality. In *Program Comprehension, 2001. IWPC 2001. Proceedings. 9th International Workshop on*, pages 26–35. IEEE, 2001.

[53] Carolina Cruz-Neira, Daniel J Sandin, and Thomas A DeFanti. Surround-screen projection-based virtual reality: the design and implementation of the cave. In *Proceedings of the 20th annual conference on Computer graphics and interactive techniques*, pages 135–142. ACM, 1993.

[54] Ulrika Wiss, David Carr, and Håkan Jonsson. Evaluating three-dimensional information visualization designs: A case study of three designs. In *Information Visualization, 1998. Proceedings. 1998 IEEE Conference on*, pages 137–144. IEEE, 1998.

[55] Jason Leigh, Andrew E Johnson, Christina A Vasilakis, and Thomas A DeFanti. Multi-perspective collaborative design in persistent networked virtual environments. In *Virtual Reality Annual International Symposium, 1996., Proceedings of the IEEE 1996*, pages 253–260. IEEE, 1996.

[56] Philippe Dugerdil and Sazzadul Alam. Execution trace visualization in a 3D space. In *Information Technology: New Generations, 2008. ITNG 2008. Fifth International Conference on*, pages 38–43. IEEE, 2008.

[57] Leonel Merino, Mohammad Ghafari, Craig Anslow, and Oscar Nierstrasz. Cityvr: Gameful software visualization. In *Software Maintenance and Evolution (ICSME), 2017 IEEE International Conference on*, pages 633–637. IEEE, 2017.

[58] Rodrigo Souza, Bruno Silva, Thiago Mendes, and Manoel Mendonça. SkyscrapAR: An augmented reality visualization for software evolution. In *Proc. of 2nd Brazilian Workshop on Software Visualization (WBVS 2012)*, 2012.

[59] Taeheon Kim, Bahador Saket, Alex Endert, and Blair MacIntyre. Visar: Bringing interactivity to static data visualizations through augmented reality. *arXiv preprint arXiv:1708.01377*, 2017.

[60] Anders Mikkelsen, Sondre Honningsøy, Tor-Morten Grønli, and George Ghinea. Exploring microsoft hololens for interactive visualization of uml diagrams. In *Proceedings of the 9th International Conference on Management of Digital EcoSystems*, pages 121–127. ACM, 2017.

[61] Patrick Reipschläger, Burcu Kulahcioglu Ozkan, Aman Shankar Mathur, Stefan Gumhold, Rupak Majumdar, and Raimund Dachselt. DebugAR: Mixed Dimensional Displays for Immersive Debugging of Distributed Systems. In *Extended Abstracts of the 2018 CHI Conference on Human Factors in Computing Systems*, page LBW117. ACM, 2018.

[62] Feng Zhou, Henry Been-Lirn Duh, and Mark Billinghurst. Trends in augmented reality tracking, interaction and display: A review of ten years of ISMAR. In *Proceedings of the 7th IEEE/ACM International Symposium on Mixed and Augmented Reality*, pages 193–202. IEEE Computer Society, 2008.

[63] Gregory Goth. Brave NUI world. *Communications of the ACM*, 54(12):14–16, 2011.

[64] Shao-Ning Chang and Wei-Lun Chen. Does visualize industries matter? A technology foresight of global Virtual Reality and Augmented Reality Industry. In *Applied System Innovation (ICASI), 2017 International Conference on*, pages 382–385. IEEE, 2017.

[65] Geoffrey S Hubona, Gregory W Shirah, and David G Fout. 3D object recognition with motion. In *CHI'97 extended abstracts on Human factors in computing systems*, pages 345–346. ACM, 1997.

[66] Monica Tavanti and Mats Lind. 2D vs 3D, implications on spatial memory. In *Information Visualization, 2001. INFOVIS 2001. IEEE Symposium on*, pages 139–145. IEEE, 2001.

[67] Colin Ware and Glenn Franck. Viewing a Graph in a Virtual Reality Display is Three Times as Good as a 2D Diagram. 1994.

[68] Colin Ware, David Hui, and Glenn Franck. Visualizing object oriented software in three dimensions. In *Conference Proceedings, pp*, volume 612, page 620. Citeseer.

[69] Craig Anslow, Stuart Marshall, James Noble, Ewan Tempero, and Robert Biddle. User evaluation of polymetric views using a large visualization wall. In *Proceedings of the 5th international symposium on Software visualization*, pages 25–34. ACM, 2010.

[70] Richard Wettel, Michele Lanza, and Romain Robbes. Software systems as cities: A controlled experiment. In *Proceedings of the 33rd International Conference on Software Engineering*, pages 551–560. ACM, 2011.

[71] Richard Wettel and Michele Lanza. Visualizing Software Systems as cities. In *Visualizing Software for Understanding and Analysis, 2007. VISSOFT 2007. 4th IEEE International Workshop on*, pages 92–99. IEEE, 2007.

[72] Jun Rekimoto and Mark Green. The information cube: Using transparency in 3D information visualization. In *Proceedings of the Third Annual Workshop on Information Technologies & Systems (WITS'93)*, pages 125–132, 1993.

[73] Raimund Dachselt and Anett Hübner. Three-dimensional menus: A survey and taxonomy. *Computers & Graphics*, 31(1):53–65, 2007.

[74] Ayan Chatterjee. *Building Apps for the Universal Windows Platform: Explore Windows 10 Native, IoT, HoloLens, and Xamarin*. Apress, 2017.

[75] Rudolph P Darken and Helsin Cevik. Map usage in virtual environments: Orientation issues. In *Virtual Reality, 1999. Proceedings., IEEE*, pages 133–140. IEEE, 1999.

[76] Internet article: How to develop for microsoft hololens : http://www.hypergridbusiness.com/2017/01/how-to-develop-for-microsoft-hololens/. Eingesehen am: 04.02.2018.

[77] Thomas Panas, Thomas Epperly, Daniel Quinlan, Andreas Saebjornsen, and Richard Vuduc. Communicating software architecture using a unified single-view visualization. In *Engineering Complex Computer Systems, 2007. 12th IEEE International Conference on*, pages 217–228. IEEE, 2007.

[78] Greg Parker, Glenn Franck, and Colin Ware. Visualization of large nested graphs in 3D: Navigation and interaction. *Journal of Visual Languages & Computing*, 9(3):299–317, 1998.

[79] Geoffrey Ellis and Alan Dix. An explorative analysis of user evaluation studies in information visualisation. In *Proceedings of the 2006 AVI workshop on BEyond time and errors: novel evaluation methods for information visualization*, pages 1–7. ACM, 2006.

[80] Catherine Plaisant. The challenge of information visualization evaluation. In *Proceedings of the working conference on Advanced visual interfaces*, pages 109–116. ACM, 2004.

[81] Melanie Tory and Torsten Moller. Evaluating visualizations: do expert reviews work? *IEEE computer graphics and applications*, 25(5):8–11, 2005.

[82] Andreas Kerren, Achim Ebert, and Jörg Meyer. Introduction to human-centered visualization environments. In *Human-Centered Visualization Environments*, pages 1–9. Springer, 2007.

[83] Robert Kosara, Christopher G Healey, Victoria Interrante, David H Laidlaw, and Colin Ware. Thoughts on user studies: Why, how, and when. *IEEE Computer Graphics and Applications*, 23(4):20–25, 2003.

[84] Penny Rheingans. Are we there yet? Exploring with dynamic visualization. *IEEE Computer Graphics and Applications*, 22(1):6–10, 2002.

[85] Jonathan Lazar, Jinjuan Heidi Feng, and Harry Hochheiser. *Research methods in human-computer interaction*. Morgan Kaufmann, 2017.

[86] Jeffrey Rubin and Dana Chisnell. *Handbook of usability testing: howto plan, design, and conduct effective tests*. John Wiley & Sons, 2008.

[87] Jakob Nielsen. Usability inspection methods. In *Conference companion on Human factors in computing systems*, pages 413–414. ACM, 1994.

[88] Andrian Marcus, Denise Comorski, and Andrey Sergeyev. Supporting the evolution of a software visualization tool through usability studies. In *Program Comprehension, 2005. IWPC 2005. Proceedings. 13th International Workshop on*, pages 307–316. IEEE, 2005.

[89] Jonathan I Maletic, Andrian Marcus, and Louis Feng. Source viewer 3D (sv3D): a framework for software visualization. In *Proceedings of the 25th International Conference on Software Engineering*, pages 812–813. IEEE Computer Society, 2003.

[90] MW Van Someren, YF Barnard, and JAC Sandberg. The think aloud method: a practical approach to modelling cognitive. 1994.

[91] Leonel Merino, Johannes Fuchs, Michael Blumenschein, Craig Anslow, Mohammad Ghafari, Oscar Nierstrasz, Michael Behrisch, and Daniel A Keim. On the Impact of the Medium in the Effectiveness of 3D Software Visualizations. In *Software Visualization (VISSOFT), 2017 IEEE Working Conference on*, pages 11–21. IEEE, 2017.

[92] Ben Shneiderman. *Designing the user interface: strategies for effective human-computer interaction*. Pearson Education India, 2010.

[93] Development of NASA-TLX (Task Load Index): Results of empirical and theoretical research.

[94] Sandra G Hart. NASA-task load index (NASA-TLX); 20 years later. In *Proceedings of the human factors and ergonomics society annual meeting*, volume 50, pages 904–908. Sage Publications Sage CA: Los Angeles, CA, 2006.

[95] Barbara-Ulrike Groß. Bestimmung von Schwierigkeitsgraden in einer zu entwickelnden Versuchsumgebung . Diplomarbeit, Universität Berlin, 2004.

Geistige Anforderung	Wie viel geistige Anforderung war bei der Informationsaufnahme und bei der Informationsverarbeitung erforderlich (z.B. Denken, Entscheiden, Rechnen, Erinnern, Hinsehen, Suchen...)? War die Aufgabe leicht oder anspruchsvoll, einfach oder komplex, erfordert sie hohe Genauigkeit oder ist sie fehlertolerant?
Körperliche Anforderung	Wie viel körperliche Aktivität war erforderlich (z.B. ziehen, drücken, drehen, steuern, aktivieren...)? War die Aufgabe leicht oder schwer, einfach oder anstrengend, erholsam oder mühselig?
Zeitliche Anforderung	Wie viel Zeitdruck empfanden Sie hinsichtlich der Häufigkeit oder dem Takt mit dem die Aufgaben oder Aufgabenelemente auftraten? War die Aufgabe langsam und geruhsam oder schnell und hektisch?
Leistung	Wie erfolgreich haben Sie Ihrer Meinung nach die vom Versuchsleiter (oder Ihnen selbst) gesetzten Ziele erreicht? Wie zufrieden waren Sie mit Ihrer Leistung bei der Verfolgung dieser Ziele?
Anstrengung	Wie hart mussten Sie arbeiten, um Ihren Grad an Aufgabenerfüllung zu erreichen?
Frustration	Wie unsicher, entmutigt, irritiert, gestresst und verärgert (versus sicher, bestätigt, zufrieden, entspannt und zufrieden mit sich selbst) fühlten Sie sich während der Aufgabe?

Abbildung 27: In der Nutzerstudie verwendete Übersetzung des NASA Task Load Indexes [93].

Task 1:

	Geistige Anforderung	Körperliche Anstrengung	Zeitliche Anstrengung	Leistung	Anstrengung	Frustration	Ø
A	1	2	1	1	2	1	
B	2	1	3	5	2	1	
C	2	1	2	1	1	1	
D	2	1	3	6	2	2	
E	1	1	1	1	1	1	
F	2	2	4	1	2	2	
G	3	2	6	1	2	5	
Ø	1,86	1,43	2,86	2,29	1,71	1,86	**2,00**

Task 2:

	Geistige Anforderung	Körperliche Anstrengung	Zeitliche Anstrengung	Leistung	Anstrengung	Frustration	Ø
A	1	2	1	1	2	1	
B	2	1	1	6	1	2	
C	2	1	1	1	1	1	
D	2	1	1	2	1	1	
E	1	1	1	1	1	1	
F	2	1	2	1	1	2	
G	5	1	4	1	3	5	
Ø	2,14	1,14	1,57	1,86	1,43	1,86	**1,67**

Task 3:

	Geistige Anforderung	Körperliche Anstrengung	Zeitliche Anstrengung	Leistung	Anstrengung	Frustration	Ø
A	1	2	1	1	2	1	
B	3	1	3	3	3	4	
C	2	2	2	1	1	1	
D	1	1	2	1	1	1	
E	1	1	1	2	2	2	
F	3	5	3	4	5	4	
G	2	2	2	1	1	2	
Ø	1,86	2,00	2,00	1,86	2,14	2,14	**2,00**

Task 4:

	Geistige Anforderung	Körperliche Anstrengung	Zeitliche Anstrengung	Leistung	Anstrengung	Frustration	Ø
A	2	3	3	2	3	2	
B	2	2	2	6	2	2	
C	3	3	2	1	2	1	
D	2	1	1	2	1	2	
E	1	5	3	5	1	7	
F	2	7	6	5	6	7	
G	2	7	5	2	5	7	
Ø	2	4	3	3	3	4	**3,21**

Øges.	**1,96**	**2,14**	**2,39**	**2,32**	**2,04**	**2,46**	**2,22**

Tabelle 3: Ergebnisse des Nutzertests (7 ist die höchste und 1 ist die niedrigste Möglichkeit).