INF
Informatik

Seminar Ausgewählter Themen

Visualisierung von Quellcode in spieleähnlichen Umgebungen

Tobias Braun
Matrikel-Nr.: 767369

Marco Kern
Matrikel-Nr.: 767027

Alexander Nuding
Matrikel-Nr.: 767636

BetreuerInnen: Prof. Dr. Christian Kücherer, Franziska Strobusch

Abgabedatum: 21. Juni 2022

Abstract:
[**Kontext und Motivation**] Visualisierung von Quellcode ermöglicht Entwickler:innen ein besseres Verständnis über verschiedene Eigenschaften einer Software zu erlangen. Quellcode von Softwaresystemen zu visualisieren wird mit zunehmender Größe des Softwaresystems immer komplexer. Die Übersichtlichkeit und Verständlichkeit der Visualisierung mit üblichen Methoden in 2D-Umgebungen ist dabei häufig nicht zufriedenstellend und erschwert das Einarbeiten in die Programmstruktur. Mithilfe von Werkzeugen wie CodePark und CodeCity lassen sich Softwaresysteme in 3D- und Extended-Reality-Umgebungen darstellen und übersichtlicher, sowie verständlicher strukturieren. [**Forschungsfrage/Problem**] Die Vorteile sowie Probleme und Herausforderungen der 3D Visualisierung im Bereich der Softwareentwicklung sind noch unklar. Inwiefern sich immersive Technologien hierbei gezielt einsetzen lassen um den Nutzen zu verbessern, oder ob diese zusätzlich das Ziel der besseren Übersichtlichkeit behindern, soll ebenfalls in dieser Arbeit untersucht werden. [**Lösungsideen/Ergebnisse**] Grundsätzlich erweitert sich der Raum zur Visualisierung von Quellcode in 3D-Umgebungen um eine weitere Dimension. Diese bietet eine erweiterte Tiefe der Strukturierung der Software, wodurch hierarchische Verhältnisse im Quellcode verbessert dargestellt werden können. Mit Hilfe einer Literaturrecherche werden bestehende Ansätze identifiziert und erläutert. Es werden die Vorteile aber auch Probleme der Ansätze aufgezeigt und bewertet. Ebenfalls wird überprüft bei welchen Problemen in der Softwareentwicklung diese Ansätze einen Vorteil gegenüber 2D Entwicklungsumgebungen bieten. [**Beitrag**] In diesem Artikel werden gezielt verschiedene Techniken zur Visualisierung von Quellcode in 3D Umgebungen, die Computerspielen ähneln, aufgegriffen und evaluiert. Dabei wird anhand von Beispielen aus der Grundlagenliteratur eine Problembeschreibung erstellt und Herausforderungen, sowie deren Lösungsmöglichkeiten, benannt.

Inhaltsverzeichnis

1 Einführung

Die Visualisierung von Software eignet sich dazu, die Struktur und das Verhalten eines Softwaresystems darzustellen. Dabei werden verschiedene Methoden und Werkzeuge angewandt. Üblicherweise kommen dabei Techniken zum Einsatz, die einfache Grafiken zur Darstellung verwenden, allerdings werden immer komplexere Methoden und Werkzeuge eingesetzt, die es ermöglichen eine Software in einer dreidimensionalen Umgebung zu visualisieren. Bei dieser Arbeit stehen Techniken im Fokus, die Softwaresysteme in spieleähnlichen Umgebungen, wie beispielsweise in grafischen Darstellungen mit Unity, visualisieren.

1.1 Ziele und Forschungsfragen

Das Ziel dieser Arbeit besteht darin, verschiedene Visualisierungsmöglichkeiten für Softwaresysteme vorzustellen und auf Probleme und Herausforderungen einzugehen. Dabei werden ausgewählte Methoden und Umgebungen zur Visualisierung erläutert und evaluiert. Ebenso wird geprüft bei welchen Problemen 3D Visualisierung in der Praxis effektiv angewendet werden kann, um die Arbeit der Entwickler:innen zu erleichtern. Die Arbeit behandelt die Nutzung von Virtual- und Augmented-Reality bei der Visualisierung von Quellcode und beleuchtet dabei Vor- und Nachteile. Dabei sollen die Ansätze und daraus resultierenden Probleme, aktueller 3D Code Visualisierungssysteme herausgearbeitet und wenn möglich Lösungen aufgezeigt werden.

1.2 Methodik

Diese Arbeit wurde im Rahmen einer Problemlösung für die Identifikation von Softwareeigenschaften verfasst. Die Literaturrecherche wurde dabei über eine gezielte Suche nach dem Schneeballprinzip und erstellte Suchterme durchgeführt. Dabei wurde für jede Forschungsfrage ein eigener Suchterm erstellt und eine passende Auswahl an Literatur getroffen. Die Auswahl wurde zusätzlich auf Artikel, deren Veröffentlichungsdatum nach dem 01.01.2000 liegt, beschränkt. Die Literaturdatenbanken von IEEE[1] und ACM[2] dienten als Hauptquellen für die Literatur aus dem Bereich der Informatik und Computerwissenschaften.

[1] https://ieeexplore.ieee.org/Xplore/home.jsp
[2] https://dl.acm.org/

1.3 Struktur der Arbeit

In Kapitel 2 wird die Visualisierung von Quellcode in Extended-Reality-Umgebungen behandelt. Dabei werden zwei Modelle, sowie immersive Technologien zur Visualisierung vorgestellt und evaluiert. In Kapitel 3 werden Probleme in der Software Entwicklung beleuchtet, bei welchen eine 3D Visualisierung gegenüber herkömmlichen integrierten Entwicklungsumgebungen (IDEs), wie Visual Studio (VS)[3] oder Eclipse[4], Vorteile bietet. Ebenso wird eine kombinierte Nutzung mit 2D IDEs betrachtet. Kapitel 4 behandelt die Vor- und Nachteile verschiedener, üblicher Visualisierungsformen, sowie Probleme und Herausforderung, die bei der Visualisierung und Navigation von Quellcode im 3D Raum auftreten. Abschließend werden unsere Ergebnisse in Form eines Fazit zusammengefasst und bewertet. Ebenso werden die Forschungsfragen dort erneut aufgegriffen und beantwortet.

[3] https://visualstudio.microsoft.com/de/
[4] https://www.eclipse.org/

2 Visualisierung von Quellcode in Extended-Reality-Umgebungen

Von Marco Kern

Die Softwareentwicklung ist ein Bereich, der eine Vielzahl von Aufgaben umfasst. Übergeordnet beschreibt die Softwareentwicklung das Herstellen und Entwerfen von Software in verschiedensten Einsatzbereichen, allerdings gleichzeitig auch die Wartung und das Erweitern von vorhandener Software. Vor allem in größeren Softwaresystemen können mit zunehmendem Alter und wachsender Komplexität Komplikationen ausgelöst werden, die es Entwickler:innen erschweren, sich in das Softwaresystem einzuarbeiten oder eine Übersicht zu bekommen [SMKP18]. Dadurch wird eine Visualisierung des Softwaresystems und dessen Quellcode zur Notwendigkeit, wobei dieses Vorgehen mittlerweile durch entwickelte Methoden und verschiedene Darstellungen über diverse Medien unterstützt wird.

Während traditionelle Visualisierungen, wie beispielsweise Komponenten- oder Klassendiagramme, auf herkömmlichen Entwicklungsumgebungen basieren, können neuartige Umgebungen einen Mehrwert an Darstellungsmöglichkeiten liefern. Visualisierungsmethoden nutzen hierbei oftmals Metaphern aus der realen Welt um Softwaresysteme in immersiven Umgebungen besser darstellen zu können, als die traditionellen Visualisierungsmethoden. Vor allem räumliche Darstellungen sind sehr beliebt, welche auch bei Visualisierungsmethoden wie CodeCity von Wettel et al.[SKR19] und CodeHouse von Hori et al.[HKI19] zum Einsatz kommen. Beide Werkzeuge machen Gebrauch von der räumlichen Darstellung und der Interaktion mit den Nutzer:innen, welche von immersiven Technologien zur Verfügung gestellt und unterstützt werden.

Die genannten Methoden und Technologien werden dieses Kapitel vorstellen. Dabei werden die verschiedenen Merkmale der einzelnen Visualisierungsmethoden erläutert und ein Vergleich zweier Methoden nach CodeCity und CodeHouse durchgeführt. Darüber hinaus werden Eigenschaften und Nutzungsmöglichkeiten von immersive Technologien im Zusammenhang mit den Visualisierungmethoden dargestellt. Abschließend befasst sich dieses Kapitel mit der Evaluation der benannten Techniken und bewertet dabei deren Nutzen im Zusammenhang mit der Interaktion eines Nutzers.

2.1 Visualisierungstechniken aus der Praxis

In der Praxis werden traditionelle Visualisierungstechniken, wie eine herkömmliche Entwicklungsumgebung, vorwiegend verwendet. Der Drang zukunftsorientierte Techniken dafür einzusetzen, steigt allerdings und die heranwachsende Generation an Softwa-

reentwicklern fördert in einigen Bereichen der Softwareentwicklung möglicherweise ein Umdenken [SMKP18].

2.1.1 CodeCity und CodeHouse als Visualisierungsmethoden

Eine Vielzahl an Studien befasst sich damit, neue Ansätze und Methoden zu finden, mit denen sich Softwaresysteme übersichtlicher und verständlicher darstellen lassen. Einige Studien nutzen dabei die Herangehensweise, Software als Gebäude oder Städte zu interpretieren [SKR19][HKI19]. Sowohl für CodeCity und CodeHouse existieren Werkzeuge, die zur Anwendung öffentlich und frei verfügbar sind. Die Werkzeuge beider Methoden wurden mit dem Ziel entwickelt, eine möglichst hohe Verständlichkeit für die Nutzer:innen zu haben. Das Ziel bezieht sich dabei auf das gesamte Softwaresystem, wobei die grundlegende Struktur ebenso wie die Details der einzelnen Teilkomponenten vermittelt werden sollen [HKI19].

Zwei konkrete Beispiele, die nach diesem Prinzip entwickelt wurden ist EvoStreets, welches von Steinbrückner entwickelt wurde [SKR19], und CodeHouse, wobei die erste Methode auf dem Modell der CodeCity aufbaut. Diese Methoden charakterisieren sich wie folgt:

EvoStreets: Abbildung 2.1 zeigt diese Methode. Diese basiert auf der Studie von Steinbrückner [SKR19] und fokussiert sich auf die metaphorische Darstellung von Quellcode als ein städteartiges Konstrukt. Dabei wird die interne Struktur des Softwaresystems mithilfe von Straßen abgebildet, wobei ein hierarchisches Gebilde entsteht. Einzelne Pakete sind mit diesen Straßen verbunden. Die Straßen teilen sich bei untergeordneten Paketen entsprechend auf und je tiefer die Ebene wird, desto dünner werden die Straßen. Einzelne Knoten, welche einzelne Dateien oder Klassen verkörpern können, werden meist als Würfel dargestellt, wobei diese Formen je nach Darstellung variieren können. Die Darstellung kann vom Nutzer beeinflusst werden, dieser kann gezielt Größe und Formen von Knoten oder Straßen durch gesammelte Metriken verändern. Abhängigkeiten zwischen einzelnen Knoten werden durch Verbindungen erkenntlich gemacht.

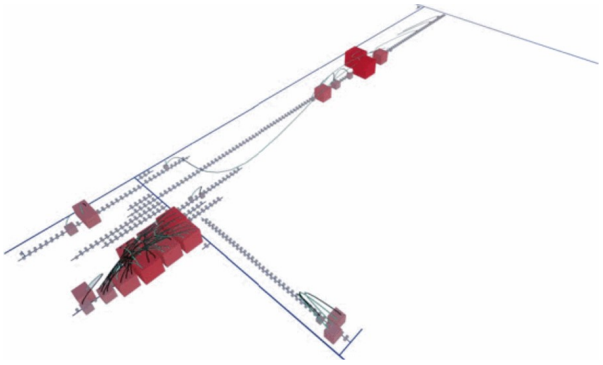

Abbildung 2.1: Visualisierung mit EvoStreets [SKR19]

CodeHouse: Abbildung 4.2 zeigt ein Beispiel dieser Methode. Diese arbeitet mit einer ähnlichen Darstellungsweise wie CodeCity, wobei diese sich metaphorisch an dem

Aufbau eines Hauses orientiert. Das Softwaresystem wird hierbei im Inneren eines Zylinders dargestellt. Die oberste Darstellungsebene ist die Systemstruktur, welche analog zu Stockwerken im Zylinder umgesetzt wird. Innerhalb dieses Stockwerkes sind die jeweiligen Module der einzelnen Ebenen als Räume dargestellt. Diese lassen sich virtuell betreten, wobei jeder Raum die verschiedenen Funktionen und Methoden darstellt und diese durch Linien als Verbindungen als abhängig erkenntlich macht. In der CodeHouse Visualisierungsmethode wird es den Nutzer:innen ebenfalls ermöglicht, durch Metriken Einfluss auf die Größe, Formen und Farben von den einzelnen Objekten in der Darstellung zu nehmen.

2.1.2 Immersive Technologien als Basis zur Visualisierung

Herkömmliche Methoden zur Darstellung von Softwaresystemen, wie beispielsweise Komponenten- oder Klassendiagramme, beziehen sich oftmals auf zweidimensionale Umgebungen oder dreidimensionale Darstellung in 2D-Umgebungen, auch 2,5D-Umgebungen genannt. Durch die Weiterentwicklung von immersiven Technologien werden Visualisierungstechniken, welche auf diesen Technologien aufbauen, immer interessanter und bieten eine Vielzahl an Mehrwert. Unter anderem kann bei der Nutzung solcher Technologien auf den immersiven Darstellungsraum zurückgegriffen werden, der sich aus einer dreidimensionalen Umgebungen mit verschiedenen Interaktionsmöglichkeiten ergibt.

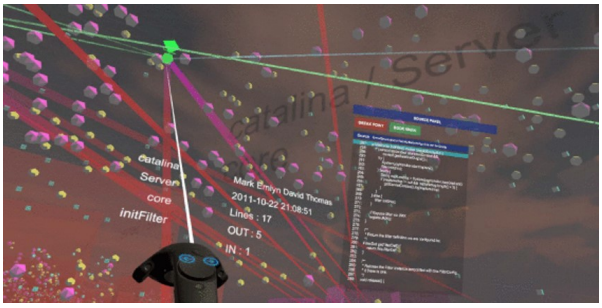

Abbildung 2.2: Visualisierung mit immersiven Technologien [HKI19]

Die Visualisierung von Softwaresystemen durch immersive Technologien wird bereits in verschiedenen Studien als Basis genutzt, um Visualisierungsmethoden umzusetzen [SKR19][HKI19]. Durch den virtuellen Raum [BW22] eröffnet sich den Nutzer:innen ein komplett anderes Erlebnis, das Softwaresystem zu verstehen und Zusammenhänge visuell wahrzunehmen. CodeHouse verwendet bereits einige dieser erweiterten Möglichkeiten und bietet die vollständige Navigation und Bedienung der Visualisierung über VR-Controller [HKI19]. Die Nutzer:innen können sich durch einfache Bedienung durch Zeigen oder verschiedene Gesten im System fortbewegen und frei umschauen und sind lediglich durch die Begrenzung des Raumes, in dem das Softwaresystem visualisiert ist, eingeschränkt. Zusätzlich bietet CodeHouse die Möglichkeit einen Debugger zu benutzen, um den Nutzer:innen eine zusätzliche Ebene zu bieten. Ein Debugger ist ein Werkzeug zum Auffinden von Fehlern Software. Dieser lässt sich ebenfalls per VR-Controller bedienen

und wurde von Hori et al. mit der Begründung eingebaut, die intuitive Bedienung zu fördern [HKI19].

Abbildung 2.3: Visualisierung mit immersiven Technologien [HKI19]

2.2 Evaluation der Visualisierungstechniken

In der Praxis werden überwiegend herkömmliche Entwicklungsumgebungen oder Visualisierungslösungen im zweidimensionalen Darstellungsraum verwendet, da die Visualisierung von Softwaresystemen mit immersiven Technologien bis zum Jahr 2022 noch nicht gründlich durch Studien erforscht wurde [SMKP18]. Da EvoStreets nach dem CodeCity Prinzip und das CodeHouse Prinzip als Visualisierungsmethoden im Rahmen von Studien im nahen Zeitraum vor 2022 entwickelt wurden, gibt es dabei wenig Informationen aus der Anwendung in der Praxis. Darüber hinaus ist es allerdings trotzdem möglich eine Evaluation über diese Visualisierungsmethoden, auch unter dem Einsatz von immersiven Technologien, durchzuführen.

2.2.1 Vor- und Nachteile von CodeCity und CodeHouse

CodeCity und CodeHouse sind beides Visualisierungsmethoden die sich anhand von Metaphern aus der realen Welt gebildet haben. Der Bezug dabei ergibt sich daraus, dass Softwaresysteme oftmals mit Gebäuden verglichen werden [HKI19] und die Nutzer:innen durch Vertrautheit mit der Umgebung besser zurecht kommen. Während CodeCity in 2D-, 2,5D- und immersiven Darstellungsräumen umgesetzt und angewendet werden kann, spezialisiert sich die CodeHouse Methode nur auf die Visualisierung mit immersiven Technologien. In diesem Unterkapitel werden ausschließlich die Merkmale der beiden Methoden untersucht, wobei die CodeCity Methode anhand von EvoStreets evaluiert wird. Die Umsetzung im immersiven Umgebungsraum wird im nächsten Unterkapitel separat betrachtet.

EvoStreets ermöglicht den Nutzer:innen eine kompakte, strukturierte Übersicht über das Softwaresystem. Durch die Unterschiede in der Größe einzelner sogenannter Straßen und Gebäude, entsteht für die Nutzer:innen eine verbesserte Wahrnehmung der einzelnen Teilkomponenten des Softwaresystems. EvoStreets bietet außerdem den Vorteil,

dass sich die Größe der einzelnen Darstellungsobjekte durch Metriken verändern lässt. Dadurch lassen sich gezielt Metriken einsetzen um Vergleiche zwischen Komponenten durchzuführen. Da EvoStreets sowohl in 2D- als auch in 2,5D- und immersiven Umgebungen dargestellt werden kann, bietet die Methode einerseits eine zusätzliche räumliche Wahrnehmung für die Nutzer:innen, allerdings können bei der Darstellung auch Probleme mit Verdeckungen einzelner Teile der Darstellung auftreten. Ein weiters Problem der EvoStreets Methode liegt darin, dass lediglich die Struktur des Softwaresystems dargestellt wird. Für den Nutzer liegt somit expliziter Quellcode in der Visualisierung verborgen und muss beispielsweise über eine Entwicklungsumgebung eingesehen werden.

CodeHouse ermöglicht den Nutzer:innen zusätzlich zur Visualisierung der Struktur eine Darstellung des Quellcodes inklusive Debugger. Die Struktur wird bei diesem Modell durch das Einteilen des Zylinders in stockwerkartige Schichten verdeutlicht. Innerhalb dieser Schichten sind die jeweiligen Module und Klassen als Quader dargestellt, wobei CodeHouse ebenfalls den Vorteil bietet, die Form und Farbe dieser Quader durch den Einsatz von Metriken zu beeinflussen. Die Nutzer:innen befindeen sich bei dieser Visualisierungsmethode im Zentrum des Zylinders und kann die einzelnen Darstellungsobjekte um sich herum betrachten. Dabei können jedoch ebenfalls Probleme durch Verdeckung einzelner Verbindungen zwischen Modulen und Klassen auftreten oder Unübersichtlichkeiten entstehen, wenn große Mengen an Verbindungen übereinander liegen. Da CodeHouse in immersiven Umgebungen genutzt werden kann, lässt sich die Bedienung und die Darstellung dadurch zusätzlich um die entsprechenden Funktionen, wie die Nutzung von VR-Controllern, erweitern. Allerdings ist CodeHouse nur im virtuellen Raum nutzbar und somit entsteht eine Hürde bei der Nutzung dieser Visualisierungsmethode, da ein Medium zwingend nötig ist, welches virtuelle Realität unterstützt. Meistens ist dies mit Mehrkosten für die Nutzer:innen verbunden.

2.2.2 Chancen und Probleme immersiver Technologien

Immersive Technologien sind eine Möglichkeit, die herkömmliche und traditionelle Visualisierung von Quellcode zu überdenken. Durch gezielte Gestiksteuerung kann die Interaktion mit der Visualisierung intuitiver gestaltet werden [SMKP18]. Dadurch werden nicht nur die kognitiven Fähigkeiten besser mit eingebunden, sondern ebenfalls ein schnellerer Umgang mit der Darstellung erreicht [MSK+20]. Dies geht aus einer Studie von Mehra et al. [MSK+20] hervor, wobei ein Experiment durchgeführt wurde, bei dem Probant:innen verschiedene Aufgaben in einer Softwarevisualisierung in unterschiedlichen Umgebungen, durchführen musste. Dabei wurde das Ergebnis erzielt, dass Nutzer:innen im immersiven Umgebungsraum eine kürzere Berarbeitung der Aufgaben hatten. Diese Zeitersparnis hatte laut Mehra et al. keinerlei negativen Einfluss auf die Richtigkeit und Genauigkeit des Ergebnisses. Das Problem bei dem Einsatz von immersiven Technologien besteht darin, dass die Nutzung mit einer Anschaffung eines Mediums zwingend verbunden ist. Ebenso ist der Einsatz dieser Technologien noch nicht sehr etabliert und erforscht [SKR19], weshalb das Angebot an nutzbaren Werkzeugen kleiner ist als bei herkömmlichen Visualisierungsumgebungen und somit weniger Variation bietet.

3 3D Visualisierung als Hilfsmittel in der Softwareentwicklung

Von Alexander Nuding

In der Softwareentwicklung werden verschiedenste Hilfsmittel und IDEs wie Visual Studio, Eclipse oder Intellij[1] eingesetzt um die Entwickler:innen bei ihrer Arbeit zu unterstützen. Seit dem Jahr 2001 wird auch immer mehr auf die Visualisierung in 3D gesetzt, so finden sich bis zum Jahr 2000 bei ACM nur 18 Artikel zum Thema 3D Visualisierung in der Softwareentwicklung. In den Jahren von 2001 bis 2010 wurden dagegen bereits 73 Artikel zu diesem Thema veröffentlicht, im Zeitraum von 2011 bis 2021 stieg die Anzahl dann auf 85.

3.1 Bewertung Einsatzgebiete der 3D Visualisierung in der Softwareentwicklung

Durch das wachsende Interesse an der 3D Visualisierung entstehen neue Werkzeuge für Entwickler die viele neue Möglichkeiten für die Softwareentwicklung bieten. So beschreiben Hori et al. in [HKI19] die Möglichkeit das Lernen von Struktur und Details einer neuen Software mithilfe einer 3D Visualisierung über CodeHouse zu vereinfachen. Für eine effektive Nutzung dieser neuen Möglichkeiten ist es wichtig diese bei Problemen einzusetzen bei denen sie Vorteile gegenüber den herkömmlichen IDEs wie Visual Studio bieten.

3.1.1 3D Visualisierung zur Erkennung von Duplikaten

Es gibt bereits Studien zu dem Erkennen von Code Duplikaten mit Hilfe von 3D Visualisierung, so vergleichen Steinbeck et al. in [SKR19] die EvoStreets Visualisierungstechnik in einer 2D, 2.5D sowie 3D Umgebung beim Lösen von Aufgaben im Hinblick auf Quellcodeduplikate. Die Studie hat 34 Teilnehmer:innen wobei EvoStreets als orthografische Projektion, 2.5D Umgebung und 3D Umgebung in Virtual Reality (VR) verwendet wurde. Die Teilnehmer:innen wurden in sechs Gruppen von je fünf bis sechs Teilnehmer:innen aufgeteilt, dadurch gab es für jede Umgebung pro Durchlauf 2 Gruppen. Für 2D, 2.5D und 3D mussten die Teilnehmer:innen jeweils als erstes herausfinden welches von zwei Systemen mehr Duplikate von Code Fragmenten besitzt. Die zweite

[1] https://www.jetbrains.com/de-de/idea/

Aufgabe bestand daraus zu bestimmen welche zwei Subsysteme am meisten dupliziert wurden, als letztes sollten die Teilnehmer:innen das Subsystem zu identifizieren welches die meisten Dateien mit Duplikaten enthält. Vor jedem Durchlauf wurden die Teilnehmer:innen dabei an einem Beispielsystem trainiert bis sich in der Lage fühlten die zuvor beschriebenen Aufgabe mit der aktuellen Umgebung zu lösen.

Aufgabe	1			2			3		
Umgebung	2D	2.5D	VR	2D	2.5D	VR	2D	2.5D	VR
min	20	69	56	51	32	56	69	32	54
max	331	205	953	152	335	208	304	381	362
Mittelwert	115.6	115.5	234.4	92.0	136.3	109.6	133.3	138.5	160.8
Median	115.0	107.0	129.5	89.0	119.5	105.0	107.0	111.0	124.0

Tabelle 3.1: Benötigte Zeit zur Bearbeitung der Aufgaben nach Umgebung [in Sekunden] [SKR19]

Bei der Betrachtung der Ergebnisse in Tabelle 3.1 ist auffällig das es vor allem bei der ersten und dritten Aufgabe einen deutlichen unterschied zwischen dem Medianwert und der Mittelwert bei der Bearbeitung mit VR gibt. Steinbeck et al. erklären in [SKR19] die großen Unterschiede damit, dass es mehrere Ausreißer gab. So gab es zwei bei der ersten Aufgabe mit VR, einen in Aufgabe zwei in 2.5D und zwei bei der dritten Aufgabe in VR. Dabei stammten der größte Ausreißer aus der ersten Aufgabe sowie der aus Aufgabe zwei und einer der Ausreißer aus Aufgabe drei von dem gleichen Teilnehmer:in welche keine vorherigen Erfahrungen mit Softwarevisualisierung und VR hatte.

Durch die Ausreißer ist eine Betrachtung der Medianwerte sowie der Minimalwerte zu bevorzugen um den Nutzen der Visualisierung in 3D zu bewerten. Bei dem suchen von Duplikaten schneidet die Visualisierung in 3D schlecht ab und wird sowohl beim Medianwert als auch beim Minimalwert jeweils erhalten die 2D oder 2.5D Visualisierung bessere Ergebnisse. Daher ist für im Bereich der Erkennung von Duplikaten eine 3D Visualisierung in VR mit EvoStreets nicht besonders geeignet da sie keine erkennbaren Vorteile bietet und spezielle Hardware benötigt. Es könnte allerdings vorteilhaft sein eine weitere Studie zu dem Thema zu machen mit Gruppen von Teilnehmer:innen die gut mit der ihnen zugewiesenen Visualisierungstechnik vertraut sind um mögliche Optimal werte festzustellen. Damit könnte festgestellt werden ob eine der Visualisierungstechniken auf lange Sicht große Vorteile gegenüber den anderen bieten kann.

3.1.2 3D Visualisierung zur Verbesserung des Lernens neuer Systeme

Eine andere Visualisierungstechnik stellen Khaloo et al. in [KMT+17] mit Code Park (CP) vor, hier werden einzelne Klassen eines Projektes als Räume in einer 3D Umgebung dargestellt. Das Ziel von CP hierbei ist es Entwickler:innen das lernen neuer Systeme zu erleichtern indem das Räumliche Erinnerungsvermögen der Nutzer:innen angesprochen wird. Über eine Studie wurde das System evaluiert, dabei waren die Teilnehmer:innen 28 Studenten:innen aus der Universität von Central Florida die bereits Erfahrungen mit der Programmiersprache C# sowie der Vergleichsumgebung Visual Studio (VS) hatten. Die Teilnehmer:innen sollten für die Studie fünf verschiedene Aufgaben zum Verständnis des Projektes jeweils mit VS und CP, wie das finden eines gültigen Nutzernamens mit dem zugehörigen Passwort(T1) sowie einer abstrakten Klasse(T2), bearbeiten. Ebenfalls sollte ein mögliche Stelle wo eine neue Funktion X implementiert werden könnte aufgezeigt werden(T5). Als Beispielprojekte wurde ein Konsolen basierter Bibliotheksmanager und eine Konsolen basiertes Gedächtnisspiel verwendet. Die Teilnehmer:innen wurden zufällig auf vier Gruppen aufgeteilt und bekamen ihr erstes Tool und Projekt zugeordnet so dass jede mögliche Kombination einmal vorhanden war. Im zweiten Durchgang wurden dann die Projekte und Tools gewechselt. Zur Bewertung der Visualisierungswerkzeuge wurde die Bearbeitungszeit gemessen. Ebenfalls sollte jeder der Teilnehmer:innen einen Fragebogen ausfüllen.

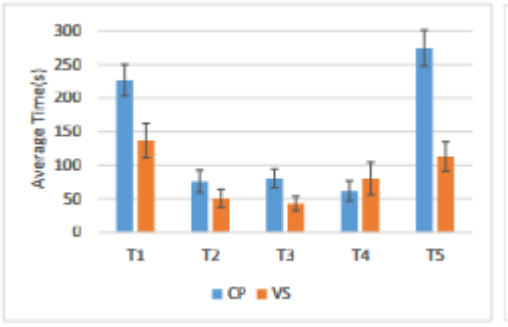

Abbildung 3.1: Medianwert der zur Bearbeitung gebrauchten Zeit [KMT+17]

Bei Betrachtung der benötigten Zeit in Abbildung 3.1 für die Bearbeitung der verschiedenen Aufgaben fällt auf das CP bei den Aufgaben eins und fünf deutlich schlechter abschließt als VS. Khaloo et al. weisen bei den Ergebnissen in [KMT+17] darauf hin, dass CP Animationen besitzt wenn zwischen einem Code Raum und der Vogelperspektive gewechselt wird. So dauert jede Animation mindestens 1,5 Sekunden und ein Wechsel von einem Code Raum in einen anderen dauert damit mindestens 3 Sekunden. Ebenfalls wird Bemerkt dass sich die Teilnehmer:innen mehrfach sich von den visuellen Aspekten von CP die sehr einem Spiel gleichen ablenken ließen.

Damit lassen sich die kleineren Nachteile von CP bei Aufgabe zwei und drei gut er-

klären, für die großen Unterschiede bei den Aufgaben eins und fünf sind diese Faktoren allerdings nicht ausreichend. Die Unterschiede zwischen CP und VS können zusätzlich auch zu teilen davon kommen, dass VS die Möglichkeit bietet, eine Definition einer Klasse oder Variable mit wenigen Klicks aufzurufen welche CP nicht besitzt. Bei der Benutzung von CP kann deshalb das betreten vieler Code Räume notwendig werden um eine Definition zu finden, was aufgrund der Animationen viel zeit benötigt. Dies scheint davon bestätigt zu werden dass die größten Unterschiede bei T1 und T5 beobachtet wurde wo es notwendig sein kann viele Klassen zu durchsuchen. Bei den Bewertungen der Nutzer schneiden CP und VS hier beide gut ab wobei CP einen nennenswerten Vorteil gegenüber VS besitzt wenn es darum geht wie einfach es zu lernen ist und wie gut es sich dazu eignet sich mit einem neues Projekt vertraut zu machen.

Eine weitere Studie zu dem lernen neuer Projekte mithilfe von 3D Visualisierung haben Mehra et al. in [MSK+20] zu XRaSE einer durchgeführt. XRaSE ist eine Anwendung die ein Prototyp Programm das ein System in einer immersiven 3D Darstellung mit Hilfe von AR Geräten visualisieren soll. An der Studie nahmen 20 professionelle Softwareentwickler:innen teil. Für die Studie wurden zwei Open-Source Java Anwendungen verwendet – Pet Clinic[2] und Chaos Monkey[3]. Bei der Durchführung wurden die Teilnehmer:innen in zwei Gruppen aufgeteilt und nach einer Lernphase sollen fünf Aufgaben bearbeitet werden. Es wurde als erstes in 2D mit Eclipse und webbasierter node-link Graph Visualisierung oder in 3D mit XRaSE (je nach Gruppe) die Fragen zum ersten Projekt bearbeitet worauf dann die Visualisierungstechnik und das Projekt gewechselt wurden zur Beantwortung für das zweite Projekt.

Nummer	Aufgabe
A1	Welches Paket enthält Methode X?
A2	Welche Methode ist der beste Kandidat für eine Hotspot Methode?
A3	Welche Klasse enthält am wahrscheinlichsten einen großen Code smell?
A4	Von wie vielen Klassen ist Klasse X abhängig?
A5	Wie ist die ungefähre Größe des Projektes?

Tabelle 3.2: Aufgaben die bearbeitet werden mussten [MSK+20]

Die Tabelle 3.2 zeigt die Aufgaben die von den Teilnehmer:innen bearbeitet wurden, dabei wurde jeweils die benötigte Zeit gemessen.

Die Ergebnisse der Studie in Tabelle 3.3 zeigen das die Visualisierung in 3D besonders bei den Aufgaben eins bis drei große zeitliche Ersparnisse bringt sowie bei Aufgabe fünf eine fast identische Zeit benötigt. Nur für die vierte Aufgabe ist eine Bearbeitung in 2D signifikant schneller gewesen.

Werden beide Studien mit ihren Ergebnissen zusammen betrachtet, lässt sich die 3D Visualisierung für das Lernen neuer Projekte grundsätzlich empfehlen. Die 3D Werkzeuge können dabei Zeitersparnisse bieten und das Lernen vereinfachen. Allerdings ist

[2]https://github.com/spring-projects/spring-petclinic
[3]https://github.com/codecentric/chaos-monkey-spring-boot

	2D		3D	
Aufgabe	Mittelwert	Median	Mittelwert	Median
A1	116.3	94.0	61.5	62.5
A2	198.4	207.5	80.5	50.5
A3	81.4	72.5	47.6	45.5
A4	100.7	91.5	168.7	153.0
A5	43.4	40.0	42.6	40.0

Tabelle 3.3: Benötigte Zeit um die Aufgaben zu lösen [MSK+20]

die 3D Visualisierung nicht für alle Aufgaben in diesem Gebiet optimal geeignet und sie sollte deshalb zusammen mit einer herkömmlichen 2D IDE verwendet werden.

3.2 3D Visualisierung als zusätzliches Hilfsmittel in der Entwicklung

3D Visualisierungsumgebungen können ein wichtiges zusätzliches Hilfsmittel sein wenn eine gewöhnliche Visualisierung in 2D etwas nicht ausreichend darstellen kann. Mit **Sequence Debugging Session View** (SDV) stellen Fontana et al. in [FP19] ein 3D Visualisierungswerkzeug vor das Debugging Daten direkt aus einer IDE auslesen, Visualisieren und speichern kann. Das Werkzeug bietet somit die Möglichkeit für Entwickler:innen frühere Debugging Sitzungen erneut aufzurufen und auch mit neueren zu vergleichen und somit das Verständnis eines Projektes zu erhöhen. Um eine erste Evaluation zu bekommen wurde das Werkzeug einem Senior Entwickler:in zu Verfügung gestellt, welcher mit ihm einen Fehler untersuchen sollte. Ein:e Junior Entwickler:in bekam dann die gesammelten und sollte den Fehler beheben, danach wurde um Rückmeldung gebeten. Dabei kam heraus dass es für den Junior Entwickler:in durch die Möglichkeit eine Debugging Sitzung erneut abzuspielen deutlich leichter fiel den Fehler zu verstehen und zu lösen.

SDV zeigt das eine Kombination von 2D IDEs und 3D Visualisierung neue Möglichkeiten bringt die weder ausschließlich mit 2D oder 3D Anwendungen erreicht werden können. Daher ist eine Kombination aus 2D und 3D Werkzeugen zu empfehlen um die Stärken beider Technologien zu Nutzen.

4 Herausforderungen und Probleme

Von Tobias Braun

4.1 Vor und Nachteile unterschiedliche Visualisierungsverfahren

Es existieren viele verschiedene, frei verfügbare Visualisierungssysteme, um Eigenschaften von Quellcode in 3D Umgebungen zu Visualisieren. Hierbei werden, je nach gewähltem Einsatzszenario und Ziel, verschiedene Eigenschaften des Codes auf verschiedene Weise visualisiert. Dabei wird meist auf Metaphern zurückgegriffen, um die Abstrakten, gewählten Features und Metriken, wie die Komplexität von Klassen und Modulen, oder ihre Beziehung zueinander, in eine leicht verständliche Visuelle Form zu übertragen.

4.1.1 Metaphern

Für die Visuelle Form müssen die Features und Metriken in etwas, möglichst Intuitiv verständliches Übersetzt werden. Hier bieten sich Metaphern aus der Realität an, wie Städte [SKR19] oder der Innenraum von Gebäuden [HKI19]. Die Struktur des visualisierten Systems, wie Klassen, Module, Bibliotheken können in 3d Objekte gekapselt und Hierarchisch, oder durch Gruppierung dargestellte werden. Dabei können Metriken, wie Modulgröße, Datenmenge, durch Eigenschaften der Objekte, wie Größe, Farbe und Form [MSK+20] direkt erkennbar gemacht werden. Die Beziehungen von Modulen untereinander, lässt sich sowohl durch ihre Nähe zueinander, wobei man hier jedoch durch den 3d Raum beschränkt ist, und es bei einer großen Menge an Modulen zu Problemen mit der Sortierung und daraus resultierenden Positionierung zueinander kommen kann, als auch durch ihre Visualisierung in Farbe und Form [MRR19], wobei hier jedoch mit der Metapher gebrochen wird und daraus resultierend die dadurch gegebene Intuitivere Navigation und Übersicht eingeschränkt wird, darstellen [KMT+17].

4.1.2 Arten der Visualisierung

Da es eine Vielzahl an möglichen Metaphern gibt, wird hier nur eine Auswahl der populärsten besprochen. Jede dieser Visualisierungsarten eignet sich für spezifischen Probleme mehr und für andere weniger, was es erschwert sie auf der Qualitativen Ebene zu vergleichen. **Stadt Metapher** Eine populäre Metapher ist, das Softwaresystem als Stadt zu betrachten. Hierbei ist die Unterteilung und Verbindung der zu visualisierenden

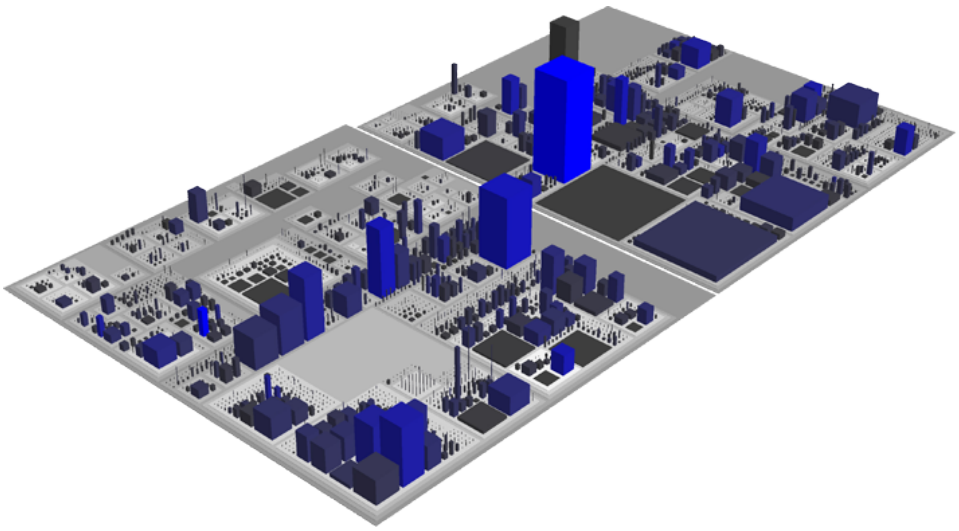

Abbildung 4.1: Visualisierung mit Code City [MBN18]

Elemente wichtig. So unterteilt Code City die Software in Distrikte, die Code Packages, Symbolisieren und diese enthalten wiederum Gebäude, die die einzelnen Klassen darstellen [MBN18], wohingegen bei Code Park die Gebäude ebenfalls Klassen Symbolisieren, jedoch nur Räumlich gruppiert und nicht in einen größeren Container eingefasst sind [KMT+17], wodurch hier eine wirkliche hierarchische Schachtelung fehlt.

Gebäude Metapher Die Software CodeHouse bedient sich wie in 2.1.1 beschrieben. Für eine bessere Übersicht sind die Räume hier wie in einem Panoptikum angeordnet, also im Kreis um den Nutzer, verteilt auf mehrere Stockwerke. Dies sorgt dafür, dass jedes Element, zu jedem Zeitpunkt, von der Mitte aus, sichtbar ist. Dabei reduziert es jedoch die Verteilung der Elemente auf zwei Dimensionen. Das Platzproblem, das bei der Stadtmetapher auftritt, die sich ebenfalls zwei Dimensionen bedient, bleibt.

Abstrakte Lösungen Visualisierungswerkzeuge wie CuboidMatrix stützen sich auf keine realitätsnahe Metapher, sondern stellen die Software als abstrakte Ansammlung von Würfeln, ohne zusätzliche Interpretationsebene dar. Sie stellt die Komponenten als simple Würfel im 3d Raum. Hierbei liegt der Fokus der Software auf der Visualisierung der Veränderungen von Metriken über einen Zeitraum und ist nicht speziell für die Code Visualisierung entwickelt worden, kann jedoch dazu eingesetzt werden, um ausgewählte Metriken zu Visualisieren [STSB16]. Sie ist jedoch auf diese beschränkt und kann keine weiteren Merkmale oder Bezüge darstellen.

4.2 Fokussetzung bei der Auswahl der Visualisierungsform

Jedes Werkzeug kann nur eine begrenzte Anzahl von Codemerkmalen und -metriken darstellen. Dadurch ist die Auswahl dieser, abhängig vom gewünschten Anwendungsbereich, zum Beispiel das schnelle Einlernen in eine neue Programmiersprache, oder

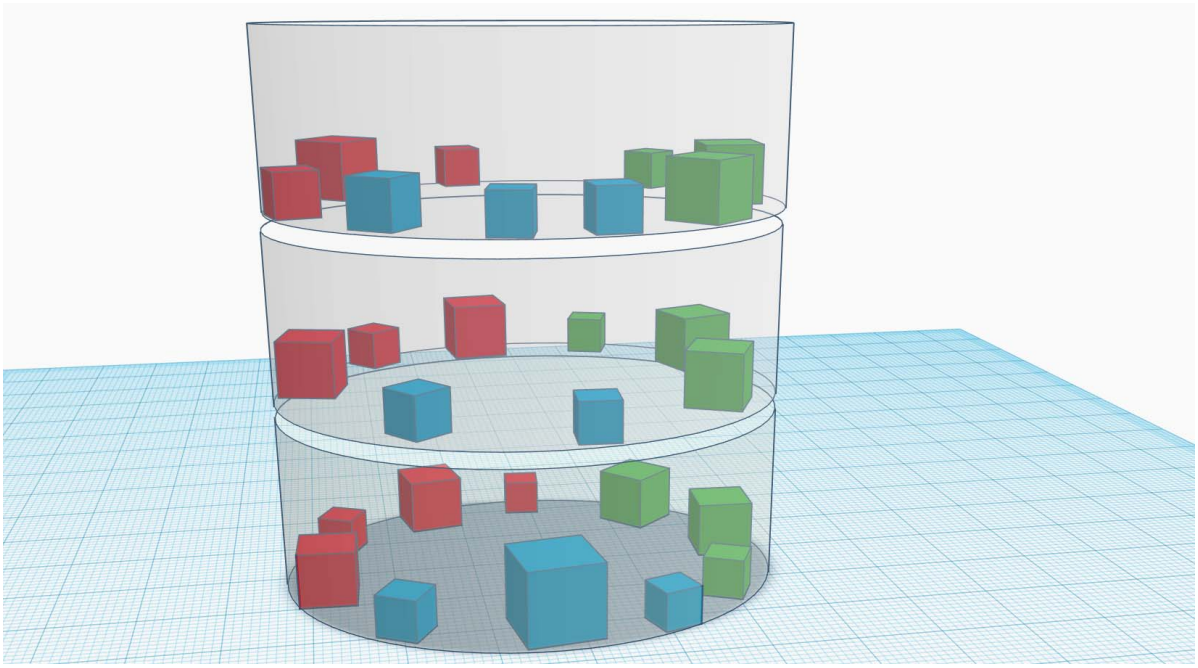

Abbildung 4.2: Visualisierung mit Code House [MSK+20]

das Verständlich machen von Kommunikationswegen in einem Software System wichtig. Dies wiederum beeinflusst die Wahl der Visualisierungsform, da diese, basierend auf der Art der Metrik gewählt werden sollten. Andere Metriken und Eigenschaften des Codes können meist nicht, oder nur unzureichend beachtet werden.

4.2.1 Ziele der Visualisierung

Die meisten Werkzeuge auf dem Markt Zielen auf einen ganz bestimmten Anwendungsbereich ab und haben ihre Metriken, Visualisierungsform und besonders die Metaphern danach gewählt.

Code Struktur Eine verbreitetes Einsatzgebiet ist die Erkundung von Software Strukturen. Werkzeuge wie Code Park [KMT+17], Code House [HKI19] oder Code City [MBN18], sollen den Nutzer:innen helfen, den Aufbau von komplexeren Softwaresystemen zu verstehen. Hierbei werden die Module der Software in einen Räumlichen Zusammenhang gebracht. Dies kann eingesetzt werden, um neue Entwickler schnell in bestehende Systeme einzuführen, oder beim Entwickeln einer neuen Software, die diese Ersetzen soll, um den Aufbau des Legacy Systems besser zu verstehen [HNS21].

Kommunikationswege 3D Visualisierung kann auch helfen die Kommunikationswege zwischen Komponenten eines komplexen Systems nach zu vollziehen und Abläufen, aber auch möglicher Schwachpunkte und Fehlerquellen zu Identifizieren. Wir als Metrik die Häufigkeit, mit der ein Pfad verwendet wird gewählt, kann die Visualisierung dabei helfen Flaschenhälse zu identifizieren [ZKS+08].

Duplikatsuche 3D Code Visualisierung lässt sich auch einsetzen, um Duplikate in

Quellcode zu finden. Hierzu können Metriken gewählt werden, die dazu führen, dass, zueinander ähnliche Software Module, ähnlich dargestellt werden. Durch diese Abstraktion Eben fällt es den Nutzer:innen leichter die Überschneidungen dieser Module, aufgrund Optischer und Struktureller Ähnlichkeit zu identifizieren [SKR19].

Edukativ Eine große Chance für 3d Visualisierungssysteme besteht auch im Edukativen Bereich. Es existiert bereits eine Vielzahl an Werkzeugen, die die Grundlagen der Softwareentwicklung in einer Spiele Ähnlichen 3d Umgebung vermitteln. Dabei bedienen sie sich meist auch weitere Elemente von Videospielen, wie Scoring Systemen, um die Nutzer:innen zu motivieren und ihren Fortschritt zu Bewerten [SKM13]. Bei dieser Form die Visualisierung ist es besonders Wichtig, gute Metaphern zu verwenden, da Nutzer:innen oft wenig, bis gar keine Vorkenntnis der Thematik haben und durch die Visualisierung die komplexen, gelernten Zusammenhänge mit bereits bekanntem verknüpfen können, um dieses Wissen später auf das Visualisierte System anwenden zu können. Es besteht die Gefahr, dass das Mentale Bild der Nutzer:in sich nur Unzureichend auf das Ursprüngliche System übertrage lässt [SKM13].

Debugging Auch bei der Fehlersuche können 3d Visualisierungswerkzeuge helfen. Während Programme über Zeit immer größer und komplexer werden, [MB21] steigt der Aufwand diese zu warten und zu erweitern. Werkzeuge, wie Sequence Debugging Session View (SDV), sollen hier helfen. SDV erzeugt interaktive 3D Diagramme, die die Abläufe innerhalb einer Software und die Kommunikation der einzelnen Komponenten untereinander darstellt. Dies kann dabei helfen Fehler schneller auf einen bestimmten Code Bereich einzugrenzen und Fehler in den Abläufen zu finden [FP19].

4.3 Orientierung und Navigation von Quellcode in 3D Umgebungen

Wie bei einem klassischen Integrated Development Enviroment (IDE), bieten auch 3D Code Visualisierungswerkzeuge verschiedene Möglichkeiten durch das betrachtete Softwaresystem zu Navigieren und diesen zu betrachten.

4.3.1 Kamera Positionierung

Im Gegensatz zu einer klassischen IDE erlauben 3D Visualisierungswerkzeuge eine Vielzahl an Möglichkeiten den Code zu betrachten. Dabei wird wie bei Computerspielen, auf eine Virtuelle Kamera zurückgegriffen, im Gegensatz zu klassischen IDEs, die nur einen starre 2D Betrachtung des Codes ermöglichen. Die Positionierung dieser Kamera sollte passend zur gewählten Visualisierung und Metapher passen. Visualisierungswerkzeuge die auf eine Stadtmetapher zurückgreifen wählen so gewöhnliche eine orthographische Ansicht, ähnlich zu Strategie Spielen. Diese erlaubt eine bessere Übersicht über große, komplexe Strukturen [SKR19]. Die Werkzeuge Code Park und Code House erlauben auch die Betrachtung aus einer First Person Perspektive, wie sie bei viele Computerspielen üblich ist, wobei diese bei Code Park [KMT+17] hauptsächlich für die direkte Betrachtung von Quellcode verwendet wird, ersteres, bietet jedoch auch eine,

der orthogonalen ähnliche Vogelperspektive. Code House bleibt durchgehend in einer First Person Perspektive [HKI19] wodurch zwar keine, möglicherweise desorientierenden Perspektivenwechsel nötig sind, jedoch die eine Perspektive, sowohl zur Nah, als auch Fern Betrachtung und Navigation genutzt wird und eine, dafür optimierte Perspektive fehlt [MSK+20].

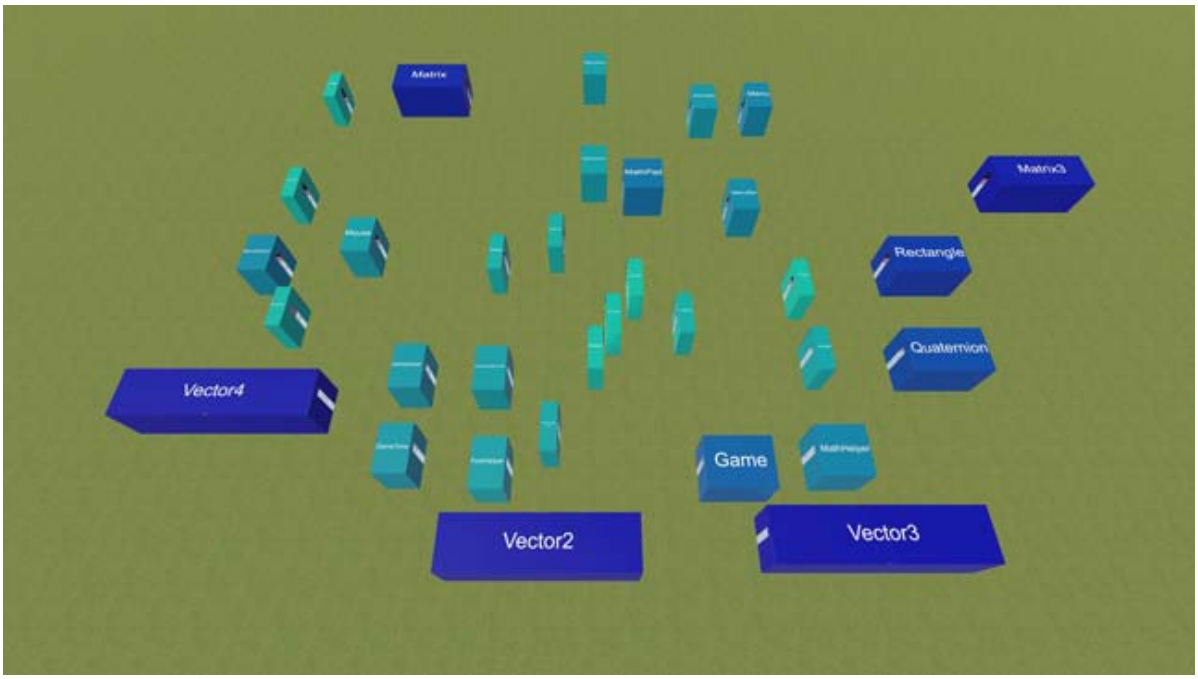

Abbildung 4.3: Vogelperspektive in Code Park [KMT+17]

4.3.2 Navigationsmöglichkeiten

Während klassische IDEs meist eine Vielzahl an Navigationsmöglichkeiten bieten, wie die Auswahl einer Quellcode Datei, der Sprung zur Definition eines Moduls, Klasse, oder Methode, oder eine Textbasierte Suche, bieten 3d Visualisierungswerkzeuge meist eine eingeschränkte, an die Metapher angepasste Auswahl an Navigationsmöglichkeiten. Werkzeuge die sich einer Stadt Metapher bedienen bieten für gewöhnlich mindestens eine Vogelperspektive, die einen guten Überblick über die gesamte Codebasis ermöglicht. [KMT+17] Die meisten Werkzeuge bieten auch eine freie Kamera, die es der Nutzer:in eine genauere Betrachtung einzelner Bereiche der Software erlaubt. Dabei greifen die Werkzeuge, die die direkte Betrachtung von Quellcode erlauben, dafür auf eine, den klassischen IDEs sehr ähnlichen, Textbasierte Code Visualisierung zurück.

5 Fazit

In dieser Arbeit wurden verschiedene Themenbereiche im Bezug zur Visualisierung von Quellcode in spieleähnlichen Umgebungen untersucht. Dabei wurde die Vor- und Nachteile der Visualisierung von Quellcode in Extended-Reality-Umgebungen unter der Nutzung von den Visualisierungsmethoden EvoStreets und CodeHouse erläutert, wobei der Fokus auf der Nutzung von immersiven Technologien lag. Die Evaluation ergibt, dass sich die beschriebenen Visualisierungsmethoden, unter der Nutzung von immersiven Technologien, für den Einsatz in der Praxis eignen. Dies geht aus der erhöhten Anregung der kognitiven Fähigkeiten des Nutzers hervor, ebenso wie der geförderten Interaktion bei der Nutzung dieser Technologien. Einziges Manko hierbei ist der erhöhte Aufwand bei der Inbetriebnahme von immersiven Technologien.

Zusätzlich wurden zwei mögliche Einsatzgebiete für die 3D Visualisierung, das finden von Duplikaten und das lernen eines neuen Projekts, betrachtet. Dabei wurden Ergebnisse von Studien verglichen um herauszufinden ob eine 3D Visualisierung eine Zeitersparnis bringen kann. Das Ergebnis dieses Vergleichs war dass die 3D Visualisierung keine signifikante Zeitersparnis bei dem finden von Duplikaten bringt und oft sogar langsamer war. Bei dem Lernen eines neuen Systems hingegen konnte die 3D Visualisierung oft eine bessere Zeit wie eine herkömmliche IDE vorweisen und wurde von den Teilnehmer:innen der Studien gut bewertet. Ebenfalls wurde noch die Möglichkeit betrachtet das Debugging mit einer 3D Umgebung zu erweitern, um vorherige Debuggingsitzungen zu speichern und erneut durchzugehen. Diese neue Möglichkeit war bei den Tester:innen ebenfalls beliebt und hat zu einem besseren Codeverständnis und einfacherem Finden von Fehlern geführt.

Bei der Visualisierung von Quellcode in 3D Umgebungen wird mit Metaphern gearbeitet, um Metriken des Quellcodes für den Nutzer besser Sichtbar und verständlich zu machen. Die Auswahl der Metapher bestimmt, wie und wie viele Metriken des Quellcodes dargestellt werden können. Da die Menge an Metriken, die sinnvoll gleichzeitig Visualisiert werden kann, limitiert ist, muss die Visualisierungsform anhand des zu lösenden Problems gewählt werden. Für verschiedene Aufgaben, wurden bereits Werkzeuge entwickelt, die mit eigenen, passenden Metaphern arbeiten, doch hat jede davon ihre eigenen, spezifischen Probleme, mit Übersicht, Navigation und Funktionsumfang. 3D Visualisierungswerkzeuge sind spezialisierter und daher eingeschränkte, als klassische IDEs und eignen sich nur bedingt, wenn überhaupt, als universelle Entwicklungsumgebung.

Literaturverzeichnis

[BW22] Herold R. Reiners D. Cruz-Neira C. Broll W., Grimm P. Virtual and augmented reality (vr/ar): Foundations and methods of extended realities (xr). In *Virtual and Augmented Reality (VR/AR): Foundations and Methods of Extended Realities (XR)*, Cham, 2022. Springer International Publishing.

[FP19] Eduardo A. Fontana and Fabio Petrillo. Visualizing sequences of debugging sessions using swarm debugging. In *2019 IEEE/ACM 27th International Conference on Program Comprehension (ICPC)*, pages 139–143, 2019.

[HKI19] Akihiro Hori, Masumi Kawakami, and Makoto Ichii. Codehouse: Vr code visualization tool. In *2019 Working Conference on Software Visualization (VISSOFT)*, pages 83–87, 2019.

[HNS21] Adrian Hoff, Michael Nieke, and Christoph Seidl. Towards immersive software archaeology: Regaining legacy systems' design knowledge via interactive exploration in virtual reality. In *Proceedings of the 29th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, ESEC/FSE 2021, page 1455–1458, New York, NY, USA, 2021. Association for Computing Machinery.

[JDGAGMGC05] G. Jimenez-Diaz, M. Gomez-Albarran, M.A. Gomez-Martin, and P.A. Gonzalez-Calero. Understanding object-oriented software through virtual role-play. In *Fifth IEEE International Conference on Advanced Learning Technologies (ICALT'05)*, pages 875–877, 2005.

[KMT+17] Pooya Khaloo, Mehran Maghoumi, Eugene M. Taranta, David Bettner, and Joseph J. LaViola. Code park: A new 3d code visualization tool. *2017 IEEE Working Conference on Software Visualization (VISSOFT)*, pages 43–53, 2017.

[MB21] Ali H. Mresa and Abdussalam Nuri Baryun. Assessing growth and instability of open source software systems. In *2021 IEEE 1st International Maghreb Meeting of the Conference on Sciences and Techniques of Automatic Control and Computer Engineering MI-STA*, pages 398–406, 2021.

[MBN18] Leonel Merino, Alexandre Bergel, and Oscar Nierstrasz. Overcoming issues of 3d software visualization through immersive augmented reality. In *2018 IEEE Working Conference on Software Visualization (VISSOFT)*, pages 54–64, 2018.

[MRR19] Steinbeck Marcel, Koschke Rainer, and Marc Rudel. Movement patterns and trajectories in three-dimensional software visualization. In *2019 19th International Working Conference on Source Code Analysis and Manipulation (SCAM)*, pages 163–174, 2019.

[MSK+20] Rohit Mehra, Vibhu Saujanya Sharma, Vikrant Kaulgud, Sanjay Podder, and Adam P. Burden. Towards immersive comprehension of software systems using augmented reality: An empirical evaluation. ASE '20, page 1267–1269, New York, NY, USA, 2020. Association for Computing Machinery.

[SKM13] Juha Sorva, Ville Karavirta, and Lauri Malmi. A review of generic program visualization systems for introductory programming education. *ACM Trans. Comput. Educ.*, 13(4), nov 2013.

[SKR19] Marcel Steinbeck, Rainer Koschke, and Marc O. Rüdel. Comparing the evostreets visualization technique in two d and three-dimensional environments: A controlled experiment. ICPC '19, page 231–242. IEEE Press, 2019.

[SMKP18] Vibhu Saujanya Sharma, Rohit Mehra, Vikrant Kaulgud, and Sanjay Podder. An immersive future for software engineering: Avenues and approaches. ICSE-NIER '18, page 105–108, New York, NY, USA, 2018. Association for Computing Machinery.

[STSB16] Teseo Schneider, Yuriy Tymchuk, Ronie Salgado, and Alexandre Bergel. Cuboidmatrix: Exploring dynamic structural connections in software components using space-time cube. In *2016 IEEE Working Conference on Software Visualization (VISSOFT)*, pages 116–125, 2016.

[ZKS+08] D. Zeckzer, R. Kalcklösch, L. Schröder, H. Hagen, and T. Klein. Analyzing the reliability of communication between software entities using a 3d visualization of clustered graphs. In *Proceedings of the 4th ACM Symposium on Software Visualization*, SoftVis '08, page 37–46, New York, NY, USA, 2008. Association for Computing Machinery.