

Fakultät für Informatik

## **Master Thesis**

CodeLeaves als neues Konzept der  
3D-Softwarevisualisierung und dessen Umsetzung für die  
Augmented Reality

Marcel Pütz  
2017



## ERKLÄRUNG

Ich versichere, dass ich diese Arbeit selbstständig angefertigt, nicht anderweitig für Prüfungszwecke vorgelegt, keine anderen als die angegebenen Quellen oder Hilfsmittel benutzt sowie wörtliche und sinngemäße Zitate als solche gekennzeichnet habe.

Rosenheim, den 26. November 2017

Marcel Pütz



## Kurzfassung

Visualisierung spielt in vielen Bereichen der Wissenschaft eine wichtige Rolle. In der Informatik ist die Visualisierung von Software bei der Größe und Komplexität zeitgemäßer Software-Systeme eine hilfreiche Unterstützung für deren Entwicklung, Exploration und Analyse.

Mit dem zunehmend Einzug haltenden Medium der Augmented Reality bekommt die Softwarevisualisierung neue Möglichkeiten und wird mit 3D-Modellen für den Nutzer greifbar.

CodeLeaves stellt ein neuartiges Konzept vor, mit dem eine Software mithilfe der Wald-Metapher zu dreidimensionalen Bäumen wird. Klassen werden zu Blättern und Pakete zu Ästen und ganzen Bäumen. Durch die vorhandene Baumstruktur in der Software ist mit CodeLeaves der geistige Transfer zwischen Realität und Abstraktion besonders gering.

Das Blätterdach des Waldes kann von einem sommerlichen Grün über Gelb bis hin zu einem herbstlichen Rot eine beliebige Metrik darstellen. Verbindungen in einer Software, seien es statische Abhängigkeiten oder dynamische Aufrufe, können übersichtlich auf die Baumstruktur aggregiert und interaktiv exploriert werden.

Dafür wird ein sprachunabhängiges Software-Meta-Modell entworfen, mit dem es möglich ist, Informationen der statischen und dynamischen Softwarevisualisierung darzustellen.

In einem Prototyp für die HoloLens, einer Augmented-Reality-Brille von Microsoft, wird die Generierung eines Waldes mit realistischen Beispieldaten erprobt und die dafür benötigten Layout-Algorithmen entwickelt. Dabei werden in der Natur vorkommende Verteilungsstrategien und hierarchisches Circle-Packing zur Platzierung der Elemente verwendet.

Die Interaktion mit dem Wald wird unter Einbeziehung von Herausforderungen in der Augmented Reality teils in dem Prototypen für die HoloLens implementiert und teils konzeptionell weiter ausgearbeitet.

**Schlagworte:** Konzept, 3D, Softwarevisualisierung, Wald, Metapher, Baum, Augmented Reality, AR, Brille, Software-Meta-Modell, Statik, Qualitäts-Metriken, Abhängigkeiten, Dynamik, Aufrufe, Laufzeiten, 3D Layout-Algorithmen, Sonnenblumen-Algorithmus, Circle-Packing, Interaktion, Reaktive Programmierung



# Inhaltsverzeichnis

<b>1 Einleitung</b>	<b>1</b>
1.1 Einführung in die AR . . . . .	1
1.2 Motivation für eine Softwarevisualisierung in der AR . . . . .	3
1.3 Zielsetzung der Arbeit . . . . .	3
1.4 Aufbau der Arbeit . . . . .	4
1.5 Abgrenzung . . . . .	5
<b>2 Das Konzept CodeLeaves</b>	<b>7</b>
2.1 CodeLeaves und die Metapher Software-Wald . . . . .	7
2.2 Begriffsklärung . . . . .	10
2.3 Vergleich mit anderen Konzepten . . . . .	12
2.4 Anforderungen an CodeLeaves . . . . .	16
<b>3 Architektur und Datenmodellierung</b>	<b>23</b>
3.1 Trennung in Schichten . . . . .	23
3.2 Software-Meta-Modell . . . . .	24
3.3 Internes Datenmodell . . . . .	27
3.4 UI-Datenmodell . . . . .	27
<b>4 Modellierung und Layout des Waldes</b>	<b>31</b>
4.1 Grundlegender Ansatz . . . . .	31
4.2 Verteilung von Blättern mit dem Sonnenblumen-Algorithmus . . . . .	35
4.3 Verteilung innerer Knoten mit Circle-Packing . . . . .	38
4.4 Verwendete Algorithmen in der Praxis . . . . .	43
4.5 Abbildung von Daten auf die Struktur . . . . .	45
<b>5 Grundlagen der Interaktion in der AR</b>	<b>49</b>
5.1 Eingabemöglichkeiten in der AR und mit der HoloLens . . . . .	49
5.2 Herausforderungen . . . . .	51
5.3 Technische Umsetzung von Interaktionen mit reaktiver UI . . . . .	53
<b>6 Interaktion mit CodeLeaves</b>	<b>57</b>
6.1 Basisinteraktionen . . . . .	57
6.2 Das Kontextmenü . . . . .	59
6.3 Manipulation-Indicators . . . . .	61
6.4 Das App-Menü . . . . .	61
6.5 Interaktion mit Blättern, inneren Knoten und Wurzeln . . . . .	67

<b>7 Zusammenfassung und Ausblick</b>	<b>73</b>
7.1 Zusammenfassung . . . . .	73
7.2 Ausblick . . . . .	74
<b>A Anhang</b>	<b>77</b>
<b>Literatur</b>	<b>79</b>

# Abbildungsverzeichnis

1.1	Das Reality-Virtuality-Kontinuum (nach [29]) . . . . .	2
2.1	Vorteil der Dreidimensionalität bei der Darstellung von Wurzeln . . . . .	10
2.2	Begriffsklärung . . . . .	12
2.3	Informationen über Software, an deren Visualisierung Mitarbeiter der QA-ware Interesse haben . . . . .	13
2.4	CodeLeaves und alternative Ansätze der 3D-Softwarevisualisierung . . . . .	15
3.1	Schichtentrennung im Datenmodell . . . . .	24
3.2	Meta-Modell einer Software . . . . .	26
3.3	Internes Datenmodell . . . . .	28
3.4	UI-Datenmodell . . . . .	29
4.1	3D-Objekte für den High-Fi-Prototyp . . . . .	32
4.2	Fractal-Tree [39] . . . . .	34
4.3	Spiralförmige Anordnung der Röhrenblüten einer Sonnenblume [17] . . . . .	35
4.4	Kugelkoordinatensystem mit den zu berechnenden Größen für das Hinzufügen eines neuen Blattes . . . . .	37
4.5	Sonnenblumen-Algorithmus zur Verteilung von Blättern . . . . .	38
4.6	Wang's Circle Packing Algorithmus (nach [4]) . . . . .	40
4.7	Berechnung des Mittelpunktes eines Kreises, tangential zu zwei anderen Kreisen . . . . .	41
4.8	Layout des Waldes . . . . .	44
4.9	Farbe der Blätter und Erkennung von Ausreißern . . . . .	46
4.10	Unterschiedliche Dicke der Kanten . . . . .	47
4.11	Beispiel einer Wurzel . . . . .	47
5.1	Air-Tap [5] . . . . .	50
5.2	Verdeckung von UI-Elementen . . . . .	52
5.3	Shine-through-Effect von UI-Elementen in HoloStudio [20] . . . . .	53
5.4	Beispielhafte Interaktion mit einem Blatt . . . . .	54
5.5	Datenstrom eines Labels für reaktive Programmierung . . . . .	55
6.1	Bounding-Box als Option für die Basisinteraktionen [10] . . . . .	58
6.2	Spatial Mapping beim Platzieren des Waldes . . . . .	60
6.3	Kontextmenü für den Waldboden im High-Fi-Prototyp . . . . .	60
6.4	Manipulation-Indicators am Beispiel des Interaktions-Modus Skalieren . . . . .	61
6.5	High-Fi-Prototyp der Projektauswahl im App-Menüs . . . . .	63
6.6	Low-Fi-Prototyp der Projektauswahl im App-Menü . . . . .	64

6.7	Low-Fi-Prototyp für die Einstellungen im App-Menü . . . . .	65
6.8	Visualisierung der Kreise für leichtere Interaktion . . . . .	66
6.9	Hervorhebung und Anzeige der Anzahl und Richtung von einzelnen Verbindungen einer Wurzel . . . . .	70
6.10	Kontextmenü für ein Blatt . . . . .	71
A.1	Low-Fi Prototyp für den Import eines neuen Projekts anhand des Beispiels SonarQube . . . . .	77
A.2	Kontextmenü für einen inneren Knoten . . . . .	78
A.3	Kontextmenü für eine Wurzel . . . . .	78

# **Tabellenverzeichnis**

2.1 Akzeptanzkriterien zu User-Story 1 . . . . .	17
2.2 Akzeptanzkriterien zu User-Story 2 . . . . .	17
2.3 Akzeptanzkriterien zu User-Story 3 . . . . .	18
2.4 Akzeptanzkriterien zu User-Story 4 . . . . .	19
2.5 Akzeptanzkriterien zu User-Story 5 . . . . .	20
2.6 Akzeptanzkriterien zu User-Story 6 . . . . .	20
2.7 Akzeptanzkriterien zu User-Story 7 . . . . .	21
4.1 Eckdaten des Beispielprojektes Air . . . . .	33
6.1 Mögliche Aktionen und deren Erreichbarkeit . . . . .	68



# 1 Einleitung

„Virtual reality was once the dream of science fiction. But the internet was also once a dream, and so were computers and smartphones. The future is coming.“

---

Mark Zuckerberg  
Facebook CEO, 2014

„I think AR is [...] big, it's huge. I get excited because of the things that could be done that could improve a lot of lives.“

---

Tim Cook  
Apple CEO, 2017

## 1.1 Einführung in die AR

Viele der einflussreichsten Technologieunternehmen arbeiten an der *Virtual* bzw. *Augmented Reality* (VR bzw. AR). Tim Cook ist überzeugt davon, dass die AR die nächste “big idea” nach dem Smartphone wird [34].

Nicht nur Großkonzerne wie Apple, Facebook oder Samsung arbeiten intensiv in diesem Bereich. Ein Start-up-Unternehmen namens *Magic Leap* entwickelt eine AR-Brille und wird von Investoren in Milliardenhöhe unterstützt [25]. Laut einem Cover-Artikel der Zeitschrift *Wire* ist die noch unter Verschluss gehaltene Technologie allen Konkurrenzprodukten weit voraus. Das Release der Hardware ist Stand heute noch nicht bekannt, aber es lässt sich ein Trend erkennen, der die nächsten Jahre viele neue Möglichkeiten eröffnen und möglicherweise die Digitalisierung revolutionieren wird.

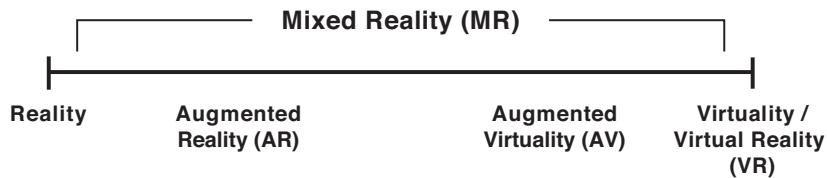
Diese Arbeit wird sich mit einer Anwendung der AR beschäftigen – der Softwarevisualisierung. Die Spezialisierung auf AR kann nach der Abgrenzung von AR zur VR besser nachvollzogen werden.

**VR** ist eine Umgebung, in der der Betrachter vollkommen von einer computergenerierten Welt umgeben ist, die oft die reale Welt imitiert, aber auch rein fiktiv sein kann [29].

## 1 Einleitung

Obwohl der Begriff AR zunehmend in der Industrie Verwendung findet, entbehrt er doch einer einheitlichen Definition. In [2] wird AR als „*Variation*“ von VR betrachtet. Dagegen vermittelt P. Milgram in [29] ein vollständigeres Verständnis, weshalb sich die Begrifflichkeiten in dieser Arbeit daran anlehnen sollen. Nach Milgram existieren die beiden entgegengesetzten Extreme der Realität und der Virtualität. Alles dazwischen ist die sogenannte *Mixed Reality (MR)*.

**MR** ist eine Umgebung, in der Elemente der realen und einer virtuellen Welt zusammen dargestellt werden [24].



**Abbildung 1.1** Das Reality-Virtuality-Kontinuum (nach [29])

Dieses *Reality-Virtuality-Kontinuum* ist in Abbildung 1.1 dargestellt, in dem gut zu erkennen ist, dass AR zu der Mixed Reality gehört. In den meisten Quellen wie [2, 1, 24] wird bei der AR noch die Komponente der Interaktion aufgeführt. AR kann deshalb folgendermaßen definiert werden:

### Definition 1.1: AR

AR ist die Erweiterung der realen Welt durch computergenerierte Elemente, mit denen der Betrachter in Echtzeit interagieren kann.

Auch die *Augmented Virtuality*, also die Erweiterung der virtuellen Welt durch reale Elemente, gehört zur MR.

Wie viele der einflussreichsten Menschen der Technologie-Industrie, sieht Tim Cook mehr Zukunft in der AR. Er begründet das in einem Interview damit, dass diese Technologie die wirkliche Welt nicht ausschließt, sondern die Realität erweitert und Teil von zwischenmenschlicher Kommunikation sein kann [34].

Stellen wir uns ein Hologramm vor, das auf einem Konferenztisch Gestalt annimmt und eine Software visualisiert. Entwickler, Projektleiter oder auch Kunden versammeln sich um den Tisch und können miteinander interaktiv die Software betrachten, evaluieren und gesuchte Informationen daraus ziehen.

Dies wäre mit VR nicht möglich, da der Betrachter von der Außenwelt abgeschottet ist. Deshalb wird im Zuge dieser Arbeit mit der momentan ausgereiftesten Technologie der AR gearbeitet – der *HoloLens* von Microsoft.

## 1.2 Motivation für eine Softwarevisualisierung in der AR

Die Technologie der AR bietet uns viele neue Möglichkeiten. Eine Motivation dieser Arbeit ist es, sich produktiv mit einer neuen, zukunftsträchtigen Technologie zu beschäftigen. Das ist jedoch nur die eine Seite. Die weitaus größere Motivation ist, die zuvor noch nicht dagewesene Zugänglichkeit und Interaktion mit dreidimensionaler Visualisierung auszunutzen. Visualisierung im Allgemeinen begegnet uns in vielen Bereichen unseres Lebens und nimmt eine wichtige Rolle ein.

Niemand konnte bislang unser Sonnensystem von außen betrachten. Dennoch haben wir alle eine ziemlich gute Vorstellung wie dieses aufgebaut ist. Durch die Visualisierung der Planeten und der Sonne entsteht in uns ein geistiges Abbild der Realität. Das Konzept komplexe Realitäten zu abstrahieren und zu visualisieren, um dadurch die Realität besser verstehen zu können, ist in vielen Disziplinen der Wissenschaft vertreten.

Neben den Naturwissenschaften wie Physik, Chemie oder Biologie, nimmt Visualisierung auch in der Informatik eine wichtige Rolle ein. In vielen Bereichen müssen Informationen in eine visuelle Form gebracht werden, die für einen menschlichen Betrachter leichter zu verstehen und zu interpretieren ist.

Gerade bei komplexen Software-Systemen ist das der Fall. Soll zum Beispiel die zu Grunde liegende Struktur einer Software Außenstehenden erklärt werden, gelingt das mit einem visuellen Modell wie einem UML-Diagramm sicherlich besser, als sich direkt mit dem Source-Code auseinander zu setzen.

Angefangen bei UML-Diagrammen waren die meisten Darstellungsformen der Softwarevisualisierung bislang meist zweidimensional. Mit AR wird dieser Disziplin der Visualisierung jedoch wortwörtlich ein neuer Raum an Möglichkeiten eröffnet und diese Arbeit soll damit beginnen, den Raum zu füllen.

## 1.3 Zielsetzung der Arbeit

Für die Zielsetzung einer 3D-Softwarevisualisierung in der AR sollten zunächst die allgemeinen Ziele einer Softwarevisualisierung betrachtet werden. Softwarevisualisierung ist für Diehl die „visualization of artifacts related to software and its development process“ [14]. Diese Definition ist sehr breit gefasst. Wird der Fokus mehr auf die Nutzerorientierung gelegt, lässt sich Softwarevisualisierung wie folgt definieren:

### Definition 1.2: Softwarevisualisierung

Softwarevisualisierung ist die bildliche oder auch metaphorische Darstellung einer Software, um dem Nutzer durch Vereinfachung und Abstraktion das bessere Verständnis oder die einfachere Analyse von Software zu ermöglichen.

In dieser Arbeit soll eine solche Softwarevisualisierung für die AR vorgestellt werden.

## 1 Einleitung

Das neuartige Konzept *CodeLeaves* bedient sich der Metapher des *Software-Waldes* und soll vorgestellt und im Detail ausgearbeitet werden.

Im Vorfeld dieser Arbeit wurde mittels einer Studie in [35] erkannt, dass eine gute Softwarevisualisierung sowohl statische als auch dynamische Daten unterstützen sollte. Mit *CodeLeaves* soll dies möglich sein. Beliebige Daten sollen übersichtlich auf die Struktur der Software abgebildet werden können.

Durch Expertengespräche sollen User-Storys erstellt werden, um den Mehrwert des neuen Konzepts zu veranschaulichen und für weitere Überlegungen Anforderungen zu definieren.

Um diesen gewünschten Anforderungen gerecht zu werden, soll für *CodeLeaves* ein sprachunabhängiges Datenmodell entworfen werden, das alle geforderten Informationen unterstützt.

Der praktische Teil dieser Arbeit soll die prototypische Entwicklung von *CodeLeaves* sein. Dabei sollen zwei Schwerpunkte gesetzt werden: Zum einen soll die Generierung des Waldes und die dafür benötigten Layout-Algorithmen erarbeitet und für die HoloLens umgesetzt werden. Zum anderen soll ein Interaktions-Konzept ausgearbeitet werden, mit dem *CodeLeaves* bedienbar wird und eine beliebige, objektorientierte Software bis ins Detail erforscht werden kann.

### 1.4 Aufbau der Arbeit

Im nächsten Kapitel wird das Konzept von *CodeLeaves* zunächst auf High-Level-Ebene vorgestellt, um es anschließend detaillierter zu erörtern. Es werden Begriffe definiert, die für weitere Inhalte ein einheitliches Verständnis schaffen. Nachdem das Konzept geklärt wurde, werden Unterschiede und Vorteile von *CodeLeaves* zu alternativen 3D-Softwarevisualisierungen hervorgehoben. Die Befragung von Experten der Softwareanalyse und die daraus abgeleiteten Anforderungen an *CodeLeaves* schließen das erste Kapitel ab.

Das Kapitel 3 beschäftigt sich mit der Entwicklung einer geeigneten Architektur und eines Datenmodells für *CodeLeaves*. Nachdem vorhandene Datenmodelle auf Tauglichkeit für *CodeLeaves* überprüft und für nicht geeignet eingestuft wurden, wird ein neues, in Schichten getrenntes Datenmodell entwickelt.

Mit diesem Datenmodell kann der Wald für *CodeLeaves* generiert werden, was Thema des 4. Kapitels ist. In diesem Zuge werden zwei Algorithmen und die verwendete Mathematik vorgestellt, die eine möglichst natürliche und übersichtliche Struktur des Waldes produzieren. Abbildungen der HoloLens-Applikation zeigen die praktische Anwendung der theoretischen Inhalte.

Kapitel 5 behandelt Grundlagen der Interaktion in der AR, die bei der Interaktion mit *CodeLeaves* Anwendung finden. Es wird auch beschrieben, wie mit reaktiver Programmierung die Interaktion technisch sinnvoll umgesetzt werden kann.

Die Interaktion mit *CodeLeaves* ist Thema des Kapitels 6. Es wird gezeigt, wie *CodeLeaves* mit den zur Verfügung stehenden Eingabemöglichkeiten bis ins Detail exploriert werden kann.

## *1.5 Abgrenzung*

Das Kapitel 7 fasst die Ergebnisse der Arbeit zusammen und bietet eine Ausblick auf weiterführende Tätigkeiten.

### **1.5 Abgrenzung**

Die Beschaffung der Informationen über eine Software wird hier nur am Rande betrachtet. Für die Analyse einer Software müssen statische und dynamische Informationen gesammelt und aggregiert werden. Hierfür gibt es verschiedenste proprietäre sowie Open-Source-Tools. Die Anbindung an CodeLeaves würde den Rahmen dieser Arbeit jedoch sprengen. In dieser Arbeit wird aber der Anspruch gestellt, dass mit realistischen Beispieldaten gearbeitet wird, um die Umsetzbarkeit von CodeLeaves validieren zu können.

Weiterhin wird im Zuge dieser Arbeit mit Prototypen gearbeitet. Diese sind kein fertiges Produkt der Softwarevisualisierung, sondern dienen vielmehr als technischer Durchstich für die HoloLens und als Basis für eine weiterführende Entwicklung.

Die Validierung der Vorteile von CodeLeaves gegenüber alternativen Visualisierungen ist ebenfalls nicht Teil dieser Arbeit. Vielmehr könnten dafür Testnutzer im Rahmen einer Studie mit der Weiterentwicklung des Prototyps und realen Software-Systemen sowohl mit CodeLeaves als auch mit seinen Alternativen gestellte Aufgaben bewältigen. Im Vergleich von Benutzerfreundlichkeit und Effizienz kann dann überprüft werden, inwieweit CodeLeaves seinen Alternativen überlegen ist.



## 2 Das Konzept CodeLeaves

### 2.1 CodeLeaves und die Metapher Software-Wald

“Hierarchies are almost ubiquitous [...]” [38] halten Robertson *et al.* schon 1991 bei der Visualisierung von hierarchischen Informationen fest. So verhält es sich auch bei der Struktur einer Software. Jede Software mit einer geschachtelten Struktur ist hierarchisch und kann mithilfe eines Baumes dargestellt werden. „*Bäume sind eine der wichtigsten Datenstrukturen, die besonders im Zusammenhang mit hierarchischen Abhängigkeiten und Beziehungen zwischen Daten von Vorteil sind.*“ [16] stellen auch Ernst *et al.* fest.

Der Baum in der Informatik zeugt von einer ursprünglichen Metapher – dem Baum, wie er draußen in der Natur wächst. Da dieser unbestritten dreidimensional ist, ist eine Softwarevisualisierung für die Dreidimensionalität mit einer realitätsnahen Interpretation der Baum-Metapher naheliegend. Werden Bäume in 3D in natürlicher Wuchsrichtung modelliert und mehrere Bäume für eine Software verwendet, entsteht eine neue Metapher: der *Software-Wald*.

CodeLeaves stellt ein Konzept dar, das sich die Metapher des Software-Waldes zu nutze macht und wird im Folgenden auf High-Level-Ebene skizziert und danach genauer erläutert:

#### Konzept: CodeLeaves

1. Die Struktur der Software wird mit nach oben wachsenden Bäumen dargestellt.
2. Pakete im Hauptverzeichnis werden als Bäume, Unterpakete als Äste und Klassen als Blätter dargestellt.
3. Die Blätter der Bäume können durch ihre Farbe eine beliebige Metrik<sup>1</sup> visualisieren.
4. Verbindungen<sup>2</sup> zwischen einzelnen Klassen werden aggregiert auf die Struktur der Bäume als Dicke der Kanten abgebildet oder interaktiv als direkte Spinnweben dargestellt.
5. Zwischen den Bäumen entsteht ein Geflecht aus Wurzeln, was aggregierte Verbindungen zwischen den Bäumen darstellt.

---

<sup>1</sup>siehe Definition 2.2

<sup>2</sup>siehe Definition 2.3

## 2 Das Konzept CodeLeaves

**Punkt 1** ist dafür verantwortlich, dass der geistige Transfer zwischen realer Software und deren Visualisierung möglichst gering ist. Das bedeutet auch, dass diejenigen, die tatsächlich mit der Software arbeiten, sich aufgrund der bekannten Baumstruktur auch in der Visualisierung schnell zurecht finden. Im Fachjargon wird hier von der *Habitability* gesprochen, also wie schnell oder gut sich ein Betrachter in einer Software oder auch deren Visualisierung „zuhause“ fühlt [48].

**Punkt 2** beschreibt die Überführung der Struktur einer Software in einen Wald. Dabei sollen Klassen im Weiteren beispielhaft als die kleinsten betrachteten *Softwareartefakte* angesehen werden.

### Definition 2.1: Softwareartefakt

Ein Softwareartefakt ist eine definierte Einheit einer Software.

Bei feinerer Granularität könnten Klassen aber auch mit Methoden oder sogar Code-Blöcken ersetzt werden. Für Softwareartefakte, die aus anderen Softwareartefakten bestehen, einigen wir uns auf die Bezeichnung Paket. Auch hier gilt, dass die Organisation von Software genauso in Module oder Komponenten aufgebaut sein kann, solange sie hierarchisch ist.

Die Darstellung einzelner Bäume für die Pakete im Hauptverzeichnis einer Software bietet sich an. Dieser Ansatz schließt jedoch nicht aus, dass Pakete aus einer tieferen Hierarchie-Ebene als neuer Waldboden verwendet werden. Interaktionen dieser Art sind Teil des Kapitels 6.

**Punkt 3** bedeutet, dass in CodeLeaves durch die Farbe der Blätter eine gute Übersicht über eine ausgewählte Metrik der Software entsteht.

### Definition 2.2: Metrik

Eine Metrik ist eine Eigenschaft, die für jede Klasse gemessen und mit einem eindeutigen Zahlenwert festgelegt werden kann. Ein Beispiel dafür ist die Testabdeckung von Klassen.

Beispielsweise kann der Farbe der Blätter die Testabdeckung der einzelnen Klassen zugewiesen werden. Mit einer Skala von Grün bis Rot kann dann der Software-Wald bei guter Testabdeckung im sommerlichen Grün erstrahlen, oder bei einer weniger guten Testabdeckung eher in den Herbst übergehen. Die Färbung der Blätter ist flexibel auf jegliche Metrik anwendbar, die für die betrachtete Software zur Verfügung steht.

**Punkt 4** heißt, dass die Verbindungen einer Software sehr übersichtlich auf die Struktur der Software abgebildet werden können, ohne diese negativ zu beeinflussen. Eine

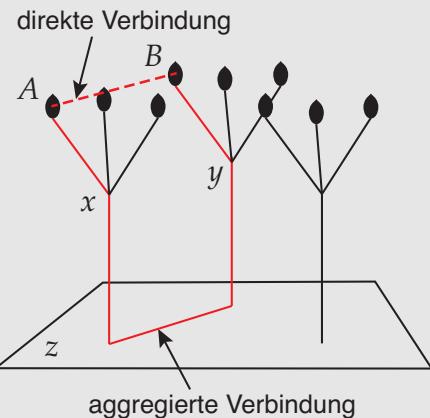
Verbindung in einer Software und die Aggregation einer solchen wird im Folgenden für das bessere Verständnis definiert.

**Definition 2.3:** Verbindung

Unter einer Verbindung in einer Software verstehen wir eine gerichtete Beziehung eines bestimmten Typs zwischen zwei Klassen. Eine Klasse kann mehrere ein- und ausgehende Verbindungen besitzen. Ein Beispiel für ein Verbindungs-Typ sind statische Abhängigkeiten von Klassen.

**Definition 2.4:** Aggregation von Verbindungen

Bei der Aggregation von Verbindungen zwischen Klassen werden die Verbindungen nicht direkt dargestellt, sondern über die Eltern geleitet. Seien  $x, y$  Pakete und Klasse  $A \in x$  besitze eine Verbindung zu Klasse  $B \in y$ , dann geht die Verbindung von  $A$  zunächst zu  $x$ . Angenommen,  $x$  und  $y$  befinden sich zudem im Paket  $z$ , dann geht die aggregierte Verbindung weiter über  $z$  zu  $y$  und schließlich zu  $B$ .



Auf der rechten Seite der Definition 2.4 ist eine aggregierte Verbindung am Beispiel von CodeLeaves zu sehen.

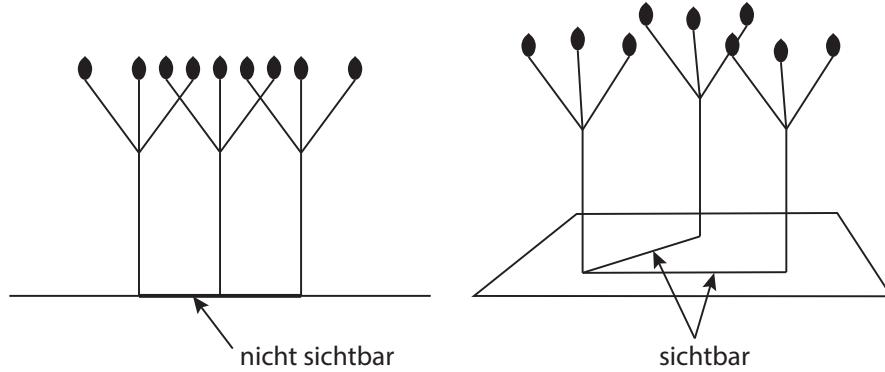
Bei mehreren Verbindungen zwischen benachbarten Paketen überlagern sich die aggregierten Abhängigkeiten zwangsläufig. Falls im Beispiel von 2.4 eine weitere Abhängigkeit von einer anderen Klasse in  $x$  zu einer anderen Klasse in  $y$  bestünde, würden sich die aggregierten Verbindungen von  $x$  bis  $y$  überlagern. Daraus resultiert, dass nicht jede aggregierte Verbindung ohne Interaktion zwangsläufig eindeutig zuzuordnen ist.

Wird aber bei jedem Verbindungsstück, sei es eine Kante oder Wurzel, die Anzahl an überlagernden Verbindungen als Dicke des Verbindungsstücks dargestellt, bekommt der Betrachter eine gute Übersicht über die Gesamtheit der Verbindungen. Durch Interaktion mit einzelnen Verbindungsstücken soll eine feingranuläre Analyse der Verbindungen möglich sein.

Das zweite Element von Punkt 4 sind Spinnweben. Damit lassen sich die Verbindungen direkt darstellen. Um bei großen Software-Systemen aber die Übersicht über den Wald nicht zu verlieren, sollten diese mithilfe von Interaktion flexibel aktivierbar sein.

## 2 Das Konzept CodeLeaves

**Punkt 5** bietet einen großen Vorteil gegenüber der Zweidimensionalität. In Abbildung 2.1 wird rechts das Prinzip des Wurzelgeflechts mit einem minimalistischen Beispiel illustriert.



**Abbildung 2.1** Vorteil der Dreidimensionalität bei der Darstellung von Wurzeln

In 3D sind die Verbindungen zwischen den Bäumen auf einen Blick ersichtlich. Bei Betrachtung der Bäume in 2D, wie es im linken Teil der Abbildung dargestellt ist, verschwinden die Verbindungen hintereinander und sind nicht zuzuordnen.

Nachdem das grundlegende Konzept von CodeLeaves verstanden ist, muss eine Begriffsklärung ein einheitliches Verständnis der verwendeten Elemente schaffen. Es wurden bereits Elemente des Waldes intuitiv verwendet, müssen aber für die Verwendung im Datenmodell im nächsten Kapitel sowie in den entwickelten Layout-Algorithmen in Kapitel 4 genau definiert werden.

## 2.2 Begriffsklärung

Die Bezeichnungen, die im Folgenden definiert werden, bauen stark auf die Datenstruktur eines Baumes in der Informatik auf – „*the most important nonlinear structures that arise in computer algorithms*“ [26] – die in einschlägiger Literatur wie [26, 16, 23] definiert wird.

Da sich aber nicht alle Begriffe der Datenstruktur auf die Elemente von CodeLeaves eins-zu-eins übertragen lassen, passen wir einige Begriffe der Datenstruktur an die Gegebenheiten von CodeLeaves an und führen neue Begriffe ein.

- Ein *Baum* (engl.: *tree*) besteht aus einer Menge von *Knoten*, die so durch *Kanten* so verbunden sind, dass keine Kreise auftreten (abgewandelt aus [23, 16]).
- Ein *Knoten* (engl.: *node*) beinhaltet Informationen zu sich selbst und hat 0 bis  $n$  *Kinder* (engl.: *children*).
- Ein Knoten mit keinen Kindern wird als *Blatt* (engl.: *leaf*) bezeichnet. Alle anderen Knoten heißen *innere Knoten* (engl.: *inner nodes*) [23].
- Kinder des selben Knotens werden *Geschwister* (engl.: *siblings*) genannt.

## 2.2 Begriffsklärung

- Alle Kinder und Kindeskinder eines Knoten werden *Nachfahren* (engl.: *descendants*) des Knotens genannt.

Bei der klassischen Definition eines Baumes wird bei dem Knoten ohne Elternteil von der „Wurzel“ gesprochen. Im Modell von CodeLeaves ist der unterste Knoten eines Baumes jedoch noch durch eine Kante mit dem Waldboden verbunden. Darüber hinaus überschneidet sich die Bezeichnung von „Wurzel“ mit den Verbindungen zwischen den einzelnen Bäumen, die bei der klassischen Definition nicht existieren. Deshalb wird für diesen speziellen Knoten eine neue Bezeichnung eingeführt.

- Der unterste Knoten eines Baumes wird als *Kronenansatz* (engl.: *crown base*) bezeichnet.
- Alle Knoten bis auf den Kronenansatz haben genau einen Knoten als *Elternteil* (auch Vorfahre genannt, engl.: *parent* oder *ancestor*).

Nachdem in den meisten Fällen Bäume in 2D nach unten wachsend dargestellt werden, was wahrscheinlich auf die Tatsache zurück zu führen ist, dass handschriftliche Diagramme tendenziell nach unten wachsend gezeichnet werden [26], wird bei Knoten auch oft von einer Tiefe gesprochen. Diese Bezeichnung wird beibehalten.

- Die *Tiefe* eines Knotens gibt an, wie viele Kanten er vom Kronenansatz aus entfernt liegt (abgewandelt aus [16]).
- Die *Höhe* (engl.: *height*) eines Knoten beschreibt die maximale Tiefe aller Nachfahren.
- Alle Knoten mit der gleichen Höhe befinden sich auf der selben *Ebene* (engl.: *level*).

Alle nachfolgenden Kanten und Knoten eines Knotens  $A$  werden in der Literatur unterschiedlich bezeichnet. Es ist die Rede von „Teilbaum“ [16] oder auch „Unterbäumen“ [23]. Wir definieren dafür einen dem 3D-Modell besser entsprechenden Begriff.

- Ein *Ast* (engl.: *branch*) ist die Menge aller Nachfahren eines Knotens und die dazugehörigen Kanten.

Bisher wurden Bezeichnungen aus Literatur verwendet oder abgeändert. Betrachten wir nun Elemente von CodeLeaves, die so nicht in der klassischen Definition eines Baumes vorkommen.

- Der Kronenansatz ist durch die Kante namens *Stamm* (engl.: *trunk*) mit dem *Waldboden* (engl.: *forest floor*) verbunden.
- Der Schnittpunkt zwischen Stamm und Waldboden nennen wir *Stammbasis* (engl.: *trunk base*). Dieser ist jedoch kein Knoten und beinhaltet auch keine Informationen.
- Ein Wald mit  $n$  Bäumen besitzt 0 bis  $n!$  *Wurzeln* (engl. *roots*), die jeweils zwei Stammbasen miteinander verbinden.
- Ein *Wald* (engl.: *forest*) besteht aus einer disjunkten Menge an Bäumen, Wurzeln und dem Waldboden.

## 2 Das Konzept CodeLeaves

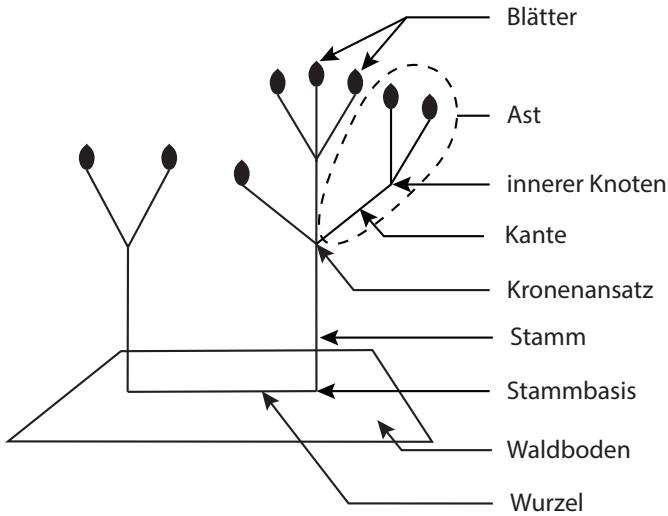


Abbildung 2.2 Begriffsklärung

### 2.3 Vergleich mit anderen Konzepten

Im Vorfeld dieser Arbeit wurde in [35] evaluiert, was eine gute Softwarevisualisierung ausmacht und unterstützen sollte. Ausgehend davon wurden vorhandene und neue Konzepte der 3D-Softwarevisualisierungen miteinander verglichen. Die Ergebnisse dieser Untersuchung sollen im Folgenden vorgestellt werden.

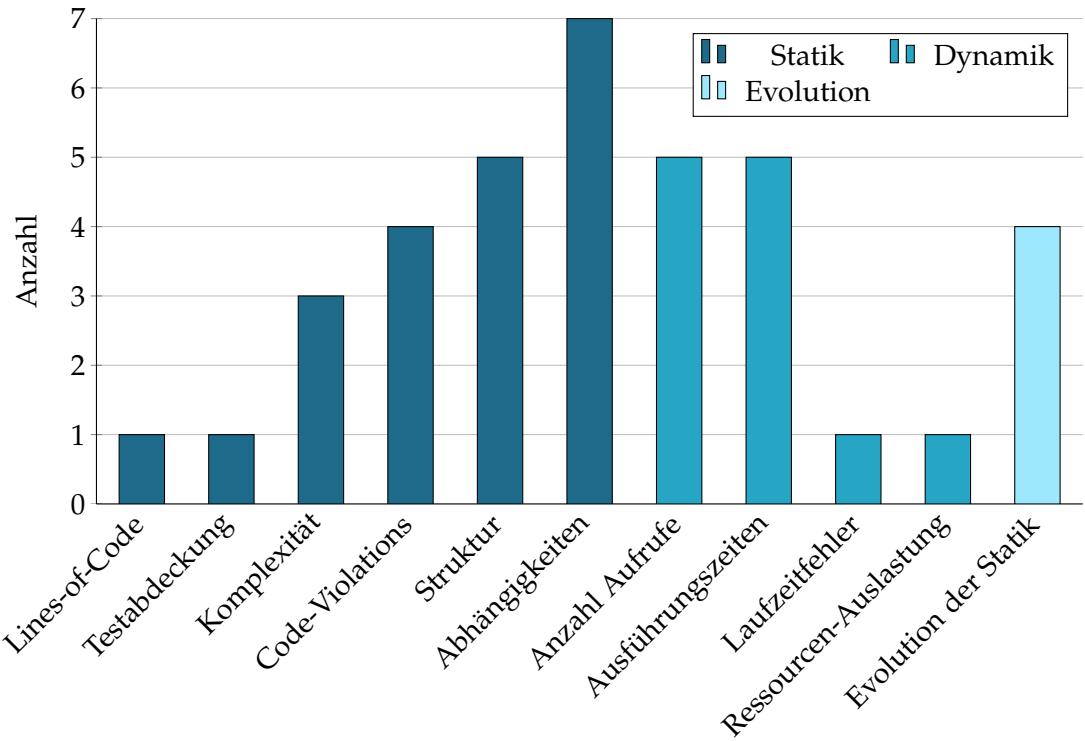
Um herauszufinden, welchen Mehrwert sich Nutzer einer Softwarevisualisierung von dieser versprechen, wurde eine Umfrage in der QAware GbmH (im Weiteren mit QAware bezeichnet) durchgeführt. Die QAware ist ein Projekthaus mit den Kerngeschäften Diagnose, Sanierung, Exploration und Realisierung von Software [36]. Durch die Erfahrung in Projekten für namhafte Kunden zeichnen sich die Mitarbeiter durch fundiertes Wissen und Expertise aus. Es wurden insgesamt 22 Mitarbeiter mit unterschiedlichen Rollen im Software-Engineering befragt.

In Abbildung 2.3 ist zu sehen, wie oft welche Informationen genannt wurden. Alle Informationen lassen sich in Diel's Kategorien der Softwarevisualisierung [14] *Statik*, *Dynamik* und *Evolution* einordnen und sind in Abbildung 2.3 entsprechend farbig gruppiert.

**Statik** sind die Informationen, die ohne die Ausführung der Software generiert werden können [14]. Als Metriken wurden Lines-of-Code (LOC), Komplexität, Testabdeckung und Code-Violations genannt. Letzteres sind Verletzungen von vereinbarten *Code-Conventions*. Komplizierter zu visualisieren sind Struktur einer Software und Abhängigkeiten darin. Aus Abbildung 2.3 geht hervor, dass diese Informationen gleichzeitig am meisten von Interesse sind.

**Dynamik** beschreibt die Informationen, die zur Laufzeit einer Software generiert werden können [14]. Besonders oft wurden Ausführungszeiten und Anzahl von Aufrufen

### 2.3 Vergleich mit anderen Konzepten



**Abbildung 2.3** Informationen über Software, an deren Visualisierung Mitarbeiter der QAware Interesse haben

genannt. Mit der Kombination aus beiden Informationen können *Bottlenecks* identifiziert werden. Auch die Visualisierung der Laufzeitfehler einer fehlerhaften Software ist für deren Analyse von Vorteil. Die Ressourcen-Auslastung ist dabei ebenfalls hilfreich, wirkt sich jedoch wenig auf das 3D-Modell der Softwarevisualisierung aus, da diese Informationen parallel zur eigentlichen Software existieren.

**Evolution** beschreibt den zeitlichen Verlauf einer Software und stellt den Entwicklungsprozess in den Vordergrund [14]. Beispielsweise kann die Entwicklung statischer Metriken verfolgt werden.

Aus den von den Mitarbeitern gewünschten Informationen und weiteren Rahmenbedingungen wurden in [35] folgende Kriterien aufgestellt, anhand derer vier am vielversprechendsten erscheinende Modelle der 3D-Softwarevisualisierung bewertet wurden.

- Statische Metriken (z. B. Komplexität)
- Struktur
- Abhängigkeiten
- Dynamik (primär Ausführungszeiten und Anzahl der Aufrufe)
- Evolution

## 2 Das Konzept CodeLeaves

- Habitability (vgl. Kapitel 2 Punkt 1)
- Drilldown
- Technische Machbarkeit

Bei dem Kriterium Drilldown wurde bewertet, wie gut eine Visualisierung ihre Informationen von High-Level bis hin zu Details darstellen kann.

Bei der technischen Machbarkeit wurde berücksichtigt, ob eine existierende Softwarevisualisierung für die HoloLens verwendbar ist. In Abbildung 2.4 sind die untersuchten Alternativen abgebildet, darunter auch ein erster Entwurf von CodeLeaves.

### CodeCity

2007 stellten R. Wettel *et al.* *CodeCity* vor, die mithilfe der *Stadt-Metapher* dreidimensionale Städte visualisiert. Darin werden Klassen als Gebäude und Pakete als Stadtviertel dargestellt [48, 49, 50]. Für die Breite und Tiefe der Gebäude wurde für die Anzahl der Attribute (engl. *number of attributes (NOA)*) und für die Höhe die Anzahl der Methoden (engl. *number of methods (NOM)*) einer Klasse gewählt.

Die CodeCity ist als Konzept sehr durchdacht, bietet durch die Metapher gute Habitability und unterstützt die Darstellung der Evolution. Auch soll nach Wettel *et al.* die CodeCity die Analyse von Software im Vergleich zu herkömmlichen Analyse-Werkzeugen signifikant verbessern [50].

Jedoch unterstützt CodeCity keine Dynamik und die Abhängigkeiten sind nur als direkte Verbindungen darstellbar, was bei größeren Software-Systemen sehr unübersichtlich wird. Die verfügbaren statischen Metriken sind begrenzt und vor allem ist die Technologie Stand heute nicht mehr produktiv einsetzbar [35].

### SoftVis3D

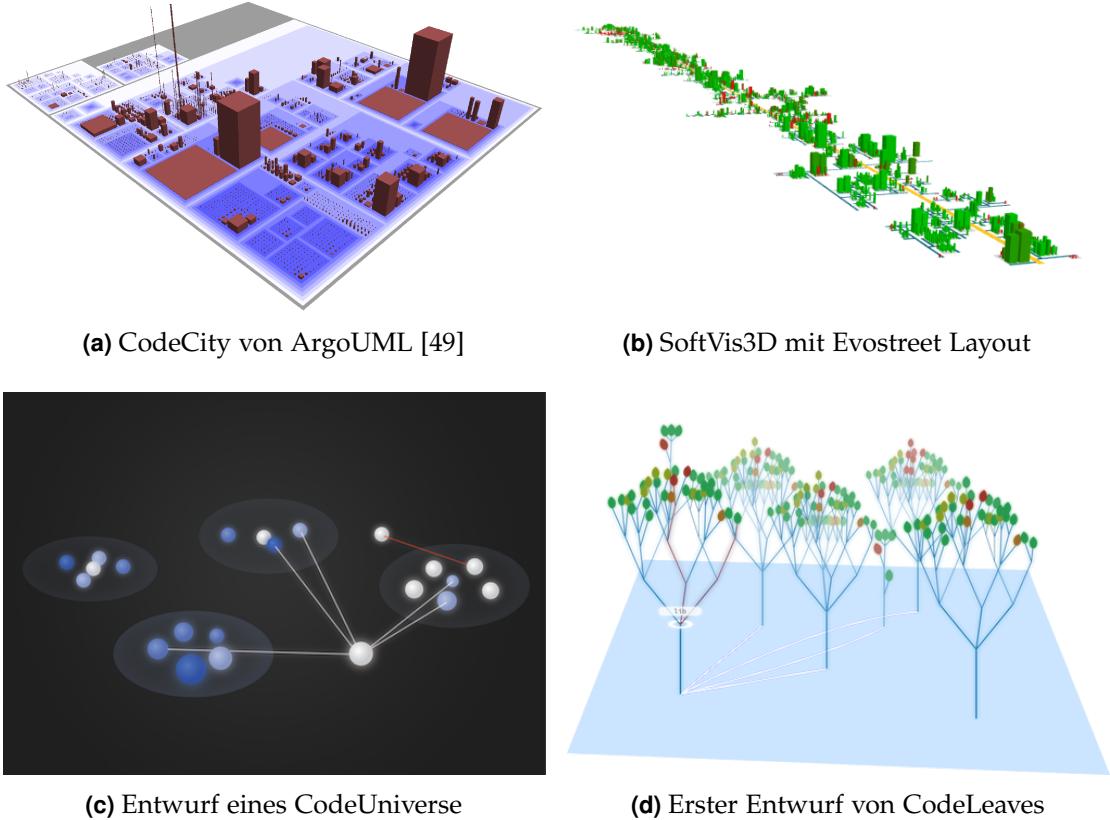
*SoftVis3D* greift das Konzept der CodeCity auf und visualisiert als Plugin für SonarQube<sup>3</sup> Projekte direkt im Browser. Durch die direkte Anbindung an SonarQube ist SoftVis3D hoch konfigurierbar und kann alle Metriken darstellen, die auch in SonarQube zur Verfügung stehen. Neben dem *District-Layout*, wie es in der CodeCity verwendet wird, unterstützt SoftVis3D darüber hinaus auch des *Evostreet-Layout*, das ursprünglich in [41] für die Evolution einer Software entworfen wurde. In diesem Layout, wie es in Abbildung 2.4b zu sehen ist, werden Pakete als sich verzweigende Straßen dargestellt. In Abbildung 2.4b fällt auf, dass die Städte mit vielen Geschwistern im root-Verzeichnis durch das Evo-Street-Layout sehr in die Länge gezogen werden, was sich negativ auf die Übersichtlichkeit auswirkt.

Die Evolution wird von SoftVis3D trotz EvoStreet-Layout nicht unterstützt. Abhängigkeiten konnten in früheren Versionen aggregiert dargestellt werden. Dafür wurden Pakete im Distrikt-Layout in übereinander liegenden Ebenen abgebildet und die aggregierten Abhängigkeiten über ein zusätzliches Hilfsgebäude zu der darüberliegenden Ebene geleitet. Dadurch ging jedoch die Stadt-Metapher und die Übersichtlichkeit

---

<sup>3</sup>SonarQube ist eine open-source Plattform für statische Code-Qualität, <https://www.sonarqube.org/>

## 2.3 Vergleich mit anderen Konzepten



**Abbildung 2.4** CodeLeaves und alternative Ansätze der 3D-Softwarevisualisierung

verloren. Die Dynamik kann in SoftVis3D ebenfalls nicht visualisiert werden. Die verwendete Technologie ist zwar mit *WebGL* für den Browser State-of-the-Art, aber für die HoloLens aktuell noch nicht sinnvoll einsetzbar [35].

### CodeUniverse

Im Zuge der Studie [35] wurde eine weitere Metapher evaluiert. Ähnlich wie in den Arbeiten [22, 3] wird die Software als Universum dargestellt. Klassen gruppieren sich als Sterne in Galaxien. Statische Metriken können dann als Farbe und Größe der Sterne widergespiegelt werden. So können „weiße Zwerge“ bis hin zu „roten Riesen“ entstehen.

Das *CodeUniverse* ist für statische Metriken gut geeignet. Auch die Evolution ist mit der Entstehung von neuen Sternen und Galaxien gut vorstellbar. Die Struktur der Software ist zwar mit der Gruppierung der Sterne gegeben, aber weniger offensichtlich wie in anderen Konzepten. Bei der Visualisierung von Abhängigkeiten und dynamischen Aufrufen stößt das *CodeUniverse* aber an seine Grenzen. Durch direkte Verbindungen zwischen den Sternen ist dies zwar möglich, aber bei großen Software-Systemen würde das schnell im Chaos enden. Auch in [3] wird beschrieben, dass eine übersichtliche Darstellung von Abhängigkeiten nur durch deren Aggregation erreicht werden kann.

## 2 Das Konzept CodeLeaves

Deshalb wird in [3] ein Konzept entworfen, dass die Klassen mit einem „*hierarchischem Netz*“ verbindet. Dieses ist nichts anderes als die vorhandene Baumstruktur der Software und hat mit der Metapher des Universums nichts mehr zu tun. Folglich wären wir wieder bei dem neuen Konzept CodeLeaves angelangt.

### Vorteile von CodeLeaves gegenüber anderen 3D-Softwarevisualisierungen

Die betrachteten Alternativen und andere haben gemein, dass sie Struktur, Dynamik und Evolution nicht vereinen. Darüber hinaus können Verbindungen zwischen Klassen nicht ohne Verlust der Übersichtlichkeit angezeigt werden. CodeLeaves soll alle drei Kategorien der Softwarevisualisierung unterstützen und ist bei der Visualisierung von Struktur und Verbindungen überlegen. Durch die Baumstruktur, wie sie auch in der Software vorhanden ist, wird die Struktur eins-zu-eins wiedergegeben. Die aggregierten Abhängigkeiten lehnen sich an die Struktur an und beeinflussen diese nicht negativ. Zusätzlich zur dreidimensionalen Struktur der Bäume wird durch das Wurzelgeflecht und die Spinnweben die Dreidimensionalität sehr gut ausgenutzt.

## 2.4 Anforderungen an CodeLeaves

Die Umfrage, die in Abschnitt 2.3 vorgestellt wurde, ergab einen guten Stimmungsbarometer für die Wünsche im Bezug auf zu visualisierende Informationen. In diesem Abschnitt soll genauer auf die Anforderungen an das CodeLeaves-Konzept eingegangen werden. Dazu wurde es zwei Experten der Softwareanalyse vorgestellt und diese befragt, was sie von CodeLeaves erwarten würden. Durch die Expertengespräche wurden User-Storys gesammelt und Akzeptanzkriterien abgeleitet.

Zur dynamischen Analyse wurde der promovierende Performance-Analyst F. Lautenschlager interviewt. Dieser hat im Zuge seiner Dissertation eine hochperformante Zeitreihendatenbank zur Speicherung und Auswertung von dynamischen Daten einer Software erarbeitet. Er ist führend auf diesem Gebiet und ist unter anderem als Referent auf internationalen Konferenzen geladen.

Als Experte der statischen Analyse wurde J. Weigend interviewt. Dieser ist „Chefarchitekt, Geschäftsführer und Mitgründer der QAware. Er studierte Informatik mit Schwerpunkt ‚Verteilte Systeme‘ an der Hochschule Rosenheim und hält dort seit 2001 Vorlesungen.“ [37]

Die Expertengespräche werden unter Anwendung der Methoden aus [8] in User-Storys 1 - 5 zusammengefasst. Tabellen 2.1 – 2.5 beinhalten die abgeleiteten Akzeptanzkriterien.

### User-Story 1: Code-Qualität (Statik)

Als *Softwarearchitekt*  
möchte ich *Code-Qualität auf die Struktur abbilden können,*  
*um Anomalien und Qualitätsdefizite zu erkennen.*

## 2.4 Anforderungen an CodeLeaves

**Tabelle 2.1** Akzeptanzkriterien zu User-Story 1

ID	Beschreibung
1.1	Die Farbe der Blätter kann eine beliebige Metrik darstellen.
1.2	Ich kann Ausreißer der ausgewählten Metrik gut erkennen.
1.3	Ich möchte möglichst viele Blätter gleichzeitig betrachten können.
1.4	Ich möchte die Blätter aus allen Winkeln gut erkennen können.

Die Blätter in CodeLeaves bieten eine hervorragende Umsetzungsmöglichkeit von User-Story 1. Sie können jegliche Metrik darstellen, seien es Zahlenwerte in Prozent oder andere Werte. Mit dem Minimum und Maximum einer ausgewählten Metrik lässt sich ein Farbverlauf festlegen, mit dem die Metrik für jede Klasse gleichzeitig betrachtet werden kann und Ausreißer aus der Menge hervorstechen.

Für diese User-Story ist es besonders wichtig, ein Layout für den Wald zu entwickeln, das die Blätter so verteilt, dass sie sich so wenig wie möglich überlappen. Dieser Ansatz fließt in Kapitel 4 mit ein.

**User-Story 2:** Kopplung und Struktur (Statik)

Als *Softwarearchitekt*  
möchte ich *die Kopplung und Struktur einer großen Software*  
*überblicken können,*  
*um die Zusammenhänge zu verstehen.*

**Tabelle 2.2** Akzeptanzkriterien zu User-Story 2

ID	Beschreibung
2.1	Die Struktur der Bäume spiegelt die tatsächliche Struktur der Software wider und ist gut zu erkennen.
2.2	Ich verstehe die Metapher des Software-Waldes intuitiv.
2.3	Die Dicke einer Kante oder Wurzel zeigt mir die Anzahl der durchfließenden Verbindungen des ausgewählten Verbindungs-Typs.

Mit den Baumstrukturen und der Dicke der Kanten und Wurzeln ist User-Story 2 mit CodeLeaves besonders gut umsetzbar. In der AR kann der Nutzer um den Wald herum gehen, sodass er die Struktur mit den aggregierten Verbindungen aus jedem Winkel betrachten kann.

Jedoch setzen Akzeptanzkriterien 2.1 - 2.2 bereits viel voraus. Aus beliebig geschach-

## 2 Das Konzept CodeLeaves

telter Software sollen Bäume entstehen, die die Struktur korrekt wiedergeben und gleichzeitig übersichtlich sind. Überschneidungen sind zu vermeiden, da sonst die Struktur nicht eindeutig zu erkennen ist. Gleichzeitig sollen die Bäume stets natürlich wirken, was das intuitive Verständnis der Metapher fördert. Diesen Herausforderungen stellen sich die Algorithmen, die in Kapitel 4 erarbeitet werden.

Akzeptanzkriterium 2.3 erfordert die Unterstützung von aggregierten Verbindungen, was im Datenmodell berücksichtigt wird.

### User-Story 3: Intuitive Erforschung (Statik)

Als *Softwarearchitekt*  
möchte ich *den Wald intuitiv erforschen können,*  
*um detaillierte Informationen zu erhalten.*

**Tabelle 2.3** Akzeptanzkriterien zu User-Story 3

ID	Beschreibung
3.1	Ich kann Elemente im Wald identifizieren.
3.2	Ich kann mir die genaue Zahl der aktuellen Metrik für ein Blatt anzeigen lassen.
3.3	Wenn ich ein Blatt auswähle, dann kann ich die ein- bzw. ausgehenden Verbindungen hervorheben.
3.4	Wenn ich eine Kante oder Wurzel auswähle, dann kann ich sehen, wie viele Verbindungen in welche Richtung fließen.
3.5	Wenn ich eine Kante oder Wurzel auswähle, dann kann ich die von den Verbindungen betroffenen Blättern hervorheben.
3.6	Ich kann einen Knoten als neuen Waldboden festlegen.
3.7	Ich kann den Zeitpunkt auswählen, auf die sich die dargestellten Daten beziehen.

- 
- 3.1 Ich kann Elemente im Wald identifizieren.
  - 3.2 Ich kann mir die genaue Zahl der aktuellen Metrik für ein Blatt anzeigen lassen.
  - 3.3 Wenn ich ein Blatt auswähle, dann kann ich die ein- bzw. ausgehenden Verbindungen hervorheben.
  - 3.4 Wenn ich eine Kante oder Wurzel auswähle, dann kann ich sehen, wie viele Verbindungen in welche Richtung fließen.
  - 3.5 Wenn ich eine Kante oder Wurzel auswähle, dann kann ich die von den Verbindungen betroffenen Blättern hervorheben.
  - 3.6 Ich kann einen Knoten als neuen Waldboden festlegen.
  - 3.7 Ich kann den Zeitpunkt auswählen, auf die sich die dargestellten Daten beziehen.
- 

User-Story 3 ist mit der AR sehr gut umsetzbar. Durch die AR ist der Nutzer mitten in der Visualisierung, die greifbar wird. Die Akzeptanzkriterien 3.1 - 3.6 liefern für die Interaktion mit CodeLeaves eine wertvolle Basis und werden in Kapitel 6 aufgegriffen.

### User-Story 4: Dynamische Daten

Als *Performance-Analyst*  
möchte ich *Funktionsaufrufe, deren Antwortzeiten, Laufzeitfehler und Ressourcen-Auslastung visualisiert haben,*  
*um das Laufzeitverhalten der Software in einem bestimmten*

*Zeitraum explorativ bewerten zu können.*

**Tabelle 2.4** Akzeptanzkriterien zu User-Story 4

ID	Beschreibung
4.1	Die Farbe der Blätter kann eine beliebige Metrik darstellen.
4.2	Ich kann Ausreißer der ausgewählten Metrik gut erkennen.
4.3	Die Dicke einer Kante oder Wurzel zeigt mir die Anzahl der durchfließenden Verbindungen des ausgewählten Verbindungs-Typs.
4.4	Ich kann mir die genaue Zahl der aktuellen Metrik für ein bestimmtes Blatt anzeigen lassen.
4.5	Wenn ich ein Blatt auswähle, dann kann ich die ein- bzw. ausgehenden Verbindungen hervorheben.
4.6	Wenn ich eine Kante oder Wurzel auswähle, dann kann ich die von den Verbindungen betroffenen Blättern hervorheben.
4.7	Wenn ich eine Wurzel oder Kante auswähle, dann kann ich sehen, wie viele Verbindungen in welche Richtung gehen.
4.8	Ich kann den Zeitraum auswählen, auf die sich die dargestellten Daten beziehen.
4.9	Ich kann zusätzliche Informationen über den Wald als Ganzes visualisieren.

Durch User-Story 4 wird deutlich, dass sich viele gleiche Akzeptanzkriterien aus statischen und dynamischen User-Storys ableiten lassen. Daran ist zu erkennen, wie sinnvoll es ist, diese beiden Sichten miteinander zu kombinieren. Ob es sich um statische oder dynamische Daten handelt, spielt für die Visualisierung großteils keine Rolle. Es ändert sich lediglich der betrachtete Zeitpunkt bei statischen Daten zu einem Zeitraum für dynamische Daten. Die Ressourcen-Auslastung sind zusätzliche Informationen, die den Wald als Ganzes betreffen und werden im Datenmodell berücksichtigt.

**User-Story 5:** Zustand der Software (Dynamik)

Als *Systemverantwortlicher*  
möchte ich *auf einen Blick den Zustand meiner Software erkennen,*  
*um bei Bedarf agieren zu können.*

## 2 Das Konzept CodeLeaves

**Tabelle 2.5** Akzeptanzkriterien zu User-Story 5

ID	Beschreibung
5.1	Wenn ich die Laubfarbe des Waldes betrachte, dann möchte ich eine möglichst gute Gesamtübersicht über die ausgewählte Metrik haben.
5.2	Ich möchte möglichst viele Blätter gleichzeitig betrachten können.

User-Story 5 weist wiederum Ähnlichkeiten zu User-Story 1 auf. Die Laubfarbe ist auch für den Systemzustand sehr gut geeignet, da sie eine gute Übersicht liefert, aber gleichzeitig jede Klasse einzeln anzeigt. Die Metapher des Software-Waldes kommt auch hier besonders gut zur Geltung, da auf einen Blick ersichtlich wird, ob es der Software gut geht (sommerlicher Wald) oder schlecht (herbstlicher Wald).

### User-Storys zur Benutzbarkeit von CodeLeaves

Aus anfänglichem Prototyping mit in der AR sind weitere User-Story entstanden, ohne die die anderen erst gar nicht möglich sind. Deshalb werden User-Storys 6 und 7 und deren Akzeptanzkriterien in Tabellen 2.6 und 2.7 im Folgenden beschrieben.

**User-Story 6:** Anpassung im Raum (Nutzbarkeit)

Als Nutzer in der AR  
möchte ich *den Wald meinen Räumlichkeiten und Wünschen anpassen können,*  
*um den Wald sinnvoll nutzen zu können.*

**Tabelle 2.6** Akzeptanzkriterien zu User-Story 6

ID	Beschreibung
6.1	Ich kann den Wald im Raum platzieren.
6.2	Ich kann den Wald skalieren.
6.3	Ich kann den Wald rotieren.

Vor allem am Anfang der Nutzung von CodeLeaves ist es wichtig, den Wald dorthin zu platzieren, wo es am meisten Sinn macht. Ist der Wald initial von realen Objekten verdeckt, kann der Nutzer nicht damit arbeiten. Für den Wechsel zwischen Übersicht und der Betrachtung von Details ist es essentiell, die Größe des Waldes jederzeit einfach anpassen zu können. Die Rotation des Waldes kann komfortabler sein, als um den ganzen Wald herum zu laufen.

Vor der Platzierung des Waldes ist noch ein weiterer Schritt erforderlich. Es muss als

## 2.4 Anforderungen an CodeLeaves

erstes ein Projekt ausgewählt werden können, das dargestellt werden soll. Die letzte User-Story beschreibt deshalb die Auswahl bzw. den Import von Projekten und Tabelle 2.7 deren Akzeptanzkriterien.

### User-Story 7: Auswahl eines Projekts (Nutzbarkeit)

Als Nutzer  
möchte ich *ein bestehendes Projekt öffnen und neue importieren können,*  
*um das Projekt zu betrachten, das mich interessiert.*

**Tabelle 2.7** Akzeptanzkriterien zu User-Story 7

ID	Beschreibung
7.1	Ich kann bereits gespeicherte Projekte öffnen.
7.2	Ich kann ein neues Projekt aus unterstützten Datenquellen importieren.
7.3	Ich kann Daten zu einem bereits gespeicherten Projekt aus unterstützten Daten hinzufügen.

Akzeptanzkriterium 7.1 ist für einen vollständigen technischen Durchstich wichtig, da so in der HoloLens Applikation ein Beispielprojekt geöffnet werden kann.

User-Storys 1 - 5 sind auf High-Level-Ebene formuliert. Viele der Akzeptanzkriterien finden jedoch schon im Prototyping Anwendung. Im weiteren Verlauf der Arbeit wird deshalb bei der Adressierung eines Akzeptanzkriteriums in folgender Form darauf verwiesen: (Akz. <ID>).

Im nächsten Kapitel wird ein Datenmodell entworfen, mit dem alle vorgestellten User-Storys umgesetzt werden können.



# 3 Architektur und Datenmodellierung

## 3.1 Trennung in Schichten

Die Datenmodellierung für CodeLeaves nimmt einen zentralen Baustein ein im Prototyp für die HoloLens ein. Bei dem Entwurfsmuster wird ein wichtiges Prinzip beachtet: Wie Buschmann *et al.* in [19] beschreiben, ist das Pattern *Layers* ein Grundprinzip bei Software-Architekturen. Durch die *Separation of concerns* werden einzelnen Schichten voneinander getrennt und können so später leichter ausgetauscht werden: “*Using semi-independent parts [...] enables the easier exchange of individual parts at a later date.*” [19]

### Backend

Gerade bei der Visualisierung von abstrakten Daten wie die einer Software, ist eine Schichtentrennung besonders wichtig. CodeLeaves hat das Ziel, eine beliebige objekt-orientierte Software darstellen zu können. Da aber nicht für jede Programmiersprache CodeLeaves angepasst werden soll und die Informationen über die Software aus unterschiedlichen Quellen stammen können, ist hier die erste Schicht *Backend* sinnvoll. In der Backend-Schicht werden die Daten über eine Software aggregiert, seien es statische oder dynamische Informationen und in ein sprach-agnostisches Format – das *Software-Meta-Modell* – gebracht. So können beliebig viele Konnektoren, d.h. die Verbindungen zu den verschiedenen Datenquellen, implementiert werden.

Wie in Abschnitt 1.5 bereits festgehalten, soll der Schritt des Transfers der realen Software in deren abstrakte Repräsentation in dieser Arbeit nur am Rande betrachtet werden. Das Datenmodell ist hingegen sehr wohl zu definieren, um die Daten der nächsten Schicht zur Verfügung zu stellen.

### Anwendungslogik

CodeLeaves könnte neben Software prinzipiell auch jegliche andere, hierarchisch strukturierte Informationen darstellen. Für die interne Datenhaltung wird deshalb in der Schicht *Anwendungslogik* das Datenmodell weiter abstrahiert und in ein Format gebracht, dass sich stark an der Baumstruktur der Informatik orientiert und nicht rein auf Software zugeschnitten ist. So könnten später auch leicht gänzlich andere Daten angebunden werden. Wir nennen das Datenmodell dieser Schicht das *interne Datenmodell*.

### 3 Architektur und Datenmodellierung

#### UI

Die letzte Schicht ist die der *UI*. Das interne Datenmodell wird für die Darstellung der Bäume weiter verarbeitet. Zum Beispiel muss für die Layout-Algorithmen die Höhe der einzelnen Knoten verfügbar sein. Dies sind aber Informationen, die nur von der UI benötigt werden und im internen Datenmodell nichts zu suchen haben. Auf der anderen Seite enthält das *UI-Datenmodell* nur Informationen, die auch tatsächlich zum Rendern der 3D-Objekte benötigt werden.

Diese Schicht könnte wieder ausgetauscht werden und unterschiedliche Layout-Algorithmen könnten unterstützt werden.

In Abbildung 3.1 ist die Beziehung der Datenmodelle in den drei Schichten abgebildet.

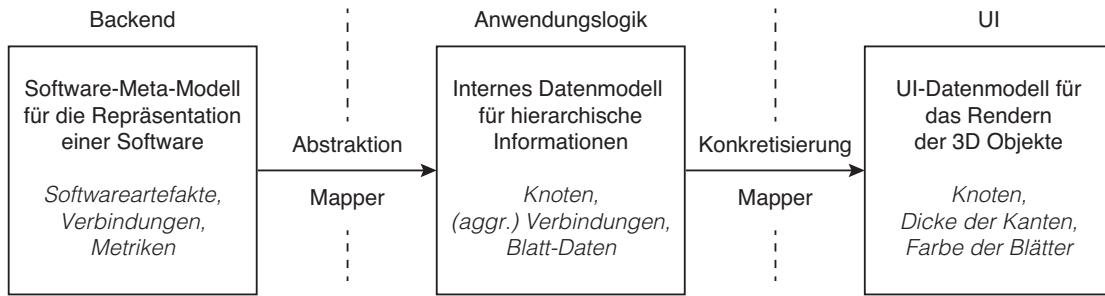


Abbildung 3.1 Schichtentrennung im Datenmodell

## 3.2 Software-Meta-Modell

### Vorhandene Meta-Modelle

CodeCity verwendet als Meta-Modell für Software das sogenannte *FAMIX*-Modell in Version 2.1. Dieses wird bis heute von *Moose*, dem Software-Analyse-Tool, auf dem CodeCity basiert, protegiert und ist heute in Version 3.0 verfügbar [28]. Das Meta-Modell ist sehr umfangreich und beinhaltet 51 verschiedene Entities. Es wird in einem eigenen Dateiformat namens MSE, vergleichbar mit JSON oder XML, gespeichert. Da aber dynamische Informationen nicht im FAMIX-Modell enthalten sind, eignet sich das Format nicht, um in CodeLeaves verwendet zu werden.

Ein Meta-Modell, das dynamische und statische Informationen vereint, existiert nach heutigem Wissensstand nicht.

Für eine reine dynamische Analyse ist das Tool *ExplorVis* sehr ausgereift [18]. Das im nachfolgende entwickelte Meta-Modell ist von Aspekten aus dem FAMIX- und dem ExplorVis-Modell inspiriert. Dabei soll das Modell aber möglichst einfach gehalten werden und nur das beinhalten, was CodeLeaves auch darstellen kann.

#### Entwicklung eines eigenen Modells

Um uns vor Augen zu führen, welche Informationen relevant sind, rufen wir uns die User-Storys aus Kapitel 2 in Erinnerung.

Aus den User-Storys 1 und 4 ergeben sich folgende Metriken, die das Software-Meta-Modell unterstützen muss:

- Statische Metriken zur Code-Qualität
- Antwortzeiten
- Laufzeitfehler

Aus User-Story 2 und 4 gehen folgende Verbindungen hervor:

- Statische Abhängigkeiten
- Dynamische Aufrufe

Diese Metriken und Verbindungen müssen auf die Struktur der Software abgebildet werden. Damit lässt sich das Modell einer Software, ob statisch oder dynamisch betrachtet, auf drei wesentliche Elemente reduzieren:

1. Struktur
2. Metriken
3. Verbindungen

Diese drei Elemente sind in Abbildung 3.2 mit den Klassen `SoftwareArtefact`, `Metric` und `Relation` zu finden. Für den besseren Lesefluss wird im Folgenden auf den Zusatz „...-Klasse“ verzichtet.

Die `Metric` ist ein Schlüssel-Wert-Paar, das durch Codesnippets ergänzt werden kann. So ist es zum Beispiel möglich, bei Laufzeitfehlern zu sehen, an welcher Stelle die Fehler aufgetreten sind. Der `Key` identifiziert die Metrik, wie zum Beispiel statische Testabdeckung oder dynamische Laufzeitfehler und der `Value` enthält die Zahl, die dann in die Farbe der Blätter umgewandelt werden kann.

Die Struktur wird im Software-Meta-Modell nicht wie in den anderen beiden Schichten mit einem Kompositum-Muster entworfen, sondern nur mit dem `SoftwareArtefact`, das eine Liste von Kindern besitzt. Diese Liste ist für eine Klasse leer. Grund für diese Modellierung ist die leichtere Serialisierung. Das Kompositum-Muster, wie es in den andern beiden Schichten Verwendung findet, beinhaltet eine abstrakte Klasse, die vom Blatt und vom Kompositum spezialisiert wird. Für die Deserialisierung müssten bei der Erzeugung von Objekten Zusatzinformationen angegeben werden, ob es sich um ein Blatt, oder ein Kompositum handelt. Mögliche Datenquellen wie SonarQube haben in ihrem Modell für die Struktur ebenfalls nur eine Klasse, wodurch die Logik beim Import der Daten möglichst gering gehalten wird.

Analog zur Definition 2.1 spielt es bei einem `SoftwareArtefact` keine Rolle, wie die Software organisiert ist. Egal ob ein ganzes Projekt, ein Paket oder eine einzelne Methode, alles wird zu einem Softwareartefakt.

### 3 Architektur und Datenmodellierung

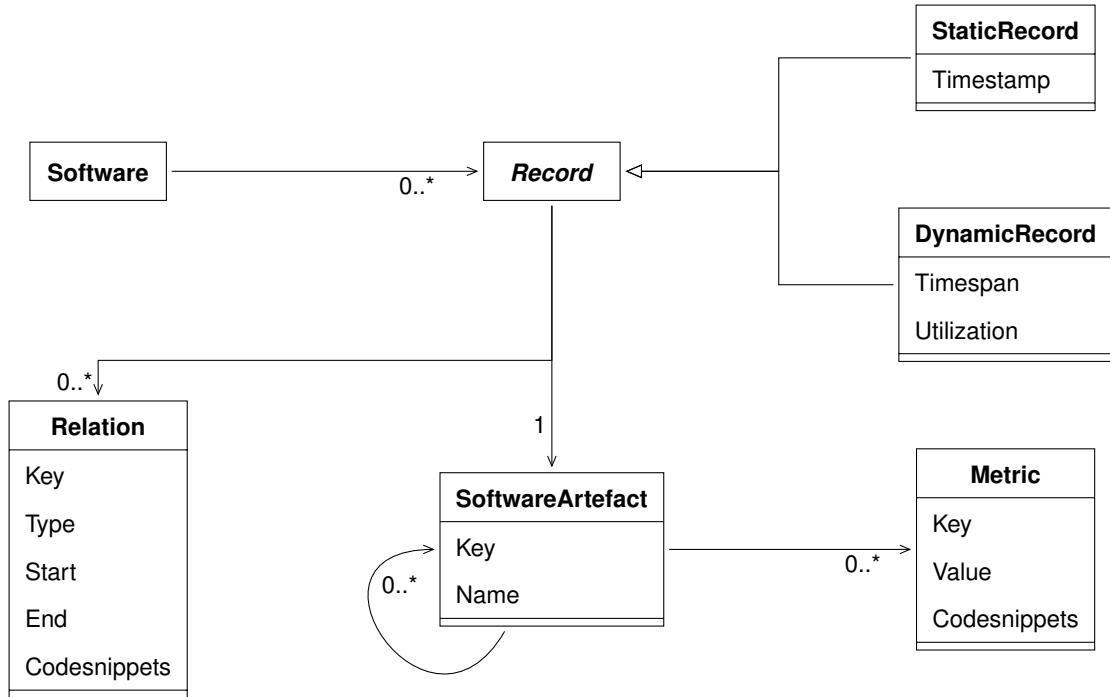


Abbildung 3.2 Meta-Modell einer Software

Jedes SoftwareArtifact besitzt einen für den Nutzer lesbaren Namen und einen Schlüssel, mit dem es eindeutig identifiziert werden kann. Daneben beinhaltet ein SoftwareArtifact beliebig viele Objekte des Klasse Metric. Werden diese jedoch einem SoftwareArtifact mit Kindern, d. h. einem Paket, zugeordnet, finden sie in der weiteren Verarbeitung keine Relevanz, da in CodeLeaves für innere Knoten keine Metriken vorgesehen sind.

Mit der Relation können Verbindungen in einer Software wie statische Abhängigkeiten oder dynamische Aufrufe gespeichert werden. Der Type gibt die Art der Verbindung an. Eine Relation ist nicht direkt mit einem SoftwareArtifact assoziiert, um zyklische Abhängigkeiten im Datenmodell zu vermeiden. Stattdessen werden Start- und Endpunkt der Verbindung mit den Schlüsseln der verbundenen Klassen angegeben.

Durch die Expertengespräche ging hervor, dass sowohl bei Metriken als auch bei Verbindungen ein Bezug zum Code wichtig ist. Demnach beinhaltet die Klasse Relation ebenfalls die Möglichkeit Codesnippets zu speichern, sodass ersichtlich wird, wo im Code die Verbindung ihren Ursprung hat.

Neben Struktur, Metriken und Verbindungen einer Software muss der Zeitraum bzw. der Zeitpunkt angegeben werden können. Damit wird auch die Evolution einer Software zu unterstützt.

Deshalb werden Struktur und Verbindungen in einem Datensatz – dem Record –

gespeichert. Eine Software wiederum kann beliebig viele Datensätze enthalten. Bei einem Datensatz muss zwischen Statik und Dynamik unterschieden werden. Bei statischen Daten wird nur ein bestimmter Zeitpunkt betrachtet, bei dynamischen ist es dagegen ein bestimmter Zeitraum. Daraus resultiert, dass ein Record von einem `StaticRecord` und einem `DynamicRecord` spezialisiert wird.

Aus User-Story 1 geht hervor, dass es bei dynamischen Daten gilt, auch die Ressourcen-Auslastung zu visualisieren. Die Informationen dazu können im `DynamicRecord` in `Utilization` gespeichert werden.

### 3.3 Internes Datenmodell

Wie in Abbildung 3.3 zu sehen, ist das interne Datenmodell sehr ähnlich zu dem des Software-Meta-Modells. Das Modell ist zunächst die semantische Transformation einer Software in ein Format, das beliebige hierarchische Daten darstellt. So wird beispielsweise Software zum Forest, eine Relation zur Connection und eine Metric zu LeafData.

Neben der semantischen Transformation wird das `SoftwareArtifact` durch ein Kompositum-Pattern mit `Node`, `InnerNode` und `Leaf` ersetzt. Mit der Methode `Find` wird das Suchen eines Knotens in den Spezialisierungen von `Node` implementiert. Die Metriken aus dem Software-Meta-Modell sind im internen Datenmodell nur noch bei Blättern verfügbar.

Beim Transfer in das interne Datenmodell werden ebenfalls die einzelnen Verbindungen aggregiert. Eine `AggregatedConnection` besteht mindestens aus einer direkten Verbindung.

Die `Utilization` aus dem `DynamicRecord` wandert als `AdditionalData` in die den abstrakten Record, da bei andern hierarchischen Daten eventuell auch für einen Zeitpunkt zusätzliche Daten relevant sein könnten.

Die Multiplizität zwischen `InnerNode` und `Node` wird mit `0..*` angegeben, was der klassischen Definition eines inneren Knoten (vgl. 2.2) widerspricht. Jedoch kann es auch bei Ausnahmefällen auch vorkommen, dass ein Softwareartefakt weder Daten, noch Kinder besitzt. Bei einem leeren Paket ohne weitere Unterpakete und Klassen wäre das der Fall. Mit der angegebenen Multiplizität kann das Datenmodell auch mit diesen Fällen umgehen. In der Visualisierung resultiert daraus eine Kante ohne Blatt.

In Abbildung 3.3 wurden nur Klassen berücksichtigt, die auch Daten zur Software enthalten. Über die Datensätze zur Software hinaus, werden für den internen Zustand von CodeLeaves weitere Informationen benötigt. Beispielsweise muss gespeichert werden, welche Metriken aktuell angezeigt werden. Einen Eindruck über die Einstellungen, die der Nutzer in CodeLeaves vornehmen kann, wird in Kapitel 6 ausgeführt.

### 3.4 UI-Datenmodell

Das UI-Datenmodell ist stark von der Praxis des Prototypings mit der HoloLens beeinflusst. Es wurde darauf geachtet, dass das Modell nur Daten beinhaltet, das auch zwangsläufig für das Rendern der Visualisierung nötig ist.

### 3 Architektur und Datenmodellierung

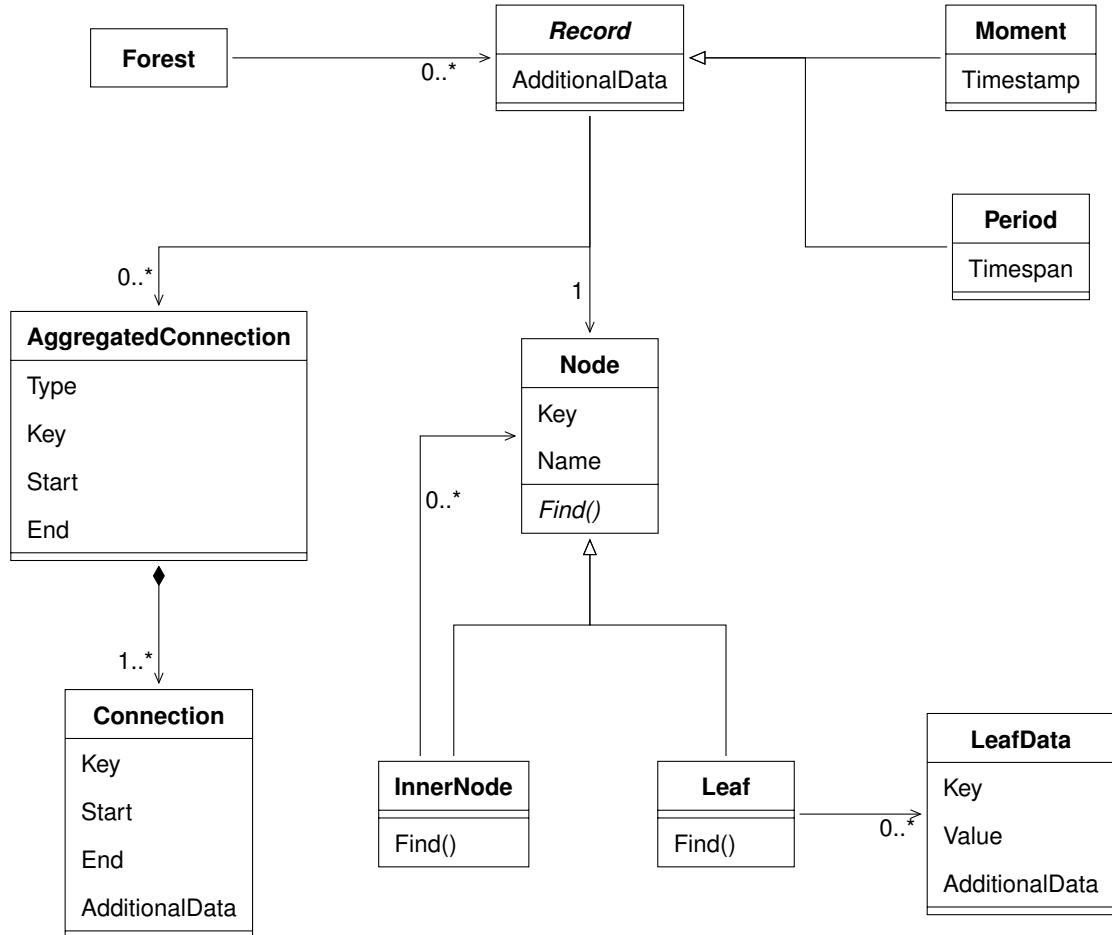


Abbildung 3.3 Internes Datenmodell

Das ist bei einem Blatt die Farbe und bei einem inneren Knoten die Dicke der dazugehörigen Kante. Beide Spezialisierungen eines `UiNode` beinhalten zusätzlich einen beliebigen Text, der abhängig von der Interaktion des Nutzers ist. Zusätzlich sind mit `isHighlighted` und `isSelected` zwei Zustände gegeben, denen entsprechend die Darstellung des Knotens angepasst werden muss. Wie die Reaktion auf Zustandsänderungen im UI-Datenmodell mit reaktiver Programmierung technisch sinnvoll umgesetzt werden kann, ist in Kapitel 5 zu finden.

In Abbildung 3.4 ist das UI-Datenmodell zu sehen.

Die `AdditionalData` aus dem internen Datenmodell finden sich in dem Entwurf des UI-Datenmodells nicht wieder. Grund dafür ist, dass der Kern der Visualisierung davon nicht betroffen ist. Die Anzeige dieser Daten kann beliebig zum Wald dargestellt werden.

Auf den `Circle` bzw. dem `Kreis` eines Knotens, wird im Zuge des Layouts im nächsten Kapitel eingegangen. Für das Layout ist es außerdem nötig, die Knoten sortieren zu können. Die abstrakten Methoden von `UiNode` sind dazu erforderlich und werden entsprechend in `UiInnerNode` und `UiLeaf` realisiert.

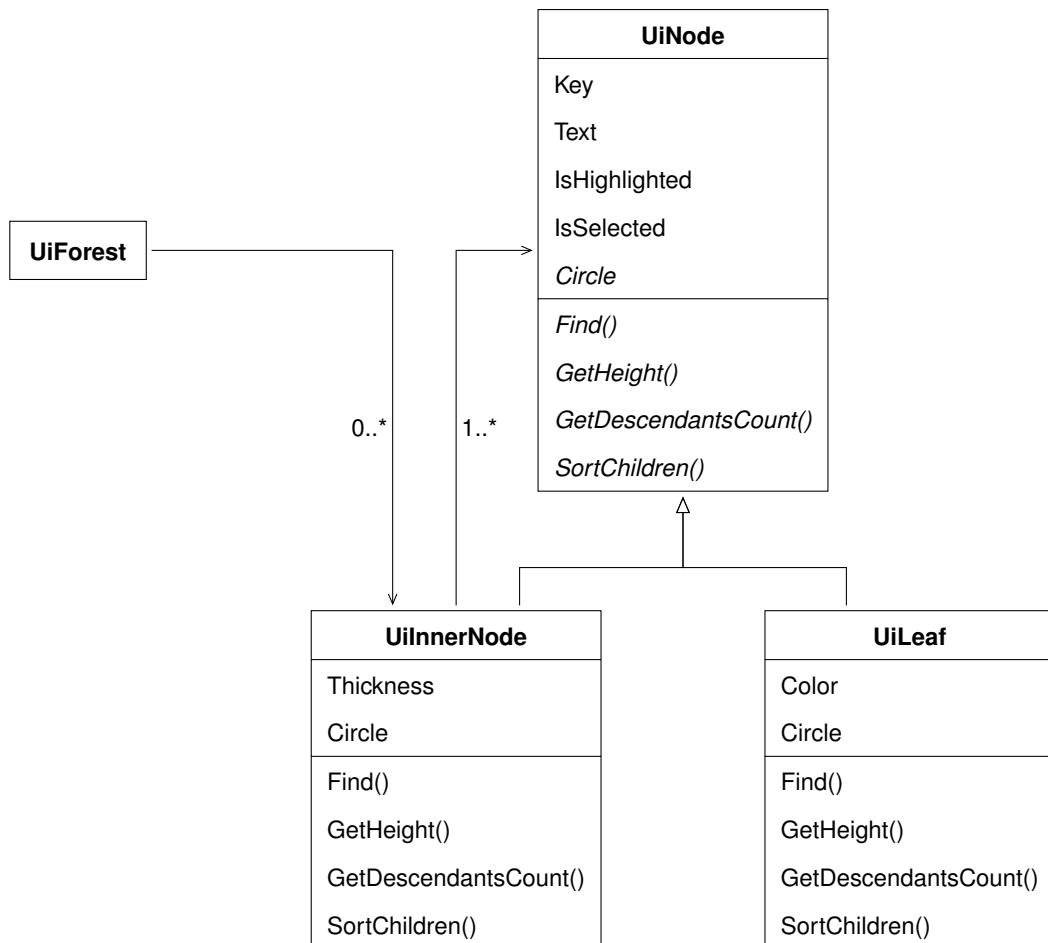


Abbildung 3.4 UI-Datenmodell



# 4 Modellierung und Layout des Waldes

## 4.1 Grundlegender Ansatz

Das Konzept CodeLeaves steht und fällt mit der korrekten und übersichtlichen Darstellung der Baumstruktur. Wegen der Akzeptanzkriterien 1.3 und 5.2 ist die Struktur der Bäume so zu generieren, dass so viele Blätter wie möglich gleichzeitig im Blickfeld des Betrachters sind. Auf der anderen Seite gilt es, wegen Akzeptanzkriterien 2.1 und 5.2 die Struktur übersichtlich und deswegen mit so wenig Überschneidungen wie möglich zu generieren. In Abschnitt 4.2 und 4.3 werden deshalb zwei Algorithmen vorgestellt, die zusammen eine solche Darstellung der Baumstruktur ermöglichen.

### Prototypen

Die Algorithmen werden in einem *High-Fi-Prototyp*<sup>1</sup> für die HoloLens implementiert. Der High-Fi-Prototyp ist wichtig, da nur damit CodeLeaves in der AR erlebt werden kann. Auch die wichtigsten Interaktionen, die im Kapitel 6 beschrieben werden, können im High-Fi-Prototyp schon getestet werden. Weitere Interaktions-Elemente werden in Kapitel 6 mit einem Papier *Low-Fi-Prototyp*<sup>2</sup> ergänzt.

Bei dem High-Fi-Prototyp wird auf den Standard-Workflow der HoloLens-Entwicklung gesetzt. HoloLens-Applikationen werden in *Unity*<sup>3</sup> entwickelt. Unity ist eine plattformübergreifende 2- und 3D-Game-Engine und nach eigenen Angaben „*Die weltweit führende Software der Game-Industrie*

Für die HoloLens wird die Position der Hauptkamera in Unity auf die Position der HoloLens übertragen. Hologramme entstehen durch normale 3D-Objekte in Unity. Diesen Objekten können Skripte zugewiesen werden, die zur Laufzeit ausgeführt werden. Die verwendete Skript-Sprache im Prototyp von CodeLeaves ist C#.

Für die Generierung von Bäumen mussten zunächst 3D-Objekte entworfen werden, die die Knoten und Blätter repräsentieren. Dies wurde nicht in Unity, sondern mithilfe des professionellen 3D-Modellierungs-Software Maya<sup>4</sup> von Autodesk bewerkstelligt.

<sup>1</sup>englisch High Fidelity, aus: high = hoch und fidelity = Genauigkeit; High-Fi-Prototypen zeichnen sich durch eine High-Tech-Darstellung der Designkonzepte aus, die zu einer partiellen bis vollständigen Funktionalität führt [15].

<sup>2</sup>englisch Low Fidelity, aus: low = gering und fidelity = Genauigkeit; Low-Fi-Prototypen zeichnen sich durch eine schnelle und einfache Übersetzung von Designkonzepten auf hohem Niveau in greifbare und testbare Artefakte aus [15].

<sup>3</sup><https://unity3d.com/de>

<sup>4</sup><https://www.autodesk.de/products/maya/overview>

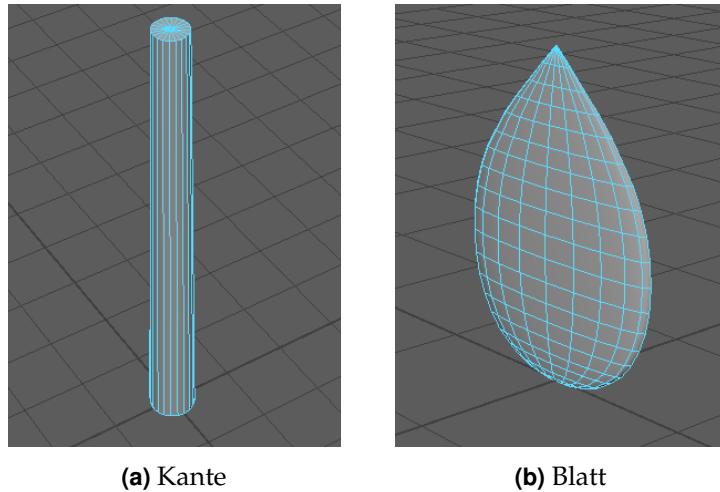
## 4 Modellierung und Layout des Waldes

### 3D-Objekte

Die Bäume für CodeLeaves kommen mit den Objekten Kante und Blatt aus. Ein innerer Knoten wird der Baum Metapher folgend immer durch die Kante zum Elternteil bzw. zum Waldboden dargestellt.

**Kanten-Objekt** Eine Kanten-Objekt ist ein vertikal ausgerichteter, nach oben leicht zulaufender Zylinder was die Abbildung 4.1a veranschaulicht ist. Die Pivots zur Position, Rotation und Skalierung wurden so gewählt, dass eine Manipulation des Objekts immer um den Mittelpunkt der Grundfläche erfolgt.

**Blatt-Objekt** Das Blatt-Objekt wurde als Abstraktion eines echten Blattes modelliert und ist in Abbildung 4.1b zu sehen.



**Abbildung 4.1** 3D-Objekte für den High-Fi-Prototyp

Das Blatt-Objekt hat aufgrund der flachen Form die Eigenschaft, dass die sichtbare Fläche bei seitlicher Betrachtung deutlich geringer ist als von vorne. Wir stellen uns einen Baum vor, bei dem der Betrachter nur die Kanten der Blätter sieht. Dadurch lassen sich die Farben der Blätter nur schwer miteinander vergleichen. Dies sollte aber aus jedem Winkel möglich sein (Akz. 1.4).

Eine Abhilfe könnte ein Blatt-Objekt sein, dass in allen Dimensionen gleich dick ist. Dies würde aber eher an eine Knospe erinnern und schaut wenig natürlich aus. Deshalb wird dem Blatt-Objekt in Unity ein Script hinzugefügt, dass das Blatt zur Laufzeit immer in die Richtung des Betrachters rotieren lässt. Ein 3D-Objekt mit solchem Verhalten wird *Billboard* genannt. Mithilfe des Billboard-Verhaltens ist gewährleistet, dass der Nutzer von jedem Winkel aus immer eine optimale Übersicht über das Blätterdach besitzt.

## 4.1 Grundlegender Ansatz

### Beschaffung von realistischen Beispieldaten

Für die Generierung der Struktur im Prototyp wurde mit einem Beispielprojekt gearbeitet: Das Projekt *Air* ist eines der größten Software-Projekte der QAware. Es wird seit rund fünf Jahren entwickelt. Dank seiner Größe und Komplexität ist es somit ein geeigneter Benchmark für die Entwicklung der UI-Schicht von CodeLeaves.

In Tabelle 4.1 sind hinsichtlich des entstehenden Waldes Eckdaten von Air angegeben.

**Tabelle 4.1** Eckdaten des Beispielprojektes Air

Bäume	Knoten	Blätter	innere Knoten
40	6736	5373	1363

Alle Projekte von QAware werden in SonarQube überwacht. SonarQube bietet eine Web-API, durch die die Struktur eines Projekts und unterstützte statische Metriken abgefragt werden können. Diese API wird im Prototyp genutzt, um die benötigten Informationen für Struktur der Software und eine Beispiel-Metrik abzufragen. Die Struktur wird bei der Abfrage der API rekursiv zusammengebaut, da SonarQube für jeden Aufruf eines Softwareartefakts immer nur dessen direkten Kinder liefert.

Das Format von SonarQube wird dabei schon in das generische Software-Meta-Modell von CodeLeaves transferiert. Da bei einem Projekt in der Größenordnung von Air sehr viele HTTP-Requests entstehen können, wurde für den Prototyp das entstandene Meta-Modell lokal in eine Datei geschrieben, sodass dieses als Beispiel-Datensatz dient. Prinzipiell können aber auch dynamisch andere Projekte geladen werden.

Das Datenmodell des Backends wird in das interne Datenmodell der Anwendungslogik und weiter in das UI-Datenmodell transferiert.

Nun liegt eine Struktur vor, die es in Bäume zu verwandeln gilt. Der grundlegende Ablauf des Algorithmus zur Generierung eines Baumes mit Knoten  $K$  als Kronenansatz und dessen Kindern  $\{(K_i) \mid i = 1, 2 \dots N\}$  lässt sich wie folgt beschreiben:

- (i) Instanziere ein Kanten-Objekt  $E$  als Stamm des Baums.
- (ii) Falls Knoten  $K$  ein Blatt ist, füge ein Blatt-Objekt am Ende von  $E$  an.
- (iii) Falls Knoten  $K$  ein innerer Knoten ist, finde für jedes Kind  $K_i$  eine passende Position, füge ein Kanten-Objekt  $E_i$  zwischen dem Ende von  $E$  und  $K_i$  an, setze  $K$  gleich  $K_i$  und  $E$  gleich  $E_i$  und geht zu (ii).

Wie die Kinder eines Knotens sinnvoll verteilt und deren Kanten entsprechend rotiert und gestreckt werden müssen, soll im Folgenden erarbeitet werden.

Es stellt sich zunächst die Frage, auf Grundlage welcher geometrischen Figur die Kinder verteilt werden. Möglich wären z.B. eine Kugel oder eine Kreisfläche.

## 4 Modellierung und Layout des Waldes

### Verteilung von Kindern auf einer Kugel

Werden die Kinder gleichmäßig auf der Oberfläche einer Kugel verteilt, wird der 3D-Raum maximal ausgenutzt. Das Ergebnis käme einem *Fractal-Tree* sehr nahe. Diese Bäume werden rekursiv durch die Neigung der Kanten konstruiert. Ein solcher Fractal-Tree ist in Abbildung 4.2 dargestellt. Das Konzept ließe sich ohne Schwierigkeiten auf die Dreidimensionalität übertragen. Das Problem dabei ist, dass bei diesem Ansatz die Kanten auch rekursiv verkürzt werden. Dadurch wird gewährleistet, dass die Äste nicht in den Boden wachsen.

Für CodeLeaves wären immer kleiner werdende Kanten wenig sinnvoll, da so eine Interaktion mit Kanten und Blättern kaum möglich wäre. Auch wären die Überschneidungen, die in Abbildung 4.2 zu sehen sind, bei größeren Bäumen nur durch Neigung der Kanten nicht zu vermeiden.

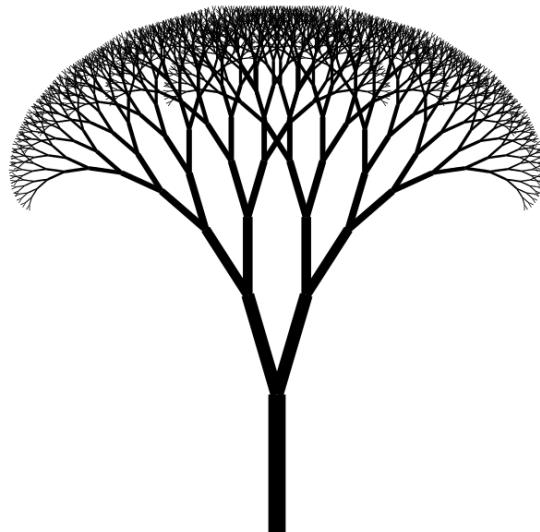


Abbildung 4.2 Fractal-Tree [39]

### Verteilung von Kindern auf einer Kreisfläche

Werden die Kinder stattdessen auf einer Kreisfläche überhalb des Elternteils positioniert, entstehen dadurch drei Vorteile:

Erstens befinden sich Softwareartefakte mit der gleichen hierarchischen Tiefe auch auf gleicher Höhe, was sich positiv auf die Übersichtlichkeit auswirkt.

Zweitens wird das Blätterdach primär am Ende des Baumes auf einer Ebene angezeigt, was aus der Vogelperspektive eine gute Betrachtung der Metriken verspricht. Gleichzeitig wird von der Seite der Blick auf die Struktur des Baumes nicht versperrt.

Drittens kann durch die erforderliche Längenanpassung der Kanten Kollisionen von Ästen vermieden werden. Der Abstand zwischen zwei Geschwistern muss so groß gewählt werden, dass die Kanten aller Nachfahren beider Geschwister nicht

## 4.2 Verteilung von Blättern mit dem Sonnenblumen-Algorithmus

miteinander kollidieren.

Der Prototyp von CodeLeaves verwendet aus genannten Gründen eine Kreisfläche für die Verteilung von Kindern eines Knotens.

### 4.2 Verteilung von Blättern mit dem Sonnenblumen-Algorithmus

Betrachten wir nun zunächst den Fall eines inneren Knotens, der nur Blätter als Nachfahren besitzt. D.h. die Verteilung kann gleichmäßig erfolgen und muss die Breite nachfolgender Knoten nicht miteinbeziehen.

Dieser Fall tritt in realen Softwareprojekten häufig auf. In Air existieren 1154 solcher Knoten, was 85% aller inneren Knoten ausmacht. Die Verteilung der Kinder eines inneren Knotens mit weiteren inneren Knoten wird in Abschnitt 4.3 behandelt.

*Es wird ein Algorithmus gesucht, bei dem Punkte auf einer Kreisfläche gleichmäßig verteilt werden.*

Dazu liefert uns die Natur ein schönes Beispiel. Die kleinen inneren Blüten der Sonnenblume, die sogenannten Röhrenblüten, die nach ihrer Befruchtung die Sonnenblumenkerne ausbilden, nutzen den Platz innerhalb der gelben Zungenblüten optimal aus [52]. Die spiralförmige Anordnung ist nicht nur schön anzusehen, sondern folgt auch einem genauen Muster.

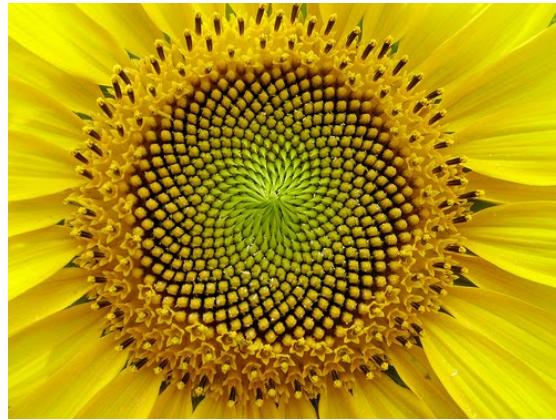


Abbildung 4.3 Spiralförmige Anordnung der Röhrenblüten einer Sonnenblume [17]

#### Der Goldene Winkel

Ausgehend von dem Mittelpunkt der Sonnenblume ist jede nachfolgender Röhrenblüte um rund  $137.5^\circ$  um den Mittelpunkt rotiert. Dieser sogenannte *Goldene Winkel* entsteht durch die Teilung des Vollkreises durch den *Goldenen Schnitt*. Der Goldene Schnitt ist ein Teilungsverhältnis, das oft bei Größenverhältnissen von einfachen geometrischen Figuren vorkommt und ist mit

## 4 Modellierung und Layout des Waldes

$$\Phi = \frac{a}{b} = \frac{a+b}{a} = \frac{1+\sqrt{5}}{2} \approx 1.618 \quad (4.1)$$

definiert [46].

Wird der Vollwinkel durch den Goldenen Schnitt geteilt, entsteht der Winkel

$$\frac{2\pi}{\Phi} \approx 222.5^\circ. \quad (4.2)$$

Die Ergänzung zum Vollwinkel

$$2\pi - \frac{2\pi}{\Phi} = \frac{2\pi}{\Phi^2} \approx 137.5^\circ \quad (4.3)$$

ist der Goldene Winkel [46].

Er hat die Eigenschaft, dass er auch bei mehrfacher Addition über Vollwinkel hinaus nie den selben Winkel ergibt. Diese Eigenschaft machen sich viele Pflanzen zunutze und kann auch in CodeLeaves Anwendung finden.

### Berechnungen für das Hinzufügen eines Blattes

Mit dem Vielfachen des goldenen Winkels ist der Winkel  $\varphi$  gegeben, um den eine neue Kante eines neuen Blattes um die Senkrechte rotiert werden muss. Für die Position eines neuen Blattes sind aber noch andere Größen zu berechnen.

Angenommen eine Kante mit der endgültigen Länge  $l$  wird an einen Knoten  $K$  angefügt, um die  $x$ -Achse mit dem Winkel  $\theta$  geneigt und anschließend um die  $y$ -Achse mit dem Winkel  $\varphi$  rotiert, so entsteht ein *Kugelkoordinatensystem* mit der Position von  $K$  als Ursprung, der  $y$ -Achse als Polachse, der Position des  $n$ -ten Kindknotens als Punkt  $P$ , dem *Polarwinkel*  $\theta$  und dem *Azimutwinkel*  $\varphi$  [33].

Diese Größen sind in Abbildung 4.4 dargestellt, wobei die Höhe  $h$  als Standardhöhe einer Kante gegeben gilt und  $E$  mit  $(0, h, 0)$  die Position des 0-ten Kindes ist. Die  $y$ -Achse wird in Anlehnung an Unity in dieser Arbeit als Senkrechte verwendet. Der Waldboden liegt demnach in einer  $xz$ -Ebene.

Der Abstand  $a_n$  zwischen dem Punkt  $E$  und  $P$  berechnet sich für den  $n$ -ten Knoten nach [43] mit

$$a_n = c\sqrt{n}, \quad (4.4)$$

wobei  $c$  eine Konstante ist und den Abstand der Kindknoten untereinander beeinflusst.

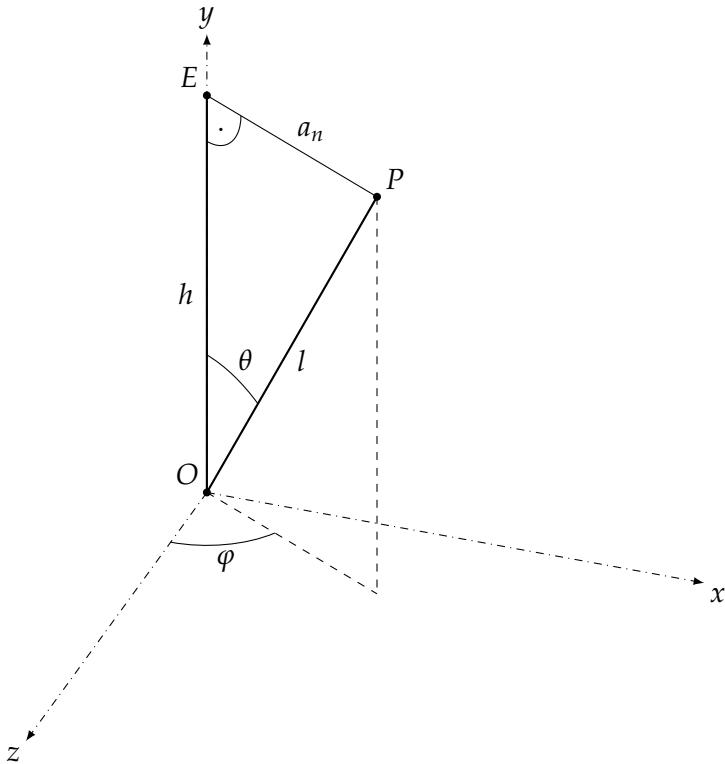
Die Rotation um die  $y$ -Achse  $\varphi$  ist wie oben hergeleitet das  $n$ -te Vielfache vom Goldenen Winkel:

$$\varphi = \frac{2\pi \cdot n}{\Phi^2} \quad (4.5)$$

Der Polwinkel  $\theta$  lässt sich mithilfe von  $h$  und  $a_n$  berechnen:

$$\theta = \tan^{-1} \left( \frac{a_n}{h} \right) \quad (4.6)$$

## 4.2 Verteilung von Blättern mit dem Sonnenblumen-Algorithmus



**Abbildung 4.4** Kugelkoordinatensystem mit den zu berechnenden Größen für das Hinzufügen eines neuen Blattes

Die Länge  $l$  der Kante kann mit

$$l = h \cdot \cos(\theta) \quad (4.7)$$

berechnet werden.

Für die Positionierung des Kindknotens muss der Punkt  $P$  vom konstruierten Kugelkoordinatensystem in das kartesische Koordinatensystem umgerechnet werden. Dies wird durch die Formel

$$\vec{P} = \begin{pmatrix} \sin(\varphi) \cdot \sin(\theta) \cdot l \\ \cos(\theta) \cdot l \\ \cos(\varphi) \cdot \sin(\theta) \cdot l \end{pmatrix} \quad (4.8)$$

erreicht [47].

Damit ist alles für den Algorithmus gegeben, der Blätter eines Knotens mit gleichmäßigem Abstand auf eine gegebene Höhe verteilt und die Kanten entsprechend rotieren und skalieren kann.

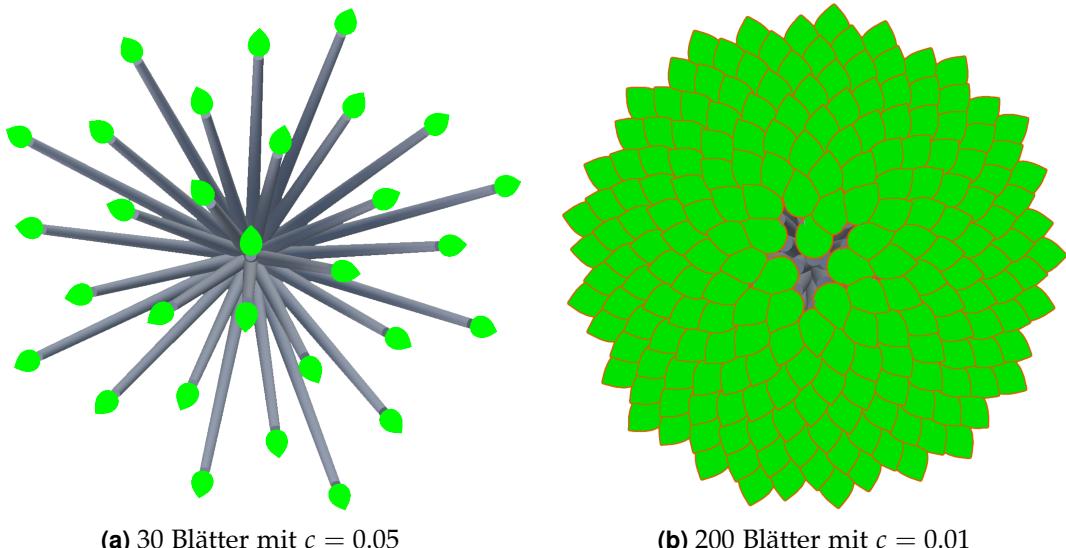
In Unity und C# ist die Implementierung dieses Algorithmus in vereinfachter Form in Listing 4.1 zu sehen ist.

## 4 Modellierung und Layout des Waldes

**Listing 4.1:** Hinzufügen von Blättern eines Knotens

```
1  for (var i = 0; i < node.Children.Count; i++)
2  {
3      var d = TreeGeometry.CalcDistance(i);
4      var phi = TreeGeometry.CalcPhi(i);
5      var theta = TreeGeometry.CalcTheta(DefaultEdgeHeight, r);
6      var l = TreeGeometry.CalcEdgeLength(DefaultEdgeHeight, theta);
7      var pVec = TreeGeometry.CalcNodePosition(l, theta, phi);
8
9      AddEdgeObject(l, theta, phi);
10     AddEmptyNodeObject(pVec);
11 }
```

Das Ergebnis ist in Abbildung 4.5 zu sehen. Auf der linken Seite (4.5a) sind 30 Blätter verteilt worden. Die Ähnlichkeit zur Verteilung der Röhrenblüten der Sonnenblume wird auf der rechten Seite (4.5b) mit der Extrem situation mit 200 Blättern und kleiner Konstante  $c$  sichtbar. Darin wurde um die Blätter eine Umrandung hinzugefügt, um die Blätter voneinander abgrenzen zu können.



**Abbildung 4.5** Sonnenblumen-Algorithmus zur Verteilung von Blättern

### 4.3 Verteilung innerer Knoten mit Circle-Packing

Die Verteilung mit dem Sonnenblumen-Algorithmus beruht auf der Tatsache, dass die Kinder des Knotens mit gleichmäßigem Abstand zueinander verteilt werden können. Dies ist bei inneren Knoten, die wiederum innere Knoten als Kinder besitzen, nicht

### 4.3 Verteilung innerer Knoten mit Circle-Packing

gegeben. In diesem Fall muss die Breite der Kinder mit berücksichtigt werden, sodass die Äste der einzelnen Kinder nicht kollidieren.

Für die Verteilung der inneren Knoten führen wir ein weiteres Merkmal ein.

#### Definition 4.1: Kreis eines inneren Knotens

Der Kreis eines inneren Knotens ist der Kreis, aus dem alle Nachfahren des Knotens von oben betrachtet nicht hinausragen.

Bei einem inneren Knoten, dessen  $N$  Kinder nach dem Sonnenblumen-Algorithmus verteilt wurden, kann der Radius seines Kreises leicht berechnet werden. Der Radius  $r$  ist gleich dem Abstand  $a_N$  des  $N$ -ten Blattes, der nach Formel 4.4 mit  $a_n = c\sqrt{n}$  und  $n = N$  berechnet werden kann.

Werden nun alle inneren Knoten so verteilt, dass sich von oben betrachtet die Kreise der Kinder nicht überschneiden, sind alle Äste eines Baumes kollisionsfrei verteilt und alle Blätter sind von oben sichtbar.

Das Problem der Verteilung von inneren Knoten kann demnach auf jeder Ebene eines Baumes auf 2D reduziert werden.

*Es wird ein Algorithmus gesucht, der Kreise unterschiedlicher Größe auf möglichst kleinem Raum überschneidungsfrei positioniert.*

Diese Aufgabenstellung wird vom sogenannten *Circle-Packing* gelöst.

Dazu hat W. Wang in [44] 2006 einen Algorithmus entworfen, der seitdem von vielen adaptiert wurde. Zum Beispiel verwendet M. Bostock, der Schöpfer in der populären JavaScript Library *d3.js* zur Datenvisualisierung, diesen Circle-Packing-Algorithmus in *d3.js* [4]. In 3D wurde er jedoch noch nicht angewandt.

Der Algorithmus muss noch an die Gegebenheiten von CodeLeaves angepasst werden. Bei Wang's Version startet der Algorithmus bereits mit drei tangentialen Kreisen. Der Mittelpunkt, um den zusätzliche Kreise positioniert werden, befindet sich zwischen den drei initialen Kreisen. Bei CodeLeaves sind aber nicht zwangsläufig bei jedem Knoten mindestens drei Kindknoten zu verteilen und als Mittelpunkt für die Verteilung der Kreise ist der Mittelpunkt des 0-ten Kreises sinnvoll, da sich so immer eine Verlängerung des Stammes ergibt.

Ist nur ein Knoten zu verteilen, wird der Mittelpunkt seines Kreises von oben gesehen (in der xz-Ebene) in den Ursprung gelegt. Der Kreis eines zweiten Knotens wird in einem zufälligen Winkel tangential zum Kreis des ersten Knotens positioniert. Bereits bei einem dritten Kreis kann Wang's Algorithmus Anwendung finden.

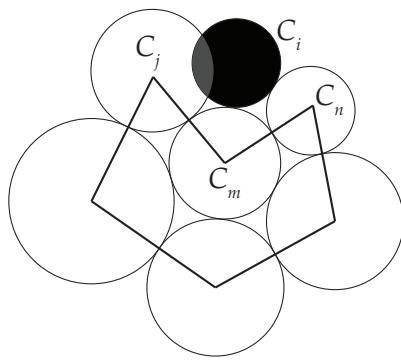
Es wird eine sogenannte *Front-Chain* verwendet, eine zyklische Liste, die alle Kreise enthält, an die potentiell ein neuer Kreis angefügt werden könnte. Die Front-Chain ist in Abbildung 4.6 als dicke Linie dargestellt. Bei der Positionierung des ersten bzw. des zweiten Kreises werden diese zur Front-Chain hinzugefügt.

## 4 Modellierung und Layout des Waldes

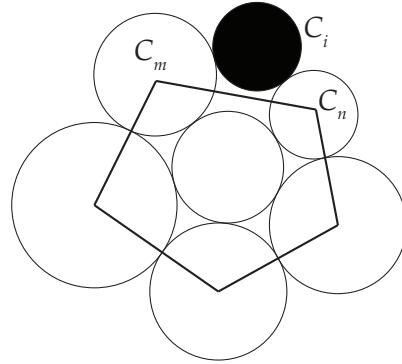
Im Ganzen ist der leicht modifizierte Algorithmus für eine Menge von Kreisen  $\{C_i \mid i = 0, 2, \dots, N\}$  im Folgenden beschrieben:

### Abgewandelte Form von Wang's Circle-Packing-Algorithmus (nach [44])

- (i) Falls  $i = 0$  setze den Mittelpunkt von  $C_0$  in den Ursprung und füge  $C_0$  zur Front-Chain hinzu.
- (ii) Falls  $i = 1$  setze den Mittelpunkt von  $C_1$  so, dass  $C_1$  in einem zufälligen Winkel tangential zu  $C_0$  ist und füge  $C_1$  zur Front-Chain hinzu.
- (iii) Für  $1 < i < N$  suche  $C_m$ , den Kreis, dessen Mittelpunkt am nächsten zum Ursprung liegt.  $C_n$  ist der Kreis in der Front-Chain nach  $C_m$  ( $n = m + 1$ ).
- (iv) Berechne den Mittelpunkt von  $C_i$  so, dass er tangential zu  $C_m$  und  $C_n$  ist.
- (v) Suche einen Kreis  $C_j$  der sich mit  $C_i$  überschneidet.
- (vi) Falls der  $C_j$  nicht existiert, setze  $C_i$  gleich  $C_{i+1}$  und gehe zu (iii).
- (vii) Falls  $C_j$  in der Front-Chain näher an  $C_m$  als an  $C_n$  liegt, lösche alle Kreise aus der Front-Chain, die zwischen  $C_j$  und  $C_n$  liegen. Setze  $C_m$  gleich  $C_j$  und gehe zu (iv).
- (viii) Falls  $C_j$  in der Front-Chain näher an  $C_n$  als an  $C_m$  liegt, lösche alle Kreise aus der Front-Chain, die zwischen  $C_m$  und  $C_j$  liegen. Setze  $C_n$  gleich  $C_j$  und gehe zu (iv).



(a) Platzieren eines neuen Kreises



(b) Anpassung der Front-Chain und  
erneutes Platzieren

**Abbildung 4.6** Wang's Circle Packing Algorithmus (nach [4])

In Abbildung 4.6 ist ein Ausschnitt aus dem Algorithmus visualisiert. Auf der linken Seite (4.6a) ist Schritt (iv) dargestellt, wobei  $C_j$  existiert und näher an  $C_m$  als an  $C_n$  liegt. Demnach werden alle Knoten zwischen  $C_j$  und  $C_n$  aus der Front-Chain entfernt (im Beispiel nur  $C_m$ ) und  $C_i$  wird erneut platziert. Das Resultat ist auf der rechten Seite (4.6b) zu sehen.

Werden die Knoten vor der Durchführung des Algorithmus nach deren Radien sortiert, befindet sich der Knoten mit dem größten Kreis genau über dem Stamm und nach außen hin werden die Äste kleiner, was einem natürlichen Wuchs am nächsten kommt. Durch die zufällige Positionierung des 1-ten Knotens wächst der Baum nicht bei jedem Knoten „in die gleiche Richtung“ was das natürliche Bild des Waldes ebenfalls fördert.

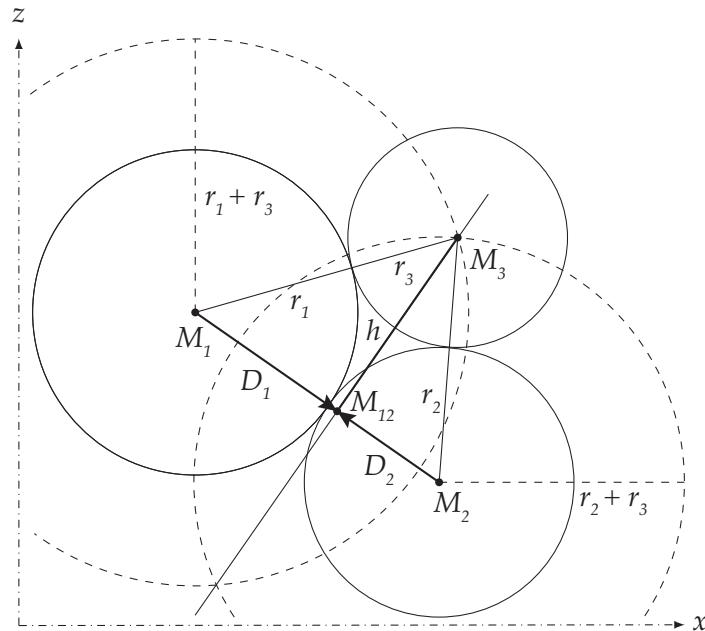
### 4.3 Verteilung innerer Knoten mit Circle-Packing

Der oben beschriebene Algorithmus setzt in (ii) und in (iv) die Berechnung eines Mittelpunktes voraus, sodass dessen Kreis tangential zu einem bzw. zwei weiteren Kreisen ist.

Bei dem 1-ten Kreis mit Radius  $r_1$  und Mittelpunkt  $M_1$ , der nur tangential zu dem 0-ten Kreis mit  $r_0$  und  $M_0 = (0, 0)$  ist und mit dem zufälligen Winkel  $\alpha$  platziert wird, ergibt sich für die xz-Ebene folgende Koordinate:

$$\vec{M}_1 = \begin{pmatrix} x = \cos(\alpha) \cdot (r_1 + r_2) \\ y = \sin(\alpha) \cdot (r_1 + r_2) \end{pmatrix} \quad (4.9)$$

Bei allen weiteren Kreisen  $C_i$  ist die Berechnung komplexer. Der Sachverhalt ist in der XZ-Ebene  $E$  in Abbildung 4.7 abgebildet. Gesucht ist  $M_3$ . Gegeben sind die Mittelpunkte  $M_1$  und  $M_2$  sowie die Radien der drei Kreise.



**Abbildung 4.7** Berechnung des Mittelpunktes eines Kreises, tangential zu zwei anderen Kreisen

$M_3$  befindet sich genau an einem Schnittpunkt der beiden Kreise  $\{X \in E \mid \overline{M_1 X} = r_1 + r_3\}$  und  $\{X \in E \mid \overline{M_2 X} = r_2 + r_3\}$ , die in Abbildung 4.7 gestrichelt veranschaulicht sind. Diese Schnittpunkte lassen sich mithilfe von Vektorrechnung nach [51] wie folgt errechnen:

$$\vec{M}_{3_{1/2}} = \vec{M}_1 + \vec{D}_1 \pm h \cdot \vec{e} \quad (4.10)$$

Dabei ist  $e$  ein zu  $M_2 - M_1$  orthogonaler Einheitsvektor, für  $h$  gilt

$$h = \sqrt{r_1^2 - |\vec{D}_1|^2} = \sqrt{r_2^2 - |\vec{D}_2|^2} \quad (4.11)$$

## 4 Modellierung und Layout des Waldes

und  $D_1$  lässt sich durch Umformen der Gleichung 4.11 mit

$$\vec{D}_1 = \frac{1}{2} \cdot \left( \frac{r_1^2 - r_2^2}{|\vec{M}_2 - \vec{M}_1|} + 1 \right) \quad (4.12)$$

darstellen.

Mit diesen Gleichungen kann der Circle-Packing-Algorithmus für Kinder eines Knotens angewandt werden. Für die Positionierung der Knoten im dreidimensionalen Raum kommt für die y-Koordinate jeweils noch die Länge der Kante des 0-ten Geschwisters hinzu. Die Kanten der Geschwister müssen entsprechend deren Position angepasst werden

### Knoten mit gemischten Kindern

Für den Fall, dass ein Paket innere Knoten **und** Blätter enthält, kann der Circle-Packing-Algorithmus ebenfalls angewandt werden. Dafür muss lediglich einem Blatt ein Radius zugewiesen werden. Mit der Verwendung der Konstante  $c$  aus dem Sonnenblumen-Algorithmus als Radius haben die Blätter bei beiden Verteilungs-Algorithmen den gleichen Abstand zueinander. Lediglich die Ausbreitung der Blätter um den Ursprung erfolgt weniger natürlich als beim Sonnenblumen-Algorithmus.

### Hierarchische Anwendung

Für die Anwendung auf mehreren Ebenen ist nach der Verteilung von Kindern eines Knotens  $K$  auf Ebene  $n$  die Berechnung des Kreises von  $K$  nötig, dass in Ebene  $n - 1$  die Generation von  $K$  verteilt werden kann.

Der Kreis von Knoten  $K$ , mit Mittelpunkt an der Position von  $K$ , der alle Kreise der Kinder von  $K$  umschließt und mindestens zu einem Kreis eines Kindes tangential ist, nennen wird den *größten umschließenden Kreis*. Sein Radius ist der Abstand vom 0-ten zum  $n$ -ten Kind, zuzüglich des Radius des  $n$ -ten Kindes. Diese Methode ist nicht der *kleinste umschließende Kreis*, reicht für die Generierung von überschneidungsfreien Bäume jedoch aus.

Für die vollständige hierarchische Anwendung des Circle-Packing wird für jeden Knoten das Circle-Packing rekursiv durchgeführt und anschließend der Radius des eigenen Kreises gesetzt. Abbruchbedingung für die Rekursion ist, dass der Knoten ein Blatt oder ein innerer Knoten mit ausschließlich Blättern ist. In letzterem Fall greift der Sonnenblumen-Algorithmus.

Der Algorithmus fängt also mit dem Knoten mit der maximalen Tiefe an und arbeitet sich über Geschwister und Elternteile bis hin zum Kronenansatz fort, bis alle Knoten eines Baumes verteilt sind.

### Verteilung von Bäumen

Nachdem die Kronenansätze Kreise besitzen, sind damit auch die Radien der ganzen Bäume bekannt. Damit können die Bäume prinzipiell beliebig platziert werden, ohne dass sich deren Äste überschneiden. Es wäre zum Beispiel eine Verteilung in einem

#### 4.4 Verwendete Algorithmen in der Praxis

Gitter möglich, was dann an eine Baumschule mit Baumreihen erinnern würde. Es ist jedoch auch ganz einfach möglich, die Bäume ebenfalls mit Circle-Packing anzugeordnen. Damit entsteht ein runder Wald und die Bäume nehmen möglichst wenig Platz ein.

Mit den beiden beschriebenen Algorithmen können Bäume garantiert überschneidungsfrei generiert und verteilt werden. Der damit generierte Wald des Projektes Air ist von der Seite in Abbildung 4.8a und von oben in 4.8b zu sehen. Die Kreise in Abbildung 4.8b sind die Visualisierungen der durch das Circle-Packing entstehenden Kreise. Aus der Vogelperspektive ist gut zu erkennen, wie die Blätter innerhalb ihrer Eltern mithilfe des Sonnenblumen-Algorithmus sehr gleichmäßig verteilt werden. Die Farben der Blätter werden in Abschnitt 4.5 aufgegriffen.

Auch das Circle-Packing ist durch die Kreise gut zu sehen. Jedoch lässt sich feststellen, dass durch die Wahl des größten umschließenden Kreises Platz "verschenkt" wird. Besonders in der Mitte der Abbildung wird dies durch die große Fläche deutlich, in der keine weiteren Knoten platziert sind.

Um das zu umgehen, müsste anstelle des größten umschließenden Kreises der kleinste umschließende Kreis verwendet werden. Dies ist ein weiteres mathematisches Problem und als Teil des *Appollonius Problem* bekannt [13].

Für den Prototyp ist die Umsetzung des kleinsten umschließenden Kreises für Co-deLeaves aber nicht essentiell. Entscheidend ist, dass die Struktur der visualisierten Software ohne Überschneidungen dargestellt wird, was auch mit dem größten umschließenden Kreis möglich ist. Die Umstellung auf den kleinsten umschließenden Kreis ist deshalb eine weiterführende Aufgabe.

## 4.4 Verwendete Algorithmen in der Praxis

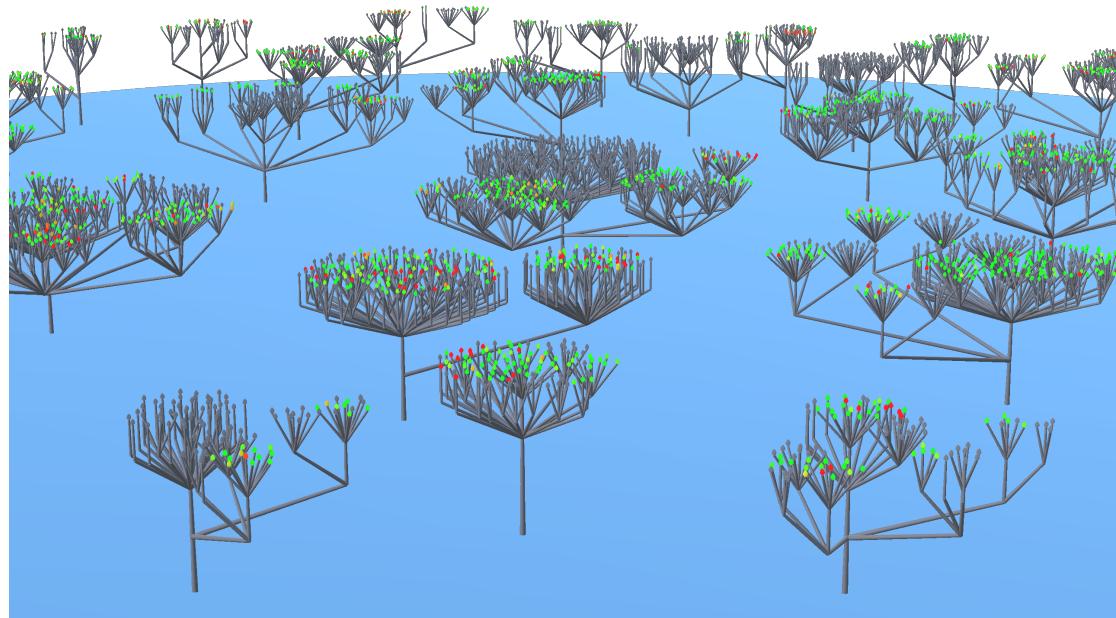
Für die Praxis in Unity ist bei der Verteilung der Bäume noch zu beachten, dass deren Radien erst verfügbar sind, nachdem der Circle-Packing-Algorithmus für alle Bäume vollständig durchlaufen wurde. Das heißt, dass bei der Generierung der einzelnen Bäume noch nicht bekannt ist, an welche Position die Bäume letztendlich gesetzt und zunächst z. B. an der gleichen Stelle generiert werden müssen. Das bedeutet wiederum, dass bei nicht blockierendem Rendern und großen Projekten wie Air zunächst die Bäume überlagernd gerendert werden und dies auch durchaus für einige Sekunden für den Nutzer sichtbar ist, bevor sich die Bäume verteilen.

Bei Air dauert die Generierung der 40 Bäume im Unity Editor auf einer Hardware mit 2.3 GHz Intel Core i7 CPU und 16 GB 1600 MHz DDR3 RAM bei 10 Messungen zwischen 6,5 und 6,8 Sekunden. Auf der HoloLens stehen keine genauen Messdaten zur Verfügung, die Generierung dauert aber noch etwas länger.

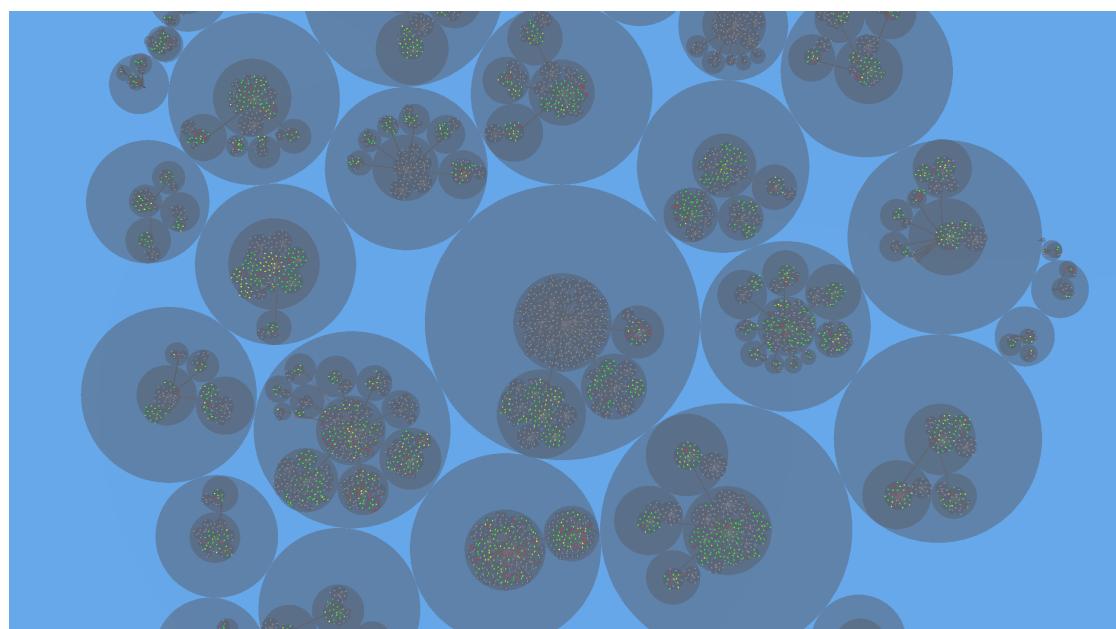
Soll vermieden werden, dass der Nutzer zunächst die überlagernde Generierung mit ansieht, dürften die Bäume erst sichtbar werden, nachdem alle Bäume fertig generiert und verteilt worden sind.

Ein weiterer zu beachtender Punkt ist die Leistungsfähigkeit der HoloLens. Für Air werden über 12 000 3D-Objekte gleichzeitig gerendert. Die aktuelle Entwickler-Version der HoloLens ist für diese Summe an zu rendernden Objekten nicht ausgelegt. Bei Generierung des Waldes wird das Bild zunächst kurz schwarz und stockt während und

#### 4 Modellierung und Layout des Waldes



(a) Seitenansicht



(b) Vogelperspektive mit visualisierten Circle-Packing

**Abbildung 4.8** Layout des Waldes

auch nach fertiger Generierung merklich. Auch bei kontinuierlichen Interaktionen wie dem Skalieren reagiert die HoloLens erst nach beendeter Eingabe. Die Nutzung von so großen Software-Projekten wie Air ist demnach mit der aktuellen Entwickler-Version der HoloLens möglich, aber nicht besonders angenehm. Dagegen laufen kleinere Projekte oder ein Ausschnitt von Air mit z. B. nur 5 statt den eigentlichen 40 Bäumen flüssig.

## 4.5 Abbildung von Daten auf die Struktur

In Abbildung 4.8 waren bereits unterschiedliche Blattfarben zu sehen. Mit der Generierung des Waldes ist das Schwierigste geschafft. Das Färben der Blätter und das Anpassen der Dicke der Kanten und Wurzeln ist dagegen problemlos möglich.

### Farbe der Blätter

In den Abbildungen 4.8 - 4.11 wird mit der Blattfarbe die Testabdeckung im Projekt Air visualisiert. Damit ist es möglich, eine beliebige Metrik darzustellen, die im Software-Meta-Modell vorliegt (Akz. 1.1 und 4.1). Die Metrik kann in einem stufenlosen Farbverlauf dargestellt werden, im Beispiel der Testabdeckung von Rot (0%) über Gelb (50%) bis hin zu Grün (100%). Klassen, bei denen im Beispiel-Datensatz keine Testabdeckung verfügbar ist, erscheinen in Grau.

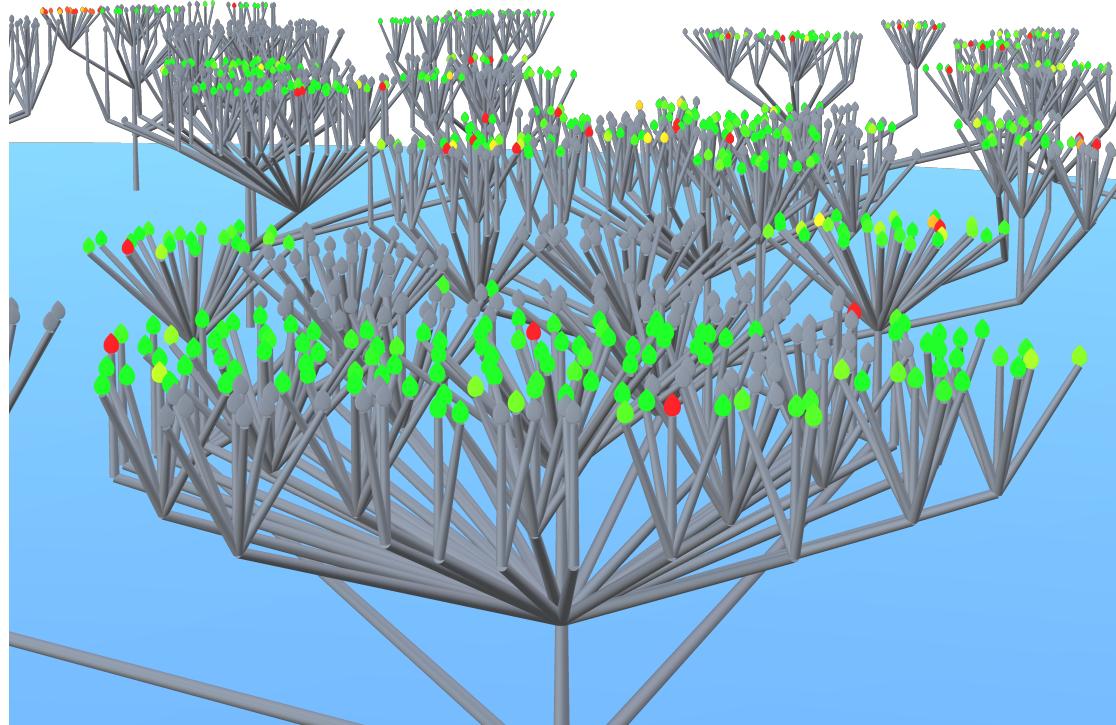
In Abbildung 4.9 ist schön zu sehen, dass die Testabdeckung im Durchschnitt sehr gut ist, obwohl immer wieder Klassen dabei sind, bei denen die Testabdeckung gering ist. Im linken oberen Eck der Abbildung ist ein Ast zu sehen, bei dem kein einziges Blatt eine gute Testabdeckung vorweisen kann. Ausreißer der aktuellen Metrik und ganze Hotspots sind demnach im Prototyp gut zu erkennen (Akz. 1.2 und 4.2).

### Dicke der Kanten

Die Dicke der Kanten lässt sich ebenfalls leicht anpassen. Die Standard-Dicke kann je nach Anzahl der direkten Verbindungen mit einer geeigneten mathematischen Funktion erhöht werden. Da der Beispiel-Datensatz von Air jedoch keine Verbindungen wie Abhängigkeiten aufweist, wird in Abbildung 4.10 ein Baum gezeigt, der mit zunehmender Höhe der Knoten entsprechend dickere Kanten besitzt. Dies ist für richtige Verbindungen wie Abhängigkeiten natürlich wenig realistisch, zeigt aber das Potential dieser Darstellung. Die Beschaffung und der Transfer von Daten über Abhängigkeiten in Air in das Software-Meta-Modell ist keine triviale Aufgabe und muss außerhalb dieser Arbeit bewältigt und vor allem auch automatisiert werden.

Die Akzeptanzkriterien 2.3 und 4.3 werden demnach im High-Fi-Prototyp bereits technisch erfüllt. Es muss aber mit realistischen Beispieldaten noch evaluiert werden, welche mathematische Funktion für die Dicke der Kanten geeignet ist.

#### 4 Modellierung und Layout des Waldes



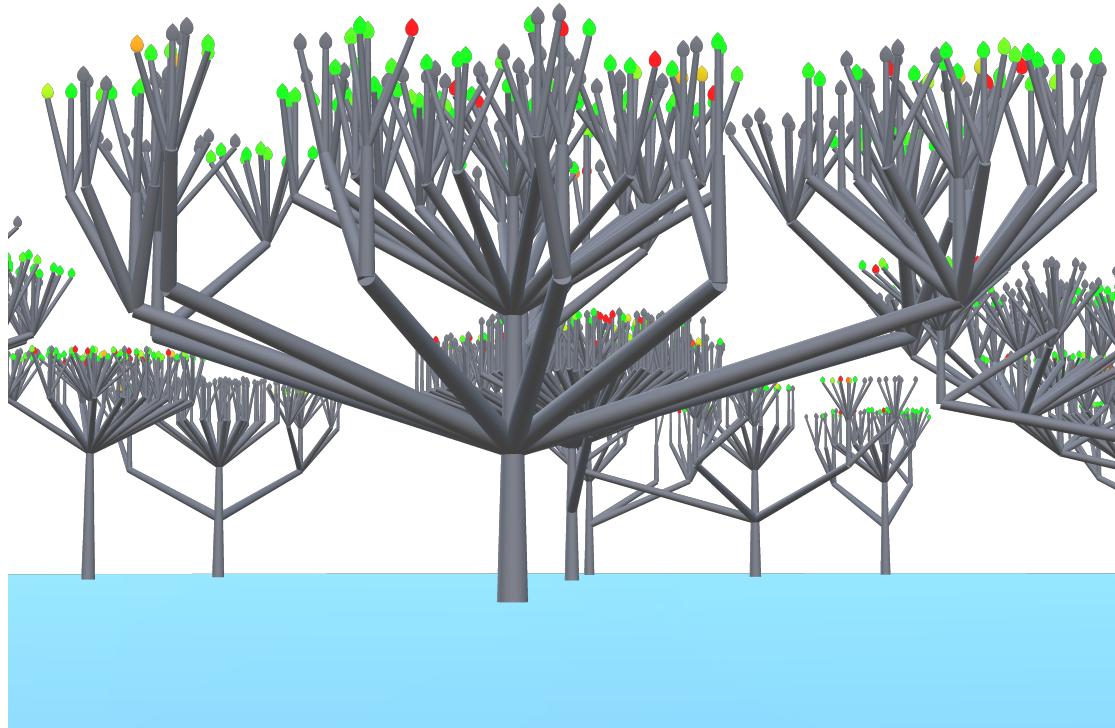
**Abbildung 4.9** Farbe der Blätter und Erkennung von Ausreißern

#### Wurzeln

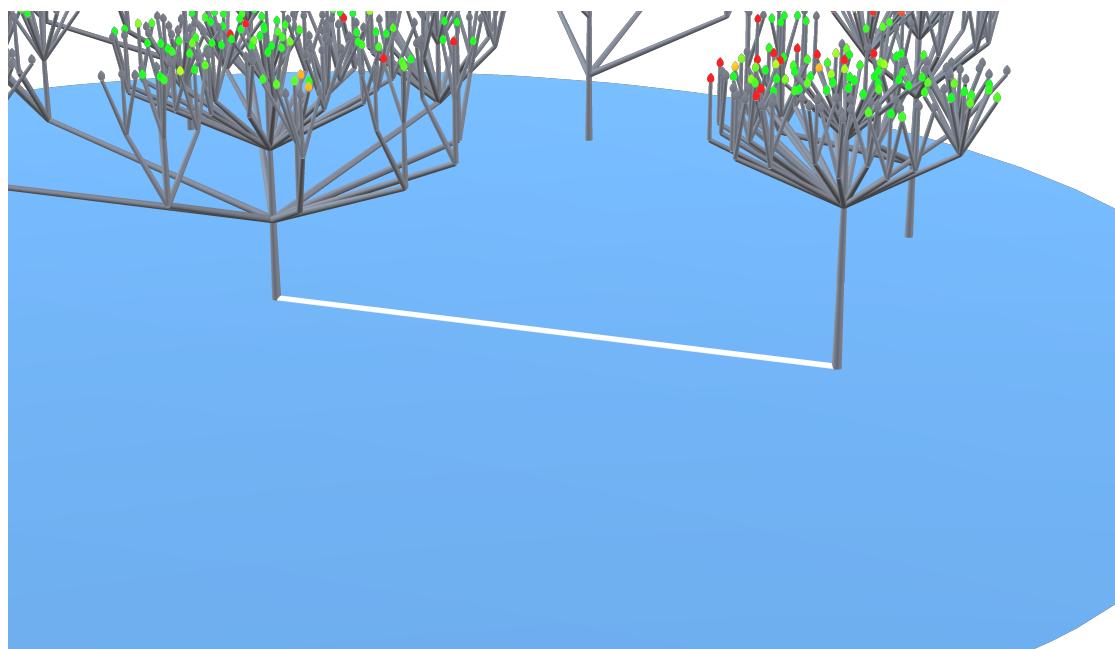
Bislang haben wir nur Wälder ohne Wurzeln gesehen. Eine realistische Darstellung vom Wurzelgeflecht ist aufgrund der unzureichenden Beispieldaten ebenfalls nicht möglich. Jedoch werden Wurzeln im High-Fi-Prototyp prinzipiell unterstützt. In Abbildung 4.11 ist eine Beispiel-Wurzel zu sehen, die zwei Bäume miteinander verbindet. Für eine Wurzel wird als 3D-Objekt ein Zylinder verwendet, sodass Wurzeln auch von der Seite gut erkennbar sind.

Die Struktur des Waldes kann nun generiert, die Blätter gefärbt und die Dicke der Kanten und Wurzeln angepasst werden. Was fehlt, ist die Interaktion. Dazu wird im nächsten Kapitel zunächst auf Grundlagen der Interaktion in der AR eingegangen, um diese dann in Kapitel 6 bei der Interaktion in CodeLeaves anzuwenden.

#### 4.5 Abbildung von Daten auf die Struktur



**Abbildung 4.10** Unterschiedliche Dicke der Kanten



**Abbildung 4.11** Beispiel einer Wurzel



# 5 Grundlagen der Interaktion in der AR

## 5.1 Eingabemöglichkeiten in der AR und mit der HoloLens

In den frühen 70er-Jahren wurden erste Modelle der Computermaus entwickelt und mit Apple's Lisa aus dem Jahr 1983 wurde sie zum Markterfolg. Heute ist die zeigerbasierte Interaktion mit Computern nicht mehr wegzudenken.

Neben der zeigerbasierten Eingabe hat sich mit dem Einzug von Smartphones und Tablets auch die touchbasierte Eingabe etabliert. Diese Art der Eingabe kann für monitor-basierte AR verwendet werden. Ein Beispiel dafür sind iOS Geräte, die Apple's AR-Kit unterstützen.

Für die AR mit *Head-Mounted-Displays* (HMD) Devices, die wie die HoloLens vom Nutzer als Brille getragen werden, ergeben sich neue Möglichkeiten, mit Anwendungen zu interagieren. Dies sind folgende:

1. Blickrichtung
2. Gesten
3. Sprache
4. Controller

### Blickrichtung

Bei der HoloLens ist die *Blickrichtung* (engl.: *gaze*) ein elementarer Bestandteil der Bedienung [6]. Die Blickrichtung wird im Normalfall mithilfe eines Cursors visualisiert, der der Kopfbewegung des Nutzers folgt und so immer zentral im Sichtfeld der HoloLens bleibt. Der Cursor wird so positioniert und orientiert, dass er sich auf der Oberfläche des Objekts befindet, das sich in Blickrichtung befindet und dem Nutzer am nächsten ist. Das Ziel der Blickrichtung nennen wir *fokussiertes Objekt*.

### Gesten

Nachdem bei HMD-Devices Hologramme im realen Raum platziert werden, ist Gestensteuerung eine intuitive Eingabemöglichkeit. Beispielsweise muss die Möglichkeit der Auswahl, Platzierung, Skalierung oder Rotation von Hologrammen ermöglicht werden.

Allgemein kann zwischen zwei Arten von Gesten unterschieden werden [5].

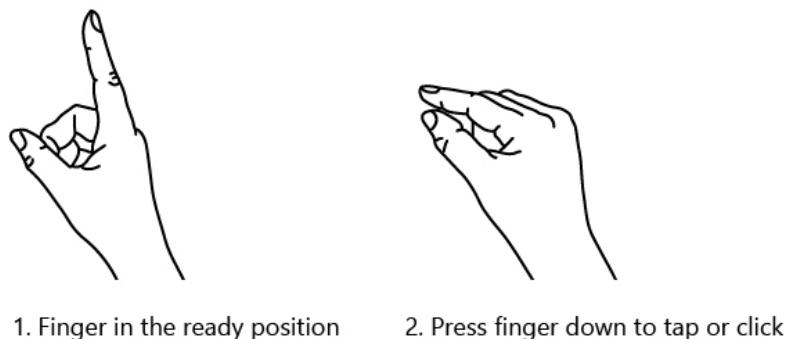
## 5 Grundlagen der Interaktion in der AR

**Diskrete Gesten** sind solche, in denen die Ausführung der Geste einen binären Status besitzt. D.h. die Ausführung der Geste ist die Information selbst und trägt keine weiteren Informationen. Diese Gesten lassen sich mit einem Klick einer Computermaus vergleichen.

**Kontinuierliche Gesten** sind solche, bei denen das Ausmaß der Geste eine Rolle spielt. Das Ausmaß bestimmt die Größe der Eingabe. Vergleichen wir diese Gesten mit einer Computermaus, wäre das die Mausbewegung.

Theoretisch sind beliebig viele Gesten denkbar. Zum Beispiel könnte für das Rotieren von Objekten die Rotation einer Hand verwendet werden. Mit Produkten wie z.B. Kinetic<sup>1</sup> können solche Gesten auch erkannt werden. Bei der HoloLens sind die unterstützen Gesten jedoch stark begrenzt. Es existieren insgesamt fünf verschiedene Gesten, die von der HoloLens als solche identifiziert werden können. Die sogenannte *Bloom*-Geste ist für das Windows-Menü reserviert. Sie wird ausgeführt, indem der Nutzer alle Fingerspitzen zusammen führt und dann die Hand öffnet und eine aufgehende Blume imitiert.

Als weitere diskrete Geste kann der *Air-Tap* verwendet werden. Dieser ist in Abbildung 5.1 veranschaulicht. In Verbindung mit der Blickrichtung können so Hologramme angeklickt werden.



**Abbildung 5.1** Air-Tap [5]

Neben dem Air-Tap werden auch der *Double-Tap* und die *Tap-and-hold*-Geste erkannt.

Mit der *Manipulation*-Geste stellt Microsoft eine kontinuierliche Geste zur Verfügung. Wird ein Air-Tap gehalten und die Hand anschließend bewegt, so kann als Eingabe die Positionsänderung der Hand verwendet werden. Damit ist beispielsweise ein Drag-and-Drop von Hologrammen möglich.

Die HoloLens kann zwischen linker und rechtem Hand unterscheiden. Das bedeutet, dass auch Interaktion mit der Verwendung von beiden Händen denkbar sind.

<sup>1</sup><https://developer.microsoft.com/de-de/windows/kinect>

## 5.2 Herausforderungen

### Sprache

Neben den Gesten kann auch durch Sprachsteuerung mit Hologrammen interagiert werden. In einer Applikation können beliebige Befehle definiert werden, die dann von der HoloLens automatisch erkannt werden.

Da die Interaktion mit Gesten auf Dauer für die Arme ermüdend sein kann, bietet sich die Sprachsteuerung zur Unterstützung der Gesteuerung an. Beide Eingabemethoden können auch gut miteinander kombiniert werden. So kann z. B. mithilfe von Sprachbefehlen zwischen unterschiedlichen Interaktionsmodi umgeschalten werden. Dadurch können unterschiedliche Interaktionen wie Skalieren und Rotieren mit der gleichen Geste durchgeführt werden.

### Controller

Die vierte Möglichkeit der Interaktion mit HMD-Devices sind Controller. Die meisten VR-Headsets sind nur mit solchen Controllern zu bedienen. In der aktuellen Version unterstützt die HoloLens jedoch keine Controller. Bei anderen Windows Mixed Reality Geräten wie die *Immersive* Headsets von Acer und HP<sup>2</sup> können mit den *Motion* Controllern, ähnlich wie bei Gaming Konsolen, mit verschiedenen Tasten unterschiedliche Eingaben getätigt werden. Bei der HoloLens steht lediglich der *Clicker* zur Verfügung, ein kleines Bluetooth-Gerät, das durch physisches Klicken den Air-Tap ersetzen kann. Bei dauerhafter Eingabe von Air-Tap-Gesten ist der Einsatz des Clicker angenehm, da der Nutzer die Hand nicht im Sichtfeld der HoloLens halten muss.

## 5.2 Herausforderungen

Bei der Entwicklung von AR-Applikationen sind einige Herausforderungen zu beachten, die auch für CodeLeaves von Relevanz sind. Im Zuge der Entwicklung von der HoloLens-Anwendung HoloStudio hat Senior Holographic Designer M. Ghaly in [20] eine Case Study veröffentlicht. Mitunter traten die im Folgenden betrachteten Herausforderungen auf.

### Hologramme außerhalb des Sichtfelds

Da Hologramme im ganzen Raum, in dem sich der Betrachter befindet, platziert werden können, ist nicht immer gewährleistet, dass sie sich auch im Blickfeld des Betrachters befinden. Besonders bei der aktuellen Entwickler-Version der HoloLens ist das Sichtfeld, in dem Hologramme angezeigt werden können, mit rund 30 Grad [12] gering.

Eine Variante, dies zu umgehen, ist die Verwendung von *Tag-along*-Objekten. Diese Objekte bleiben immer im Blickfeld des Betrachters. Schaut der Nutzer in eine andere Richtung, wird das Objekt neu positioniert, sodass es wieder ins Blickfeld des Nutzers gelangt. Innerhalb des Blickfelds bleiben Tag-along-Objekte aber stationär, sodass der

---

<sup>2</sup>[https://developer.microsoft.com/en-us/windows/mixed-reality/immersive\\_headset\\_hardware\\_details](https://developer.microsoft.com/en-us/windows/mixed-reality/immersive_headset_hardware_details)

## 5 Grundlagen der Interaktion in der AR

Nutzer mit der Blickrichtung weiterhin mit ihnen interagieren kann. Das Windows-Startmenü der HoloLens ist ein solches Tag-along-Objekt.

In der Fallstudie aus [20] wurde festgestellt, dass die Probanden sich von solchen Tag-along-Objekten in manchen Fällen bedrängt fühlen und deshalb sollten diese sparsam verwendet werden.

Eine andere Möglichkeit ist es, auf Hologramme, die sich außerhalb des Sichtfelds befinden, gezielt aufmerksam zu machen. Das kann mit verschiedenen Techniken erreicht werden. Eine davon ist die Verwendung von *Spatial-Sound*, was bedeutet, dass Geräusche ebenfalls im Raum platziert werden können und vom Nutzer dreidimensional mit Richtung und Entfernung wahrgenommen werden können. Dies bietet sich vor allem bei Spielen mit bewegten Charakteren an. Bei statischem Inhalt wie CodeLeaves sind Geräusche nicht besonders intuitiv.

Es kann auch mit zusätzlichen Objekten gearbeitet werden, die den Nutzer darauf hinweisen, wo seine Aufmerksamkeit erfordert wird. Solche Hilfs-Objekte werden in der HoloLens-Entwicklung *Direction-Indicators* genannt. Diese können unterschiedliche Gestalt annehmen. Bei anderen HoloLens-Applikationen sind Pfeile, Licht oder Denkblasen im Einsatz.

### UI von anderen Objekten verdeckt

Ein Problem, das in vielen HoloLens-Applikationen auftritt, ist, dass UI-Elemente von anderen Hologrammen, die näher am Nutzer positioniert sind, verdeckt werden können.

Das Problem ist in Abbildung 5.2 illustriert. Darin wird in CodeLeaves ein Button eines Kontextmenüs (siehe Abschnitt 6 von einem Stamm verdeckt.



**Abbildung 5.2** Verdeckung von UI-Elementen

Analog zu 2D Pop-ups ist es möglich, UI-Elemente vor allen anderen Elementen zu platzieren. In 3D bedeutet das, dass das UI-Element vor dem Hologramm erscheinen muss, dass sich am nächsten am Nutzer befindet. Dabei ist zu beachten, dass durch den räumlichen Versatz der Bezug zum fokussierten Objekt nicht verloren gehen darf.

### 5.3 Technische Umsetzung von Interaktionen mit reaktiver UI

Ein alternativer Lösungsansatz ist der *Shine-through*-Effekt, wie er bei HoloStudio Verwendung findet und in Abbildung 5.3 veranschaulicht wird. Dabei bleiben UI-Elemente räumlich bei dem fokussiertem Objekt, jedoch scheinen sie durch davor liegende Hologramme durch. Diese Variante ist auch für anzuzeigenden Text gut geeignet.

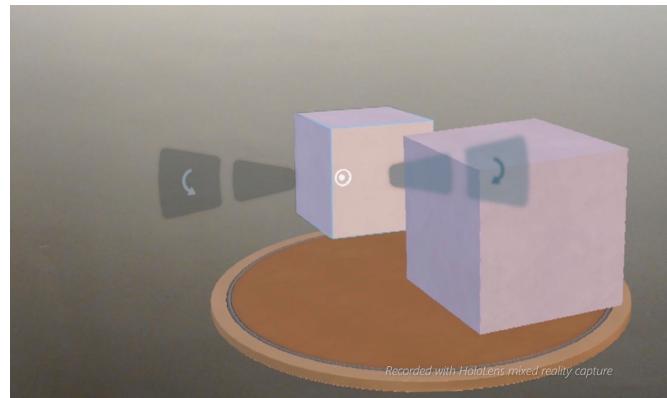


Abbildung 5.3 Shine-through-Effect von UI-Elementen in HoloStudio [20]

## 5.3 Technische Umsetzung von Interaktionen mit reaktiver UI

### Motivation

Eine der offensichtlichen Interaktionen mit CodeLeaves ist das Anklicken von Knoten. Der Nutzer erwartet damit nähere Informationen zu einem Paket oder einer Klasse zu erhalten. Betrachten wir zunächst eine sehr einfaches Szenario:

*Wenn der Nutzer auf ein Blatt klickt, soll daraufhin der Name der repräsentierten Klasse über dem Blatt erscheinen.*

Diese Interaktion ist in Abbildung 5.4 veranschaulicht.

Bei der Generierung des Baumes wird zu jedem Blatt ein Label hinzugefügt, in dem beliebiger Text gespeichert werden kann. Das Anzeigen des Namens ist demnach kein Problem und könnte wie folgt implementiert werden:

**Listing 5.1:** Direkte Manipulation (Negativbeispiel)

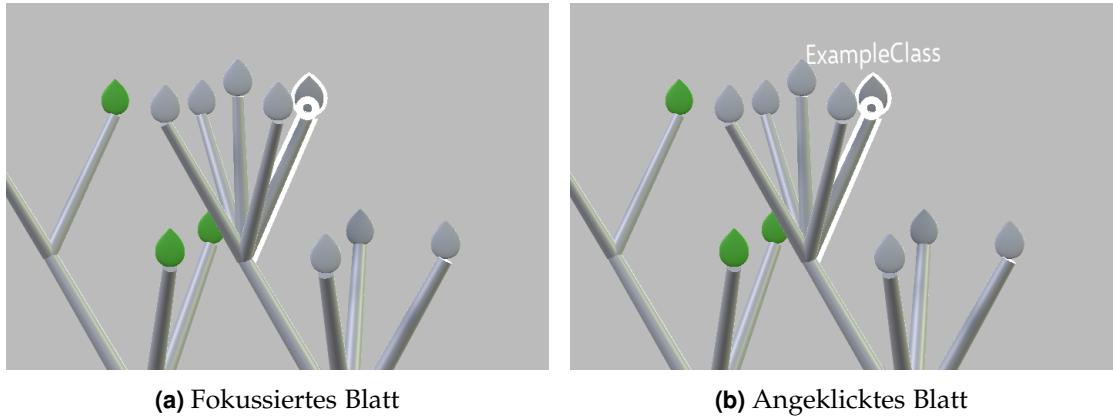
```
1  public class NodeInputHandler : MonoBehaviour, IInputClickHandler
2  {
3      public void OnInputClicked(InputClickedEventData eventData)
4      {
5          // Get label of currently clicked node
6          var labelObject = GetLabelObject(gameObject);
7      }
}
```

## 5 Grundlagen der Interaktion in der AR

```

8           // Toggle active
9             labelObject.SetActive(!labelObject.activeSelf);
10            }
11        }

```



**Abbildung 5.4** Beispielhafte Interaktion mit einem Blatt

Mit diesem Ansatz könnte also auf das Ereignis des Air-Taps auf einfachste Weise reagiert werden. Wird das Blatt angeklickt, wird das Label-Objekt des Blattes aktiviert und der Name wird damit sichtbar. Entsprechend kann bei erneutem Klick das Label-Objekt wieder deaktiviert werden. Wird die Anwendungslogik aber komplexer, kommt dieser Ansatz schnell an seine Grenzen.

Eventuell könnte sich z. B. folgende Einstellung in der Applikation als sinnvoll erweisen:

*Der Nutzer kann einstellen, ob bei einem Klick auf ein Blatt der Klassename oder der Zahlenwert der gerade visualisierten Metrik angezeigt wird.*

Das eigentliche Problem bei der beschriebenen Situation ist, dass Anwendungslogik und UI-Logik zu eng gekoppelt wurden. Diese Art enger Kopplung sollte tunlichst vermieden werden. Der Programmcode in der UI-Schicht, der auf das tatsächliche Event reagiert, das durch den Nutzer ausgelöst wurde, sollte keine fachlichen Entscheidungen treffen, sondern die Anwendungslogik entsprechend benachrichtigen. Die Anwendungslogik hingegen sollte sich nicht mit der konkreten Aktualisierung der UI beschäftigen, sondern lediglich die Veränderung des Anwendungszustands im Verlauf der Zeit steuern. Dieser Zustand wiederum steht sozusagen für sich selbst. Die UI muss über Änderungen am Zustand in geeigneter Form informiert werden und sich selbst entsprechend aktualisieren.

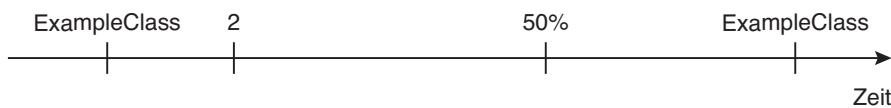
Ein besonders in UI-lastigen Anwendungen bekanntes und erfolgreiches Pattern, mit dem sich der zuvor genannte Ablauf umsetzen lässt, ist das der reaktiven Programmierung. A. Stoltz beschreibt es in [40] als „Programmierung mit asynchronen

### 5.3 Technische Umsetzung von Interaktionen mit reaktiver UI

Datenströmen.“ und trifft damit den Kern. Es geht darum, auf Änderungen von Werten zu reagieren, die durch *Streams* propagiert werden.

Streams sind nichts anderes als *Observables*, die bei jeder Änderung ihrer Werte allen registrierten *Subscribers* den neuen Wert mitteilen.

Betrachten wir in Abbildung 5.5 konkret das Beispiel des Textes, der im Label eines Blattes steht.



**Abbildung 5.5** Datenstrom eines Labels für reaktive Programmierung

Über die Zeit hinweg kann sich durch die Anwendungslogik der darzustellende Text verändern. Im Beispiel soll der Klassename, dann vielleicht Laufzeitfehler, Testabdeckung und dann wieder der Klassename dargestellt werden.

Bei der Generierung des Baumes muss dann lediglich der Label-Wert observiert und der Text entsprechend gesetzt werden.

#### Verwendung in Unity

In Unity steht für reaktive Programmierung das Open-Source-Projekt *UniRx*<sup>3</sup> zur Verfügung, das die Funktionalität der reaktiven .NET-Library Rx für Unity zugänglich macht. UniRx implementiert z. B. *ReactiveProperties*, mit denen eine einfache Verwendung des Observer-Pattern möglich ist. Darüber hinaus bietet die API hilfreiche Stream-Funktionen wie *filter*, *map*, *merge* oder *join*.

In unserem Beispiel von oben wird der Text eines Blattes des UI-Models zu einer *ReactiveProperty* und bei jeder Änderung wird der Text des Labels angepasst. Auch das Aktivieren des Labels wird über eine *ReactiveProperty* realisiert. So ist zum Beispiel das Deaktivieren von allen Labels gleichzeitig leicht möglich.

In Listing 5.2 wird gezeigt, wie eine *ReactiveProperty* für den Status *isSelected* eines *UiNodes* definiert wird. Im *TreeBuilder* (Zeile 9) wird das Label bei Aktualisierung von *isSelected* entsprechend aktiviert.

**Listing 5.2:** Observieren von Werten

```
1  public abstract class UiNode
2  {
3      public ReactiveProperty<bool> isSelected { get; set; }
4      ...
5  }
6
7  public class TreeBuilder {
8      ...
9 }
```

<sup>3</sup><https://github.com/neuecc/UniRx>

## 5 Grundlagen der Interaktion in der AR

```
9     node.isSelected.Subscribe(label.SetActive);
10    ...
11 }
12
13 public class NodeInputHandler : MonoBehaviour, IInputClickHandler
14 {
15     public void OnInputClicked(InputClickedEventData eventData)
16     {
17         InteractionManager.HandleNodeClick(GetNodeId(gameObject));
18     }
19     ...
20 }
```

Im `NodeInputHandler` wird nun die UI nicht mehr direkt manipuliert, sondern es wird eine Funktion des `InteractionManagers` aufgerufen, die sich darum kümmert, das UI-Model entsprechend zu aktualisieren.

# 6 Interaktion mit CodeLeaves

CodeLeaves kann ohne Interaktion die Struktur einer Software, eine Metrik und einen Verbindungs-Typ zwischen Klassen darstellen. Damit kann bereits ein Überblick über die Software geschaffen werden. Der bedeutende Mehrwert von CodeLeaves wird aber erst erreicht, wenn mit dem Software-Wald interagiert werden kann. Die intuitive Bedienung von CodeLeaves ist essentiell für ein tieferes Verständnis der visualisierten Informationen.

In diesem Kapitel wird deswegen ein Konzept entwickelt, mit dem es möglich ist, alle Informationen zugänglich zu machen, die CodeLeaves zur Verfügung stellen kann. Es besteht jedoch kein Anspruch auf vollständiges Interaktionsdesign nach [21] oder [9]. Vielmehr soll im Zuge dieser Arbeit Prototyping betrieben werden, durch das CodeLeaves bedienbar ist und der Mehrwert von CodeLeaves beurteilt werden kann.

Für eine weiterführende Realisierung von CodeLeaves müsste das Interaktionskonzept in einer Evaluations-Phase durch Nutzerbefragungen validiert werden und bei Bedarf verfeinert oder angepasst werden.

## 6.1 Basisinteraktionen

Grundlegend muss der gesamte Wald einige *Basisinteraktionen* unterstützen (Akz. 6.1 - 6.3). Diese sind:

- Platzieren
- Skalieren
- Rotieren

Das Platzieren ist von Anfang an notwendig, um den Wald im Raum dahin zu setzen, wo es für den Nutzer am meisten Sinn macht. Bequemerweise kann das z. B. auf einem Tisch sein oder – wenn ein solcher nicht zur Verfügung steht – auch auf dem Fußboden.

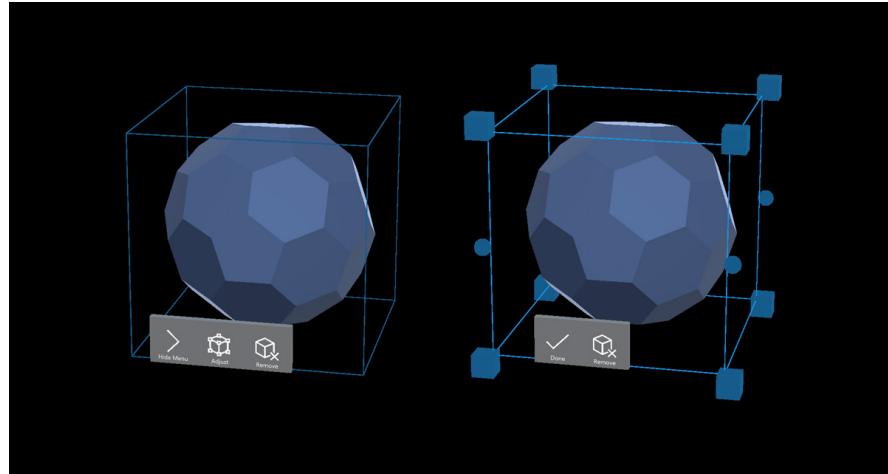
Gerade bei großen Softwaresystemen kann der Wald sehr groß werden. Daher muss die Größe des Waldes auf die Gegebenheiten der Räumlichkeiten angepasst werden können. Eine Möglichkeit zur Skalierung der Anzeige ist demzufolge unerlässlich.

Ist der Wald einmal platziert und skaliert, möchte der Nutzer sich den Wald von allen Seiten anschauen können. Er kann natürlich darum herumlaufen, aber vielleicht reicht der Platz dazu nicht aus oder ein Drehen des Waldes ist einfach bequemer.

## 6 Interaktion mit CodeLeaves

### Bounding-Box

Das Repository „Mixed Reality Design Labs“<sup>1</sup> bietet für genau diese Interaktionen eine Lösung. Die *Bounding-Box*, wie sie in Abbildung 6.1 zu sehen ist, rendert einen 3D-Rahmen um das zu manipulierende Objekt. Das Platzieren des Objekts kann mit Drag-and-Drop im gesamten Bereich der Bounding-Box erreicht werden. Mit den kleinen Würfeln an den Ecken der Bounding-Box kann skaliert werden und die kleinen Kugeln auf den vertikalen Kanten der Bounding-Box sind für das Rotieren vorgesehen.



**Abbildung 6.1** Bounding-Box als Option für die Basisinteraktionen [10]

Dieser Ansatz hat den Vorteil, dass dem Nutzer ein Interface angeboten wird, das ihm aus dem Zweidimensionalen wohl bekannt ist. In vielen Bereichen wie Text- und Bild- oder Grafikverarbeitung ist die Bounding-Box Standard. In die Dreidimensionalität übertragen kennt der Nutzer das Prinzip und weiß, was er für eine gewünschtes Ziel zu tun hat.

### Interaktionsmodi

Für CodeLeaves ist dieser Ansatz jedoch aus einem einfachen Grund schwierig. Der Wald muss nicht zwangsläufig in seiner Gänze im Blickfeld des Nutzers sein. Angenommen, der Nutzer interessiert sich vor allem für einen bestimmten Baum, möchte er diesen genauer betrachten und vergrößert den Wald soweit, dass andere Bäume aus dem Sichtfeld verschwinden. Möchte er dann den Wald wieder verkleinern, müsste er erst soweit zurück gehen, bis er die Bounding-Box nutzen kann. Das ist aber aufgrund der Räumlichkeiten vielleicht gar nicht möglich und wenn doch, nicht besonders benutzerfreundlich.

Eine Alternative muss daher gefunden werden. Es stellt sich die Frage, was immer im Sichtfeld des Nutzers ist und den Wald für den Nutzer als Ganzes repräsentieren kann. Die Antwort auf diese Frage liegt auf der Hand – der Waldboden. Dieser ist auch

<sup>1</sup>[https://github.com/Microsoft/MRDesignLabs\\_Unity](https://github.com/Microsoft/MRDesignLabs_Unity)

bei näherer Betrachtung von Ausschnitten des Waldes immer präsent und da alles aus ihm wächst, ist eine Interaktion mit ihm für die Basisinteraktionen intuitiv.

Damit steht das Ziel der Interaktion fest, jedoch können mit der begrenzten Anzahl an Gesten nicht alle Basisinteraktionen bedient werden. Daher wird bei einem Air-Tap ein Kontextmenü dargestellt, mit dem der Nutzer zwischen den drei verschiedenen Basisinteraktionen wählen kann. Bei der Auswahl der Modi Skalieren und Rotieren kann der Nutzer mithilfe der Manipulation-Geste den Wald entsprechend manipulieren. Die Ausführung dieser Geste wird näher im Abschnitt 6.3 beschrieben.

### Interaktion mit Spatial-Mapping

Für den Platzierungs-Modus bietet sich für die HoloLens die Nutzung des *Spatial-Mapping* an. Die HoloLens kann den Raum, in dem sie sich befindet, zur Laufzeit zu erfassen und einer Applikation zur Verfügung zu stellen [7]. Mit diesen Informationen ist es möglich, ein Spatial-Mapping-Mesh darzustellen, das über die reale Oberfläche des Raumes gelegt wird. Damit können Hologramme mit der Oberfläche des Raumes kollidieren und entsprechend der Umgebung platziert werden. So kann der Nutzer den Wald komfortabel auf einem Tisch oder auf dem Boden platzieren.

Bei aktivem Platzieren des Waldes wird das Spatial-Mapping-Mesh angezeigt, so dass der Nutzer sieht, wo genau der Waldboden mit der Umgebung kollidiert. Die Platzierung selbst wird nicht mit der Manipulation-Geste ausgeführt, sondern mithilfe der Blickrichtung. Der Wald folgt der Blickrichtung des Nutzers und wird immer dort platziert, wo die Blickrichtung das Spatial-Mapping-Mesh trifft. Die Manipulation-Geste ist besonders für die Eingabe einer Differenz geeignet. Für die Eingabe einer bestimmten Richtung, die für Kollision mit dem Spatial-Mapping-Mesh benötigt wird, ist die Blickrichtung die bessere Wahl. Zudem ist der „Transport“ des Waldes auf diese Weise auch über größere Strecken komfortabel möglich, da die Hand des Nutzers nicht dauerhaft oben gehalten werden muss, wie es bei der Manipulation-Geste nötig ist.

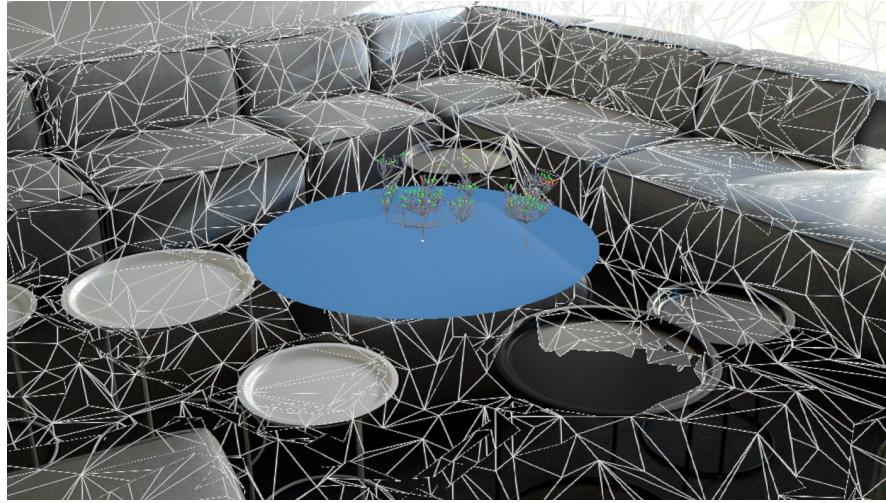
Der Platzierungs-Modus wird durch einen Air-Tap auf den Wald beendet. Der aktive Modus mit dem Spatial-Mapping-Mesh kann in Abbildung 6.2 betrachtet werden.

## 6.2 Das Kontextmenü

Das Kontextmenü, mit dem die verschiedenen Interaktionsmodi für den gesamten Wald ausgewählt werden können, lässt sich nicht nur für die Interaktion mit dem Waldboden anwenden. Auch für andere fokussierte Objekte ist eine vom Kontext abhängige Auswahl von möglichen Funktionen allgemeingültig sinnvoll. In herkömmlichen 2D Benutzeroberflächen ist dies mit einem Rechtsklick zu vergleichen. In 3D kann um den Punkt, den der Nutzer beim Aufruf des Kontextmenüs fokussiert hatte, das Kontextmenü beispielsweise in Form von Buttons in einer kreissegmentförmigen Anordnung dargestellt werden.

Die kreissegmentförmige Anordnung der Buttons hat den Vorteil, dass der Interaktionspunkt für das Öffnen des Kontextmenüs im Zentrum des Kreises ist und daher eine Verbindung mit dem fokussierten Objekt schafft. Darüber hinaus sind alle Buttons

## 6 Interaktion mit CodeLeaves



**Abbildung 6.2** Spatial Mapping beim Platzieren des Waldes

vom Cursor gleich weit entfernt und daher schnellstmöglich mithilfe der Blickrichtung erreichbar.

Die Distanz des Kontextmenüs ist den Umständen entsprechend anzupassen. Ist der fokussierte Punkt bereits näher als die Distanz, in der eine Darstellung von Interaktions-Elementen empfohlen wird<sup>2</sup>, sollte das Kontextmenü nicht noch näher am Nutzer platziert werden. Bei größerem Abstand sollte der Abstand zum Kontextmenü auf die empfohlene Distanz reduziert werden, sodass der Nutzer noch komfortabel mit dem Menü interagieren kann. Auf diese Weise ist die Überdeckung des Kontextmenüs durch andere Elemente sehr unwahrscheinlich.

In dem Fall, dass das Kontextmenü direkt an der fokussierten Position erscheint, muss das Kreissegment des Kontextmenüs so angepasst werden, dass die Buttons nicht in das fokussierte Objekt hinein ragen. Für den Waldboden ist daher ein nach oben orientiertes Kreissegment eine gute Wahl. Würde aber zum Beispiel mit einer Wand interagiert werden, sollte das Kontextmenü von der Wand weg orientiert sein.

Das Kontextmenü für den Waldboden im High-Fi-Prototyp ist in Abbildung 6.3 zu sehen.



**Abbildung 6.3** Kontextmenü für den Waldboden im High-Fi-Prototyp

In Abbildung 6.3b ist zu erkennen, dass der fokussierte Button räumlich hervorge-

<sup>2</sup>Die Distanz, in der Microsoft empfiehlt, Hologramme zu platzieren, liegt bei 2,0 Metern [6].

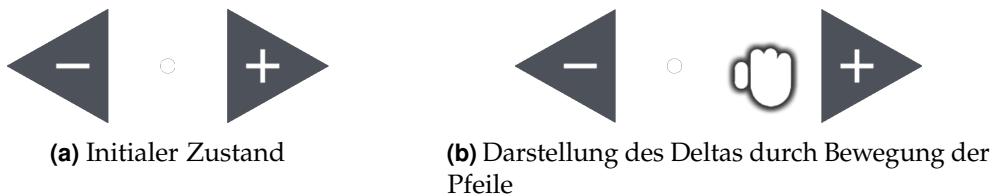
hoben ist. Dies ist eine vor der AR noch nicht dagewesene Möglichkeit, dem Nutzer ein schon fast haptisches Feedback zu geben. Angenommen,  $\vec{g}$  ist der Normalvektor der Blickrichtung des Nutzers, dann wird ein Button beim Fokussieren um  $-\vec{g} \cdot x$  transformiert, wobei  $x$  ein beliebiger Faktor ist. Bei der Ausführung der Air-Tap-Geste wird beim Schließen der Finger der Button um  $\vec{g} \cdot x$  transformiert, sodass er wieder in seine Ausgangsposition gelangt. Sobald der Nutzer mit dem Öffnen der Finger den Air-Tap beendet, wird der Button wieder um  $-\vec{g} \cdot x$  transformiert. Verlässt der Fokus den Button, wird er ebenfalls wieder in seine Ausgangsposition versetzt.

Mit diesem Prinzip kann der Nutzer Buttons räumlich „eindrücken“. Das Prinzip findet auch beim Windows-Menü der HoloLens Anwendung und zieht sich durch die gesamte UI des High-Fi-Prototyps.

### 6.3 Manipulation-Indicators

Mit dem Kontextmenü des Waldbodens kann zwischen den beiden Interaktionsmodi umgeschaltet werden, die mit der Manipulation-Geste ausgeführt werden. Bei der Manipulation-Geste weiß der Nutzer aber ohne weitere Hilfe nicht, wie er seine Hand bewegen muss, um die gewünschte Manipulation zu erreichen. Der Nutzer weiß nicht, in welcher Richtung der Wald vergrößert und in welcher er verkleinert wird. Zudem sind mehrere Achsen wie horizontal, vertikal oder diagonal denkbar. Deswegen ist es unerlässlich, dem Nutzer entsprechend Feedback zu geben. In CodeLeaves wird dies mit den *Manipulation-Indicators* erreicht. Diese tauchen auf, sobald der Air-Tap gehalten wird und indizieren die möglichen Richtungen und deren Auswirkungen.

In Abbildung 6.4 sind Manipulation-Indicators für das Beispiel des Skalierens dargestellt.



**Abbildung 6.4** Manipulation-Indicators am Beispiel des Interaktions-Modus Skalieren

Durch die Pfeile wird die Achse der Manipulation angegeben. In unserem Beispiel wird die Horizontale für die Skalierung verwendet. Durch die Symbole wird dem Nutzer eindeutig angezeigt, welche Richtung welche Auswirkung hat. Durch die Verschiebung der Pfeile um das Delta der Handposition wird dem Nutzer zudem das Ausmaß seiner Eingabe widergespiegelt.

### 6.4 Das App-Menü

Bisher wurden Interaktionen behandelt, die nur bei einem vorhanden Wald relevant sind. Der Nutzer von CodeLeaves muss jedoch zuvor auswählen können, welche

## 6 Interaktion mit CodeLeaves

Software er überhaupt visualisieren möchte (Akz. 7.1). Auch müssen die dargestellten Informationen nach der Auswahl eines Projekts konfiguriert werden können (Akz. 1.1, 2.3, 4.1 und 4.3). Dafür wird ein zentrales *App-Menü* eingeführt.

Dieses Menü orientiert sich an herkömmlichen Fenstern, wie sie bei Desktop-Systemen zu finden sind. Bevor näher auf den Inhalt des App-Menüs eingegangen wird, muss zunächst auf die Positionierung des Fensters im Raum eingegangen werden.

### Positionierung

Initial dient das App-Menü dazu, ein Projekt auszuwählen bzw. eines von einer unterstützen Datenquelle zu importieren. Deshalb sollte das Menü nach dem Start der App in jedem Fall im Fokus des Nutzers platziert werden, sodass dieser sofort mit CodeLeaves arbeiten kann. Dies kann mit dem in Abschnitt 5.1 vorgestellten Tag-along-Effekt bewerkstelligt werden. Für den initialen Zustand ist der Tag-along-Effekt des App-Menüs als nicht störend zu bewerten, da der Nutzer in jedem Fall mit dem Menü interagieren möchte und noch keine andren Inhalte vorhanden sind, die das Menü verdecken könnte.

Nach der Auswahl eines Projekts kann der Tag-along-Effekt aber durchaus als störend empfunden werden, da er das Menü im Vordergrund hält und die Sicht auf den Wald versperrt. Das Fenster des App-Menüs muss demnach die Möglichkeit unterstützen, stationär im Raum platziert oder ganz geschlossen zu werden.

Zweiteres kann mit einem Schließen-Button in einer Titelleiste realisiert werden, was dem Nutzer aus verbreiteten Systemen vertraut ist. Das erneute Öffnen kann und muss durch das Kontextmenü von Objekten des Waldes möglich sein. In Abbildung 6.3 ist diese Möglichkeit im Kontextmenü des Waldbodens bereits zu sehen gewesen. Auch im Kontextmenü eines Blattes oder eines inneren Knotens sollte das App-Menü aufrufbar sein, da der Nutzer damit Eigenschaften der Blätter und inneren Knoten verändern kann.

Die Funktion des stationären Platzierens kann ebenfalls in der Titelleiste untergebracht werden. Analog zum „Vollbild“ eines Desktop-Fensters, bei dem der Nutzer sich ausschließlich auf dieses konzentrieren möchte, wird neben dem Schließen-Button ein *Tag-along-Button* eingeführt, der für das Ein- und Ausschalten des Tag-along-Effekts zuständig ist. Damit kann der Nutzer das Fenster jederzeit dort platzieren, wo es sich gerade befindet. Wird das App-Menü aus einem Kontextmenü aufgerufen, wird der Tag-along-Effekt aktiviert, sodass der Nutzer das Fenster nicht suchen muss, sondern es zu ihm kommt, sobald er es braucht.

Die Titelleiste kann für eine weitere Funktion genutzt werden. Wird bei Desktop-Systemen ein Drag-and-drop der Titelleiste durchgeführt, lässt sich das zugehörige Fenster verschieben. Dies ist in 3D ebenfalls möglich. Wird die Manipulation-Geste bei fokussierter Titelleiste durchgeführt, wird das Fenster des App-Menüs entsprechend im Raum verschoben. Durch die dreidimensionale Eingabe der Manipulation-Geste kann der Drag-and-drop aus dem Zweidimensionalen einfach in drei Dimensionen übertragen werden.

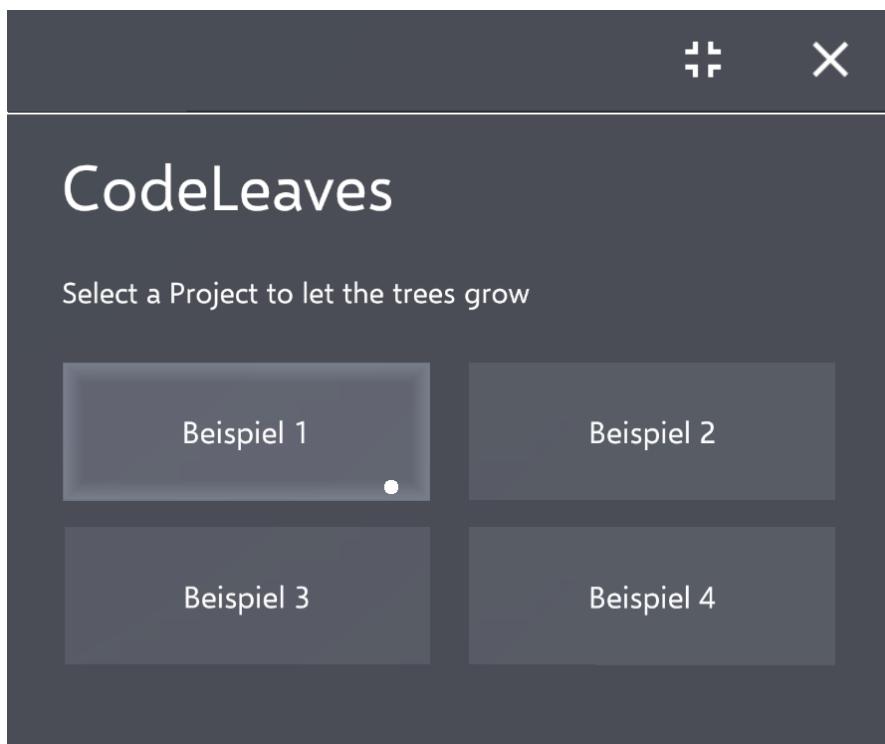
In dem High-Fi-Prototyp wurden die beschriebenen Funktionen des App-Menüs bereits umgesetzt. In Abbildung 6.5 kann das erstellte Menü betrachtet werden. Die

## 6.4 Das App-Menü

Auswahl eines Projekts beschränkt sich dabei auf gespeicherte Beispielprojekte (Akz. 7.1).

### Projektauswahl

Für die produktive Einsetzung von CodeLeaves ist ein Import von neuen Projekten nötig (Akz. 7.2). In dem Low-Fi Papier Prototyp in Abbildung 6.6 ist die Interaktion mit zusätzlicher Import-Möglichkeit dargestellt.



(a) Initialzustand zur Auswahl eines Beispielprojektes und aktivem Tag-along



(b) Titelleiste mit deaktiviertem Tag-along

**Abbildung 6.5** High-Fi-Prototyp der Projektauswahl im App-Menüs

Bei der Projektauswahl hat der Nutzer die Wahl zwischen zwei Haupt-Tasks: Der Auswahl eines bereits betrachteten Projektes und dem Import von neuen Projekten. Die Interaktion bei einem Import ist für SonarQube beispielhaft im Anhang zu finden.

Ist ein Projekt ausgewählt bzw. importiert, wird der Wald nach dem Layout generiert, das in Kapitel 4 erarbeitet wurde. Danach wird dem Nutzer die Möglichkeit geboten, den Wald so anzupassen, dass er genau die Informationen zeigt, die der Nutzer sucht.

## 6 Interaktion mit CodeLeaves

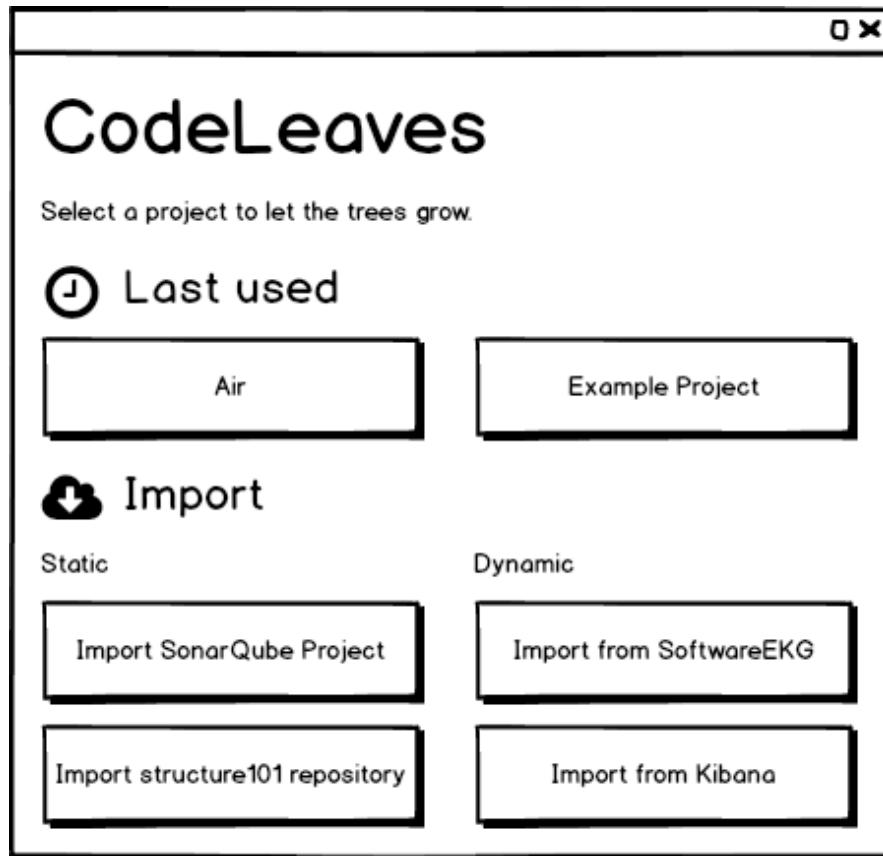


Abbildung 6.6 Low-Fi-Prototyp der Projektauswahl im App-Menü

### Einstellungen

Dies kann er ebenfalls mit dem App-Menü erreichen, das sich nun in eine Schaltzentrale mit allen nötigen Einstellungen verwandelt. Im High-Fi-Prototyp wurden nur essentielle Einstellungen eingebaut, die für die Nutzung des Prototyps unerlässlich sind. In Abbildung 6.7 ist dagegen wiederum eine vollständige Low-Fi-Prototyp-Zeichnung zu sehen.

Der Nutzer muss eine Metrik für die Farbe der Blätter (Akz. 1.1 und 4.1) und ein Verbindungs-Typ für die Dicke der Wurzeln und Kanten (Akz. 2.3, und 4.3) angeben können. Die Zuordnung von Metriken für die Farbe der Blätter und die Stärke der Verbindungen nehmen deshalb als Drop-down-Menüs zentrale Elemente in den Einstellungen ein.

Die zur Auswahl stehenden Metriken sind davon abhängig, ob der Nutzer gerade dynamische oder statische Daten betrachten möchte.

Deshalb muss als erstes die Entscheidung getroffen werden, ob statische oder dynamische Informationen betrachtet werden sollen. Mit dem „Add“ neben dieser Auswahl kann der Nutzer Daten zu dem geladenen Projekt hinzufügen (Akz. 7.3). Je nachdem welche Auswahl getroffen wird, kann der Nutzer darunter entweder einen Zeitpunkt –

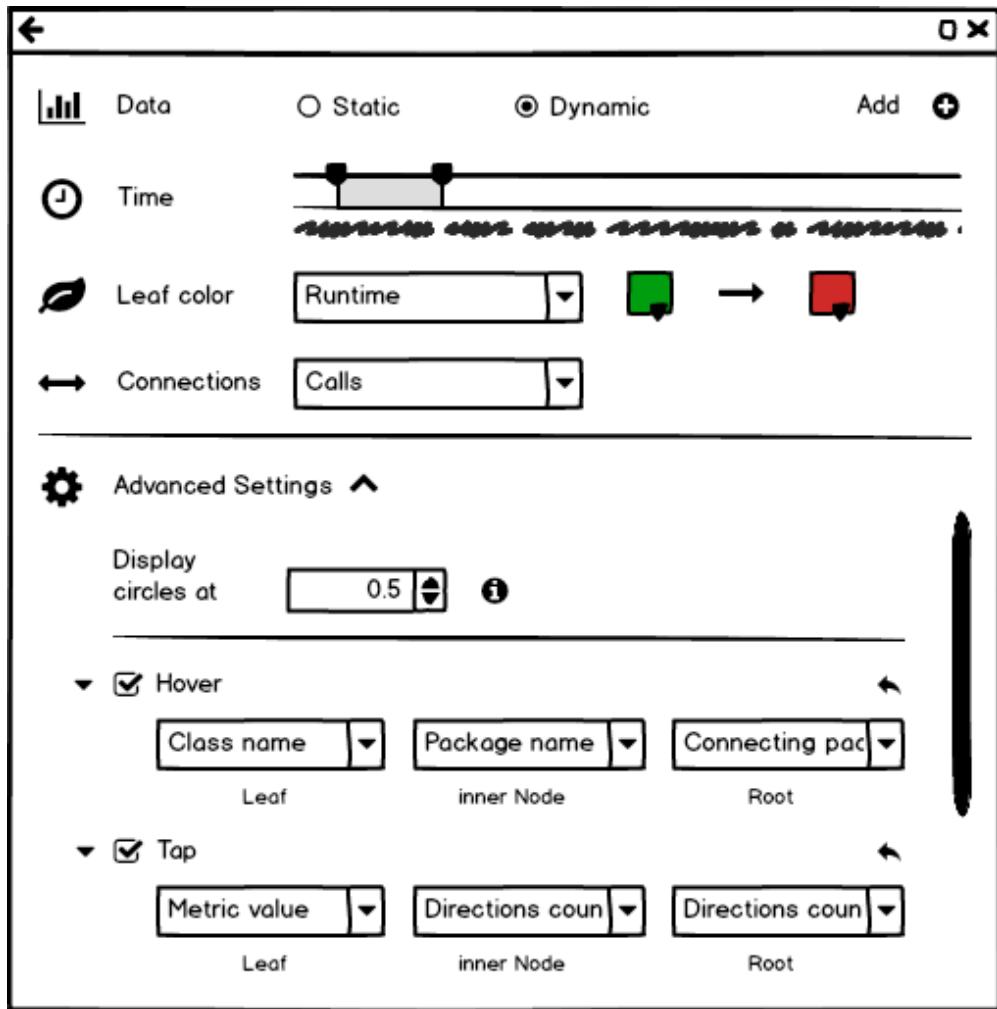


Abbildung 6.7 Low-Fi-Prototyp für die Einstellungen im App-Menü

in diesem Fall wird nur **ein** Schieberegler angezeigt – oder einen Zeitraum angeben (Akz 3.7 und 4.8).

Die Drop-down-Menüs für die dargestellten Metriken passen sich ebenfalls der Auswahl von „Data“ an. Damit sind die wichtigsten Einstellungsmöglichkeiten schon abgedeckt und der Nutzer kann die aktuell dargestellten Daten interaktiv erforschen.

D. Norman stellt im Interaction Design Klassiker *The Design of Everyday Things* sieben allgemeingültige Prinzipien für gutes Interaction Design auf. Nach dem siebten Prinzip *Constraints* sollten die möglichen Aktionen des Nutzers eingeschränkt werden, um die Aktionen zu lenken und den Nutzer nicht zu überfordern [32]. Dies lässt sich auch auf die Einstellungen für CodeLeaves anwenden.

Mit den bisher beschriebenen Einstellungen ist der Nutzer bereits in der Lage, den Wald als Ganzes zu konfigurieren. Weitere optionale Einstellungen werden deswegen in „Advanced Settings“ gekapselt, sodass sich der Nutzer damit nicht auseinandersetzen muss, wenn er das nicht möchte. Der fortgeschrittene Nutzer findet dort jedoch

## 6 Interaktion mit CodeLeaves

nützliche Anpassungen.

Zum einen hat er die Möglichkeit, den verfügbaren Eingabemöglichkeiten einer Aktion zuzuweisen (mehr dazu in Abschnitt 6.5).

Zum anderen ist in den „Advanced Settings“ die Einstellung „Display circles at“ zu finden, die durch das Prototyping mit der HoloLens entstanden ist.

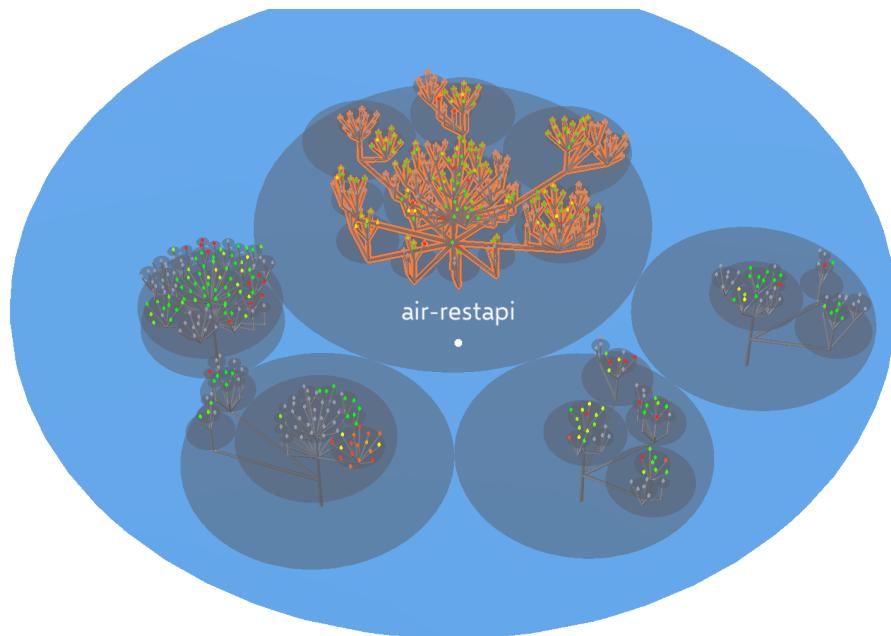
### Umschließende Kreise zur erleichterten Bedienung

Die Kreise, die zur Visualisierung des Circle-Packings im Abbildung 4.6 dargestellt sind, haben sich auch für die Interaktion als überaus hilfreich erwiesen. CodeLeaves hat auf der HoloLens mit folgendem Problem umzugehen:

*Je kleiner der Wald skaliert wird, desto schwieriger wird die Interaktion mit einzelnen Blättern oder Kanten.*

Abhilfe schaffen die umschließenden Kreise, die neben der Visualisierung auch als Ziel der Interaktion dienen können. Ein Knoten besitzt bei aktivierte Kreisen als Alternative zur Kante selbst für die Interaktion auch seinen Kreis, der deutlich mehr Fläche für die Interaktion bietet.

Abbildung 6.8 zeigt die Interaktion mit einem Kreis am Beispiel der Fokussierung. Auf diese Aktion wird im nächsten Abschnitt eingegangen.



**Abbildung 6.8** Visualisierung der Kreise für leichtere Interaktion

Durch die Einstellung für die Kreise ist konfigurierbar, ab welchem Skalierungsfaktor des Waldes die Kreise angezeigt werden. Standardmäßig wäre dies der Fall, sobald die Interaktion mit einzelnen Blättern oder Kanten nicht mehr komfortabel möglich ist.

## 6.5 Interaktion mit Blättern, inneren Knoten und Wurzeln

Der Schwellenwert dafür müsste durch weiterführende Nutzer-Befragungen evaluiert werden.

### 6.5 Interaktion mit Blättern, inneren Knoten und Wurzeln

Mit dem App-Menü haben wir die Möglichkeit, die Daten auszuwählen, die der Wald darstellen soll. Was noch fehlt, ist die direkte Interaktion mit den Objekten des Waldes – mit den Blättern, inneren Knoten und Wurzeln. Mit der Blickrichtung und den verfügbaren diskreten Gesten der HoloLens gibt es insgesamt vier verschiedene Eingabemöglichkeiten, mit denen der Nutzer mit den Objekten in Interaktion treten kann:

- Fokus
- Tap (Air-Tap)
- Double-Tap
- Tap-and-hold

Da es aber für jedes Objekt mehr mögliche Aktionen als Eingabemöglichkeiten gibt, ist das Kontextmenü immer durch mindestens eine Eingabemöglichkeit erreichbar und bietet alle verfügbaren Aktionen an.

Bevor wir eine Übersicht aller Aktionen und deren mögliche Zuordnung betrachten, widmen wir uns zunächst dem Fokus.

#### Fokus

A. Cooper ist in [9] der Meinung, dass “constant [...] feedback is exactly what users need”. So sollte abgesehen von weiteren Auswirkungen dem Nutzer immer Feedback zum aktuellen Fokus gegeben werden. Der Nutzer muss zu jeder Zeit wissen, welches Objekt er gerade fokussiert, was dazu gehört und dass er damit interagieren kann. In CodeLeaves wird dies bei Objekten des Waldes durch einen farbigen Umriss erreicht. Bei inneren Knoten werden auch alle Nachfahren hervorgehoben, da auf fachlicher Ebene alle Unterpakete auch Teil eines Pakets sind. So ist zum Beispiel auch immer zu erkennen, welche Äste zu welchem Baum gehören, auch wenn die Bäume hintereinander stehen.

Wird die Hervorhebung aber sofort aktiv, sobald die Blickrichtung ein Objekt trifft, hat das zur Folge, dass beim „Blick schweifen lassen“ die Objekte nur kurz aufblitzen. Bei der Entwicklung des High-Fi-Prototyps konnte festgestellt werden, dass diese Art der Hervorhebung zu schnell und unangenehm ist.

Um das zu vermeiden, wird das Feedback des Fokus verzögert, sodass es erst sichtbar wird, sobald der Fokus einen bestimmten Zeitraum auf demselben Objekt verweilt. Ein passendes Pendant bei Desktop-Anwendungen wären Tooltips, die erst angezeigt werden, wenn der Mauszeiger über einem Element verweilt.

Für den festzulegenden Zeitraum wurde sich an den Arbeiten [30] und [31] orientiert. Dort ist von der sogenannten *Immediate Behaviour* die Rede, die in einem Zeitraum von

## 6 Interaktion mit CodeLeaves

0,5 bis 1 Sekunde statt finden sollte. Dieser Zeitraum ist die Zeit, die ein Gesprächspartner gewöhnlich benötigt, um in einer Mensch-zu-Mensch Kommunikation zu antworten und die auch in Mensch-zu-Maschine Kommunikation eingehalten werden sollte.

Mit dieser Lösung ist es möglich, sich im Wald in Ruhe umzuschauen. Sobald der Fokus länger auf einem Objekt bleibt, wird das Objekt hervorgehoben. So weiß der Nutzer zu jeder Zeit, mit welchem Objekt er gerade interagiert.

### Übersicht über mögliche Aktionen

In der Tabelle 6.1 können alle verfügbaren Aktionen betrachtet werden. Die Icons in der zweiten Spalte befinden sich zur Wiedererkennung zum Teil auch in den Abbildungen des Low-Fi-Prototyps der Kontextmenüs. In den letzten drei Spalten ist zu finden, mit welcher Eingabe die Aktion für welches Objekt aufrufbar ist. Ist eine Zelle darin leer, ist die Aktion für das entsprechende Objekt nicht möglich. „KM“ steht dafür, dass die Aktion über das Kontextmenü aufrufbar ist.

**Tabelle 6.1** Mögliche Aktionen und deren Erreichbarkeit

ID	Aktion	Blatt	innerer Knoten	Wurzel
1		Name der Klasse, des Pakets bzw. der verbundenen Pakete anzeigen (Akz. 3.1)	Fokus	Fokus
2		Kontextmenü aufrufen	Tap-and-hold	Tap-and-hold
3		Wert der aktuellen Metrik anzeigen (Akz. 3.2 und 3.2)	Tap	
4		Liste mit allen Metriken anzeigen	KM	
5		Anzeigen, wie vielen Verbindungen in welche Richtung gehen (Akz. 3.4 und 4.7)		Tap
6		Zugeordnete Verbindungen hervorheben (Akz. 3.3, 4.5, 3.5 und 4.6)	Double-Tap	Double-Tap
7		Eingehende Verbindungen hervorheben	KM	KM

*Fortsetzung auf nächster Seite...*

## 6.5 Interaktion mit Blättern, inneren Knoten und Wurzeln

Tabelle 6.1 – Fortsetzung von vorheriger Seite

ID	Aktion	Blatt	innerer Knoten	Wurzel
8		Ausgehende Verbindungen hervorheben	KM	KM
9		Verbindungen nach Rechts hervorheben		KM
10		Verbindungen nach Links hervorheben		KM
11		Verbindungen direkt anzeigen	KM	KM
12		Objekt zum späteren Wiederfinden markieren	KM	KM
13		Wald so skalieren, dass die Auswahl das ganze Sichtfeld der HoloLens einnimmt		KM
13		Knoten als neuen Waldboden verwenden (Akz. 3.6)		KM

Die Zuordnung der Aktionen zu den vorhanden Eingabemöglichkeiten kann, wie in Abschnitt 6.4, Abbildung 6.7 zu sehen, bei Bedarf vom Nutzer angepasst werden. Dies ist sinnvoll, wenn der Nutzer mit einer anderen Zuweisung schneller oder komfortabler an gesuchte Informationen gelangt. Möchte der Nutzer zum Beispiel nur Klassen mit niedriger Testabdeckung identifizieren, ist es sinnvoll, wie in Tabelle 6.1 angegeben, bei einem Air-Tap auf ein Blatt den Wert der Testabdeckung anzuzeigen. Ist der Nutzer jedoch ausschließlich an statischen Abhängigkeiten interessiert, kann er anstatt des standardmäßigen Double-Tap auch einen einfachen Air-Tap für die Anzeige der zugehörigen Verbindungen verwenden.

Im Folgenden werden einige Aktionen noch genauer beschrieben.

**i** Die Anzeige der Namen von Klassen und Paketen ist für deren Identifikation besonders wichtig, da der Nutzer nur so weiß, welchen Teil der Software er vor sich sieht. Der Fokus ist dafür komfortabel, da ein fokussiertes Objekt zwangsläufig immer das ist, wofür sich der Nutzer interessiert und der Nutzer nicht dauerhaft Air-Taps durchführen muss, um sich umzuschauen. Der angezeigte Name befindet sich aus Nutzersicht immer über dem Cursor und wird dem Abstand entsprechend so skaliert, dass der Text immer gleich groß erscheint. Wichtig ist, dass der Text in der Entfernung des fokussierten Objekts angezeigt wird, und nicht etwa wie das Kontextmenü innerhalb von zwei Metern, da der Nutzer sonst zwischen unterschiedlichen

## 6 Interaktion mit CodeLeaves

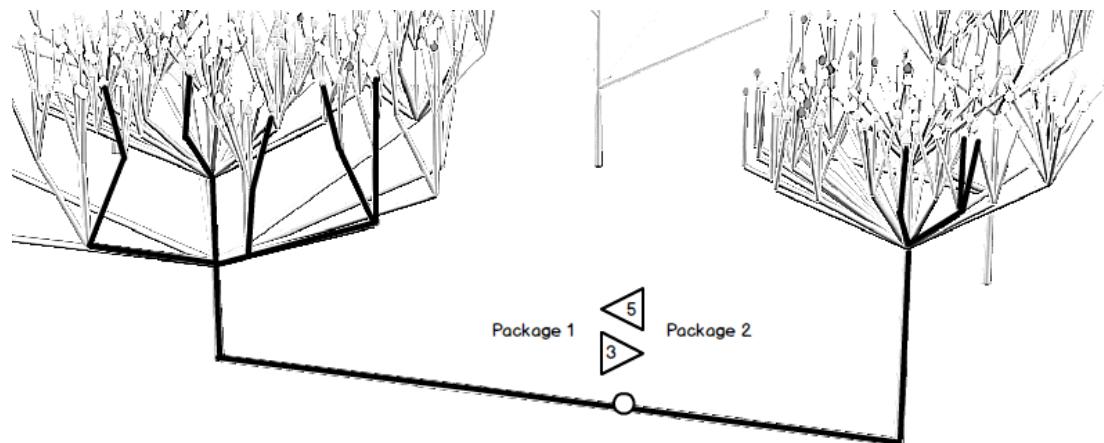
Entfernungen scharfstellen muss.

● Das Kontextmenü ist für jedes Objekt aufrufbar, da nur so alle Aktionen erreichbar sind. Die Tap-and-hold-Geste eignet sich für das Kontextmenü, da diese Eingabe bei touchbasierten Eingabegeräten ebenfalls oft für Kontextmenüs verwendet wird.

■ Beim Blatt kann der genaue Wert einer Metrik durch einen einfachen Air-Tap aufgerufen werden. Durch die Liste aller Metriken, die der Nutzer über das Kontextmenü aufrufen kann, kann der Nutzer eine bestimmte Klasse genauer untersuchen, ohne die aktuelle Metrik umstellen zu müssen.

↔ Mit der Anzahl der Verbindungen ist ersichtlich, wie die Dicke der Verbindung zustande kommt.

↔ Durch das Hervorheben der Verbindungen kann der Nutzer sehen, zu welchen Blättern die Verbindungen gehören. Die Anzahl der Verbindungen können dabei ebenfalls angezeigt werden. Dieser Fall ist am Beispiel einer Wurzel in Abbildung 6.9 illustriert. Führt der Nutzer ein Air-Tap auf eine Kante mit bereits hervorgehobener Verbindung aus, werden nur die Verbindungen hervorgehoben, die in der vorherigen Auswahl enthalten waren und die durch die ausgewählte Kante fließen. Mit diesem Prinzip kann jede Verbindung immer feiner untersucht werden.



**Abbildung 6.9** Hervorhebung und Anzeige der Anzahl und Richtung von einzelnen Verbindungen einer Wurzel

● Mit den Aktionen 7 - 10 ist es möglich, eine Vorauswahl der hervorgehobenen Verbindungen zu treffen. Zum Beispiel würden im Szenario aus Abbildung 6.9 mit Aktion 9 nur die 3 Verbindungen gezeigt, die von Paket 1 nach Paket 2 fließen. Der Aufruf dieser Aktionen ist nicht nur im Kontextmenü, sondern auch durch den

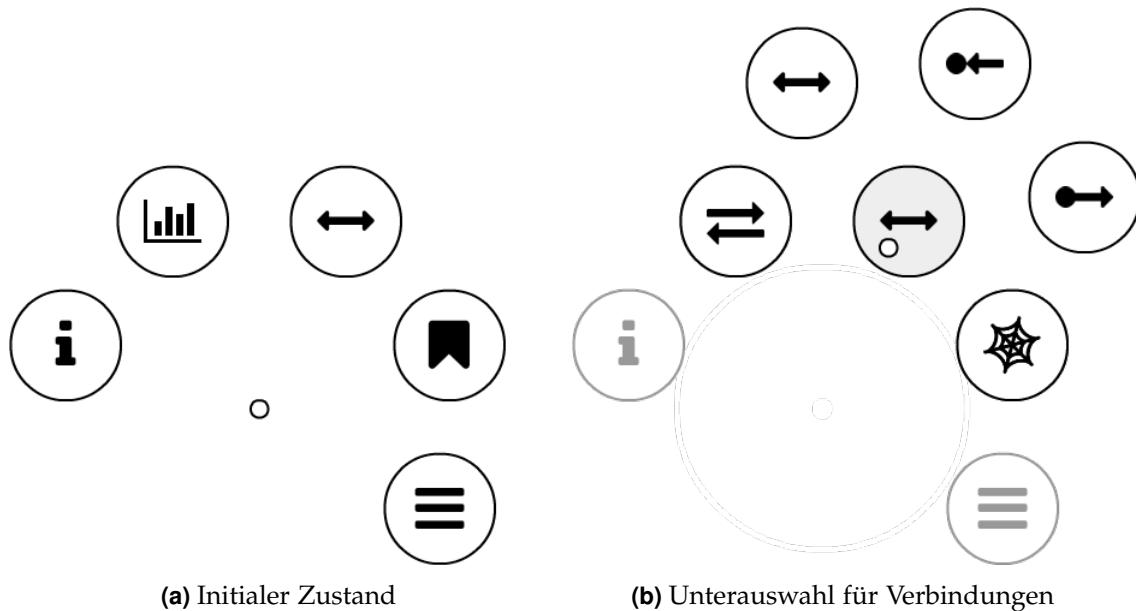
## 6.5 Interaktion mit Blättern, inneren Knoten und Wurzeln

Air-Tap auf die Pfeile für die Richtungsangaben möglich.

- 🕸️ Jedes Objekt hat im Kontextmenü die Option, die gerade ausgewählten Verbindungen auch direkt als Spinnweben anzuzeigen.
- 🔖 Aktion 12 ist komfortabel, wenn Interesse an bestimmten Klassen besteht und vielleicht unterschiedliche Daten und Metriken hintereinander ausgewählt werden und der Nutzer ein Objekt schnell wiederfinden möchte.
- ↔↔ Die Möglichkeit, ein Paket so zu vergrößern oder zu verkleinern, dass es genau in das Sichtfeld der HoloLens passt, ist bei dem kleinen Sichtfeld der HoloLens eine Funktion, die die User-Experience durchaus positiv beeinflussen und den Workflow beschleunigen kann.

Abschließend zu Interaktion mit Blättern, Kanten und Wurzeln ist in 6.10 das Kontextmenü für Blätter dargestellt. Auf der rechten Seite (6.10b) sehen wir eine Unterauswahl, die aufgeht, sobald der Nutzer auf das Symbol der Verbindungen tippt. Auf diese Weise sind auch gruppierte Kontextmenüs möglich.

Die Kontextmenüs für Kanten und Wurzeln sind im Anhang zu finden.



**Abbildung 6.10** Kontextmenü für ein Blatt



# 7 Zusammenfassung und Ausblick

## 7.1 Zusammenfassung

Zu Beginn haben wir uns dem Thema AR genähert und evaluiert, wieso AR gut für 3D-Softwarevisualisierung geeignet ist. Auf dieser Basis wurde im 2. Kapitel CodeLeaves vorgestellt, um das neue Konzept zu erfassen und dessen Stärken kennen zu lernen. Mit der Wald-Metapher können Software-Strukturen mit Bäumen hervorragend visualisiert werden. Die ausgewählte Software wird zum Wald, Pakete zu Bäumen und Ästen und Klassen zu Blättern. Ein Wurzelgeflecht verbindet einzelne Bäume.

Auf das Blätterdach lassen sich verschiedene Metriken anwenden. Mit einem sommerlichen grün bis hin zu einem herbstlichen rot verschafft der Wald mit einem Blick eine Übersicht über die gesamte Software. Besonders Bäume oder Äste mit rötlicher Lauffärbung stechen in einem sonst grünen Wald besonders gut hervor.

Nicht nur Metriken lassen sich mit CodeLeaves darstellen, auch Verbindungen zwischen einzelnen Klassen. Seien es statische Abhängigkeiten oder dynamische Aufrufe – CodeLeaves kann Verbindungen durch die vorhandenen Baumstrukturen aggregieren und so übersichtlich anzeigen. Was bei anderen Visualisierungen ein undurchdringliches Dickicht an Pfeilen ist, wird zur Dicke der Kanten und Wurzeln. Je mehr aggregierte Verbindungen durch eine Kante oder Wurzel fließen, desto dicker wird sie. Durch diese Darstellung kann schon ohne jegliche Interaktion eine gute Übersicht über die Kopplung einer Software geschaffen werden.

Verbindungen können interaktiv ebenfalls direkt angezeigt werden und fügen sich als Spinnweben zwischen den Bäumen nahtlos in die Metapher des Waldes ein.

In Kapitel 3 wird ein Datenmodell entwickelt, das einerseits Schichtentrennung zwischen Backend, Anwendungslogik und UI ermöglicht und andererseits die Visualisierung von statischen und dynamischen Daten ermöglichen. Diese Möglichkeit ist ein Alleinstellungsmerkmal von CodeLeaves.

Im Zuge dieser Arbeit wurde ein High-Fi-Prototyp für die HoloLens entwickelt. Primäres Ziel dabei war es, Algorithmen für die Generierung der 3D-Baumstrukturen zu suchen und grundlegende Interaktionen zu implementieren.

Die verwendeten Algorithmen und die dafür nötige Mathematik wurde in Kapitel 4 beschrieben. Es wird gezeigt, wie sich in der Natur vorkommende Strategien mit dem Goldenen Schnitt auf die Bäume von CodeLeaves übertragen lassen. Mit dem Sonnenblumen-Algorithmus werden Blätter eines Elternteils gleichmäßig auf einer Kreisfläche verteilt.

Durch hierarchisches Circle-Packing werden die Knoten der Bäume so verteilt, dass Äste sich nicht überschneiden können und alle Blätter des Waldes von oben betrachtet sichtbar sind.

## 7 Zusammenfassung und Ausblick

Im Zuge des High-Fi-Prototypings wurde auch festgestellt, dass sich die HoloLens in der aktuellen Entwickler-Version nicht für die Betrachtung von sehr großen Software-Systemen eignet, da die Visualisierung dann nicht mehr flüssig gerendert wird.

Neben dem Layout des Waldes wurde die Interaktion mit CodeLeaves intensiv behandelt. Dafür wurden in Kapitel 5 Grundlagen erarbeitet. Die zur Verfügung stehenden Eingabemöglichkeiten der HoloLens wurden vorgestellt und allgemeine Herausforderungen der Interaktion in der AR beleuchtet. Mit reaktiver Programmierung wurde zudem praxisnah erläutert, wie die lose Kopplung von Reaktion auf Interaktionen, Zustandsänderung der Anwendung und Aktualisierung der UI umgesetzt werden kann.

Zu guter Letzt wurden die im High-Fi-Prototyp bereits umgesetzten Interaktionen in Kapitel 6 aufgegriffen und durch einen Low-Fi-Prototyp ergänzt. Mit dem universellen Kontextmenü und den Manipulation-Indicators wurden Interaktions-Elemente eingeführt, die auch in anderen Anwendungen Verwendung finden können. Das erarbeitete Interaktions-Konzept ermöglicht es, den Wald interaktiv bis ins Detail zu erkunden.

### 7.2 Ausblick

Mit dem High-Fi-Prototyp wurden bereits die wichtigsten Schritte für eine einsatzfähige Softwarevisualisierung gegangen. Für den produktiven Einsatz sind jedoch noch einige Aufgaben zu bewältigen.

**Datenanbindung** Der Import von Daten benötigt ein Interface, wie es in Abbildung 6.6 und A.1 skizziert ist. Neben SonarQube für die Struktur der Software und deren statischen Metriken gilt es weitere Import-Möglichkeiten zu unterstützen. Für statische Abhängigkeiten bietet sich *Structure101*<sup>1</sup> an. Mit Structure101 sind Abhängigkeiten auf Klassenebene verfügbar. Der Export der Daten von Structure101 wurde bereits erfolgreich getestet und muss als nächster Schritt in das Modell von CodeLeaves gebracht werden.

Für dynamische Daten lässt sich zum Beispiel das Software-EKG der QAware verwenden. Die EKG-Kollektoren sammeln dynamische Daten aus unterschiedlichen Datenquellen wie der Programmschnittstelle JMX und Windows Performance Counter [45]. Das Software-EKG besitzt eine eigene UI zur Darstellung der gesammelten Daten. Statt dieser Visualisierung müssten die Daten in CodeLeaves importiert und auf die Struktur der Software übertragen werden.

**Evaluation der Aggregation** Liegen ausreichend Testdaten für Verbindungen in Form des Software-Meta-Modells vor, muss die Anzahl der Einzelverbindungen auf die Dicke der Kanten und Wurzeln übertragen werden. Der Aufwand für die Erweiterung des Prototyps für die Unterstützung dieser Funktionalität ist gering, da der Prototyp bereits unterschiedliche Dicken von Kanten und Wurzeln generieren kann. Jedoch muss diese Funktionalität mit realistischen Daten evaluiert werden.

---

<sup>1</sup>Structure101 ist eine agile Architektur-Entwicklungs-Umgebung (engl. ADE), mit der das Software-Entwicklungsteam eine Code-Basis organisieren kann (aus dem Englischen aus [27]).

## 7.2 Ausblick

**Evaluierung des Interaktionskonzepts** Das Interaktionskonzept, das im Rahmen dieser Arbeit entwickelt wurde, muss mit Test-Nutzern evaluiert werden. In diesem Zuge kann das Konzept überprüft und bei Bedarf überarbeitet werden. Anschließend müssen die fehlenden Elemente wie die Kontextmenüs für Blätter, innere Knoten und Wurzeln ergänzt werden.

Mit dem Import von Daten, der Umsetzung der aggregierten Verbindungen und der vollständigen Umsetzung des evaluierten Interaktionskonzepts ist CodeLeaves einsatzfähig. Mit der HoloLens sind darüber hinaus noch Erweiterungen möglich.

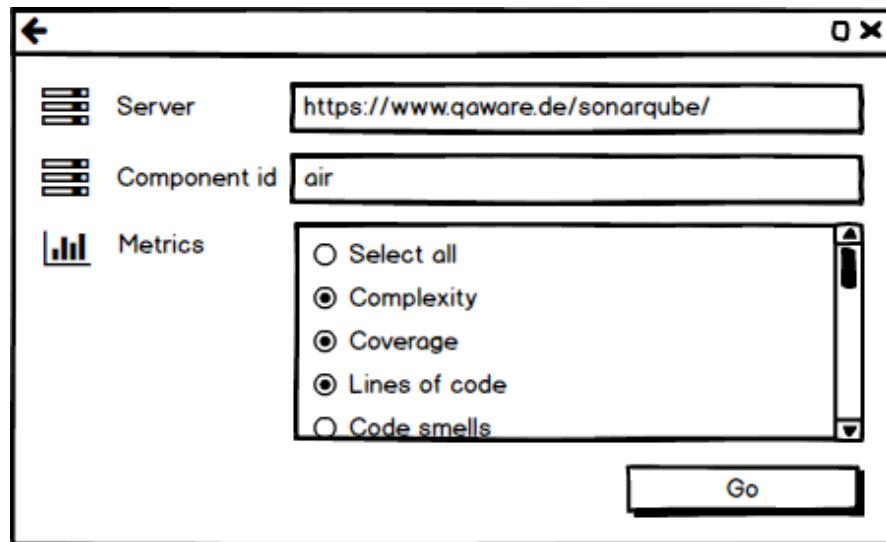
**Sprachsteuerung** Durch die Sprachsteuerung, die in die HoloLens integriert ist, könnten einige Interaktionen eventuell noch komfortabler gestaltet werden. Die Arbeit mit dem Air-Tap kann für den Arm des Nutzers auf Dauer ermüdend sein. Die Sprachsteuerung kann daher für einige Interaktionen sinnvoll als Alternative zum Air-Tap genutzt werden. Auch zusätzliche Aktionen wie „Zeige mir die Klasse ‚ExampleClass‘“ sind mit Sprachsteuerung möglich und besonders komfortabel.

**Teilen der Anwendung für mehrere Nutzer** Die Möglichkeit, holographische Inhalte gleichzeitig mit anderen Nutzern teilen zu können, ist ein großer Vorteil der AR. Diese muss jedoch auch in HoloLens-Applikationen implementiert werden. Microsoft stellt dafür die *HoloToolkit.Sharing*-Bibliothek zur Verfügung. Diese Bibliothek wurde ursprünglich für das Projekt *OnSight* entwickelt, eine Zusammenarbeit zwischen einem Microsoft Studio und der NASA, um das Rover-Planungstool der NASA durch die Nutzung von HoloLens-Geräten zu erweitern. Apps, die das Teilen unterstützen, laufen auf mehreren HoloLens-Geräten gleichzeitig und kommunizieren und synchronisieren sich in Echtzeit, sodass Nutzer gemeinsam an einer Aufgabe arbeiten können [11].

Die Umsetzung der Möglichkeit, den Wald gemeinsam zu erkunden, wäre eine überaus spannende weiterführende Aufgabe, die CodeLeaves noch attraktiver machen würde.



## A Anhang



**Abbildung A.1** Low-Fi Prototyp für den Import eines neuen Projekts anhand des Beispiels SonarQube

A Anhang

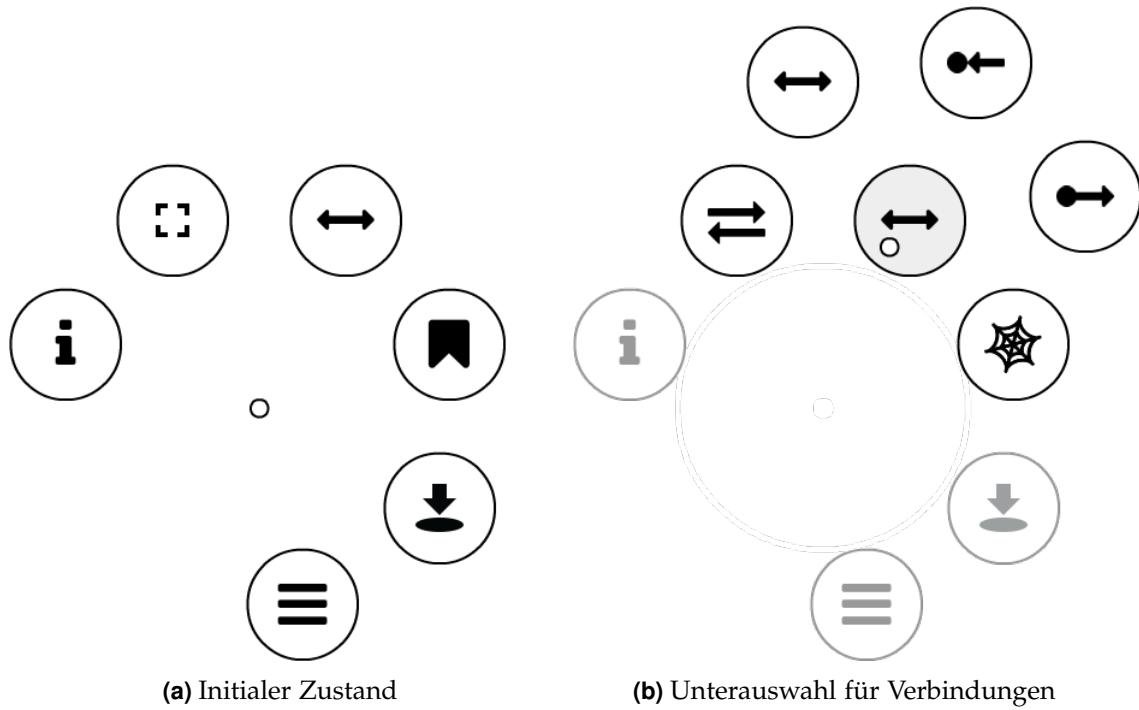


Abbildung A.2 Kontextmenü für einen inneren Knoten

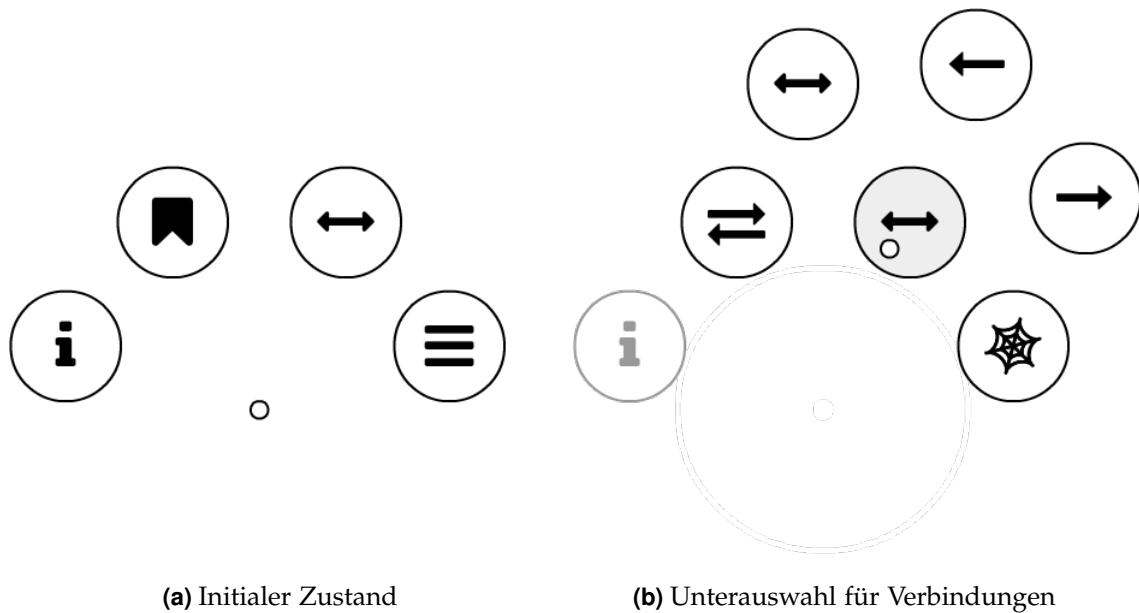


Abbildung A.3 Kontextmenü für eine Wurzel

# Literatur

- [1] Ronald Azuma u. a. „Recent advances in augmented reality“. In: *IEEE computer graphics and applications* 21.6 (2001), S. 34–47.
- [2] Ronald T Azuma. „A survey of augmented reality“. In: *Presence: Teleoperators and virtual environments* 6.4 (1997), S. 355–385.
- [3] Michael Balzer u. a. „Software landscapes: Visualizing the structure of large software systems“. In: *IEEE TCVG*. 2004.
- [4] Mike Bostock. „A Better Way to Code“. In: *Medium* (2017). URL: <https://medium.com/@mbostock/a-better-way-to-code-2b1d2876a3a0>.
- [5] Windows Dev Center. *Gesture design*. 2017. URL: [https://developer.microsoft.com/en-us/windows/mixed-reality/gesture\\_design](https://developer.microsoft.com/en-us/windows/mixed-reality/gesture_design) (besucht am 23.08.2017).
- [6] Windows Dev Center. *Interaction design*. 2017. URL: [https://developer.microsoft.com/en-us/windows/mixed-reality/category/interaction\\_design](https://developer.microsoft.com/en-us/windows/mixed-reality/category/interaction_design) (besucht am 23.08.2017).
- [7] Windows Dev Center. *Spatial mapping design*. 2017. URL: [https://developer.microsoft.com/en-us/windows/mixed-reality/spatial\\_mapping\\_design](https://developer.microsoft.com/en-us/windows/mixed-reality/spatial_mapping_design) (besucht am 11.10.2017).
- [8] Mike Cohn. *User stories applied: For agile software development*. Addison-Wesley Professional, 2004.
- [9] Alan Cooper u. a. *About face: the essentials of interaction design*. John Wiley & Sons, 2014.
- [10] Microsoft Corporation. *Mixed Reality Design Labs*. Lizenziert unter MIT. 2017. URL: [https://github.com/Microsoft/MRDesignLabs\\_Unity](https://github.com/Microsoft/MRDesignLabs_Unity) (besucht am 30.08.2017).
- [11] Microsoft Corporation. *MixedRealityToolkit-Unity - Sharing*. 2017. URL: <https://github.com/Microsoft/MixedRealityToolkit-Unity/blob/master/Assets/HoloToolkit/Sharing/README.md> (besucht am 24.10.2017).
- [12] Hannes A. Czerulla und Jan-Keno Janssen. *Microsoft HoloLens im Test: Tolle Software, schwaches Display*. c't Magazin für Computertechnik. 2017. URL: <https://www.heise.de/ct/artikel/Microsoft-HoloLens-im-Test-Tolle-Software-schwaches-Display-3248670.html> (besucht am 24.08.2017).
- [13] Nikolaos Dergiades. „The Soddy circles“. In: *Forum Geom.* Bd. 7. 2007, S. 191–197.
- [14] S. Diehl. *Software Visualization: Visualizing the Structure, Behaviour, and Evolution of Software*. Springer Berlin Heidelberg, 2007. ISBN: 9783540465041.
- [15] Florian N Egger. „Lo-Fi vs. Hi-Fi Prototyping: how real does the real thing have to be?“ In: *OZCHI*. 2000.

## Literatur

- [16] H. Ernst, J. Schmidt und G. Beneken. *Grundkurs Informatik: Grundlagen und Konzepte für die erfolgreiche IT-Praxis - Eine umfassende, praxisorientierte Einführung*. Springer Fachmedien Wiesbaden, 2016. ISBN: 9783658146344.
- [17] Blender Stack Exchange. *How to rotate multiple objects towards one point?* 2017. URL: <https://blender.stackexchange.com/questions/17713/how-to-rotate-multiple-objects-towards-one-point> (besucht am 04.09.2017).
- [18] Florian Fittkau, Alexander Krause und Wilhelm Hasselbring. „Software landscape and application visualization for system comprehension with ExplorViz“. In: *Information and software technology* 87 (2017), S. 259–277.
- [19] Buschmann Frank, Henney Kevlin und C Schmidt Douglas. *Pattern Oriented Software Architecture Volume 5: On Patterns and Pattern Languages*. 2007.
- [20] Marcus Ghaly. *Case study - 3 HoloStudio UI and interaction design learnings*. 2017. URL: [https://developer.microsoft.com/en-us/windows/mixed-reality/case\\_study\\_-\\_3\\_holostudio\\_ui\\_and\\_interaction\\_design\\_learnings](https://developer.microsoft.com/en-us/windows/mixed-reality/case_study_-_3_holostudio_ui_and_interaction_design_learnings) (besucht am 24.08.2017).
- [21] Kim Goodwin. *Designing for the digital age: How to create human-centered products and services*. John Wiley & Sons, 2011.
- [22] Hamish Graham, Hong Yul Yang und 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*. Australian Computer Society, Inc. 2004, S. 53–59.
- [23] H.P. Gumm und M. Sommer. *Einführung in die Informatik*. De Gruyter, 2009. ISBN: 9783486595390.
- [24] Hirokazu Kato und Mark Billinghurst. „Marker tracking and hmd calibration for a video-based augmented reality conferencing system“. In: *Augmented Reality, 1999.(IWAR'99) Proceedings. 2nd IEEE and ACM International Workshop on*. IEEE. 1999, S. 85–94.
- [25] Kevin Kelly. „The untold story of magic leap, the world's most secretive startup“. In: Recuperado de <https://www.wired.com/2016/04/magic-leap-vr> (2016).
- [26] Donald E Knuth. *Fundamental algorithms: the art of computer programming*. 1973.
- [27] Headway Software Technologies Ltd. *structure101*. 2017. URL: <http://structure101.com/> (besucht am 24.10.2017).
- [28] Samuel Merrill. *The moose book*. EP Dutton, 1916.
- [29] Paul Milgram u. a. „Augmented reality: A class of displays on the reality-virtuality continuum“. In: *Photonics for industrial applications*. International Society for Optics und Photonics. 1995, S. 282–292.
- [30] George A Miller. „The magical number seven, plus or minus two: some limits on our capacity for processing information.“ In: *Psychological review* 63.2 (1956), S. 81.

- [31] Denys Mishunov. „Why Perceived Performance Matters, Part 1: The Perception Of Time“. In: *Smashing Magazine* (2015). URL: <https://www.smashingmagazine.com/2015/09/why-performance-matters-the-perception-of-time/>.
- [32] Don Norman. *The design of everyday things: Revised and expanded edition*. Basic Books (AZ), 2013.
- [33] Lothar Papula. „Mathematik für Ingenieure und Naturwissenschaftler Band 3–Vektoranalysis, Wahrscheinlichkeitsrechnung, Mathematische Statistik, Fehler- und Ausgleichsrechnung, 6“. In: *Auf. Braunschweig: Vieweg* (2001).
- [34] David Phelan. *Apple CEO Tim Cook: As Brexit hangs over UK, 'times are not really awful, there's some great things happening'*. 2017. URL: <http://www.independent.co.uk/life-style/gadgets-and-tech/features/apple-tim-cook-boss-brexit-uk-theresa-may-number-10-interview-ustwo-a7574086.html#gallery> (besucht am 30.05.2017).
- [35] Marcel Pütz. „Softwarevisualisierung für Virtual Reality“. Masterseminar. Hochschule Rosenheim, 2017.
- [36] QAware GmbH. *IT-Probleme lösen. Digitale Zukunft gestalten*. 2017. URL: <http://www.qaware.de/leistung/#leistung-realisation> (besucht am 28.03.2017).
- [37] QAware GmbH. *Johannes Weigend - Chefarchitekt, Geschäftsführer und Mitgründer*. 2017. URL: <http://www.qaware.de/unternehmen/johannes-weigend/> (besucht am 01.07.2017).
- [38] George G Robertson, Jock D Mackinlay und Stuart K Card. „Cone trees: animated 3D visualizations of hierarchical information“. In: *Proceedings of the SIGCHI conference on Human factors in computing systems*. ACM. 1991, S. 189–194.
- [39] Claudio Rocchini. *Fractal canopy*. Lizenziert unter CC BY-SA 3.0. 2017. URL: [https://commons.wikimedia.org/wiki/File:Fractal\\_canopy.svg](https://commons.wikimedia.org/wiki/File:Fractal_canopy.svg) (besucht am 04.09.2017).
- [40] André Staltz. *The Introduction to Reactive Programming You've Been Missing*. Lizenziert unter CC BY-NC 4.0. 2016. URL: <https://gist.github.com/staltz/868e7e9bc2a7b8c1f754>.
- [41] Frank Steinbrück. „Consistent Software Cities: supporting comprehension of evolving software systems“. Diss. Cottbus, Brandenburgische Technische Universität Cottbus, 2013.
- [42] Unity. *Unternehmensfakten*. 2017. URL: <https://unity3d.com/de/public-relations> (besucht am 28.06.2017).
- [43] Helmut Vogel. „A better way to construct the sunflower head“. In: *Mathematical biosciences* 44.3-4 (1979), S. 179–189.
- [44] Weixin Wang u. a. „Visualization of large hierarchical data by circle packing“. In: *Proceedings of the SIGCHI conference on Human Factors in computing systems*. ACM. 2006, S. 517–520.
- [45] Johannes Weigend, Johannes Siedersleben und Josef Adersberger. „Dynamische Analyse mit dem Software-EKG“. In: *Informatik-Spektrum* 34.5 (2011), S. 484–495.

## Literatur

- [46] Eric W. Weisstein. „Golden Ratio“. In: *MathWorld – A Wolfram Web Resource* (2017). URL: <http://mathworld.wolfram.com/GoldenRatio.html> (besucht am 15.11.2017).
- [47] Eric W. Weisstein. „Spherical Coordinates.“ In: *MathWorld – A Wolfram Web Resource* (2017). URL: <http://mathworld.wolfram.com/SphericalCoordinates.html> (besucht am 15.11.2017).
- [48] R. Wettel und M. Lanza. „Program Comprehension through Software Habitability“. In: *15th IEEE International Conference on Program Comprehension (ICPC '07)*. 2007, S. 231–240. doi: [10.1109/ICPC.2007.30](https://doi.org/10.1109/ICPC.2007.30).
- [49] R. Wettel und M. Lanza. „Visual Exploration of Large-Scale System Evolution“. In: *2008 15th Working Conference on Reverse Engineering*. 2008, S. 219–228. doi: [10.1109/WCRE.2008.55](https://doi.org/10.1109/WCRE.2008.55).
- [50] Richard Wettel, Michele Lanza und Romain Robbes. „Software systems as cities: A controlled experiment“. In: *Proceedings of the 33rd International Conference on Software Engineering*. ACM. 2011, S. 551–560.
- [51] Wikipedia. *Schnittpunkt – Schnittpunkte zweier Kreise*. 2017. URL: [https://de.wikipedia.org/wiki/Schnittpunkt#Schnittpunkte\\_zweier\\_Kreise](https://de.wikipedia.org/wiki/Schnittpunkt#Schnittpunkte_zweier_Kreise) (besucht am 15.11.2017).
- [52] Matthias Zimmermann. *Gewöhnliche Sonnenblume (Helianthus annuus)*. 2017. URL: <http://www.natur-lexikon.com/Texte/MZ/003/00201-Sonnenblume/MZ00201-Sonnenblume.html> (besucht am 07.09.2017).