

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 3. November 2017

Marcel Pütz

Kurzfassung

here comes the abstract

Schlagworte: Softwarevisualisierung, Wald, Baum, 3D, Augmented Reality, Virtual Reality, Statische Codeanalyse, Abhangigkeiten

Inhaltsverzeichnis

1 Einleitung	1
1.1 Einführung in die AR	1
1.2 Motivation dieser Arbeit	3
1.3 Zielsetzung	3
1.4 Aufbau der Arbeit	4
1.5 Abgrenzung	4
2 Das Konzept CodeLeaves	7
2.1 CodeLeaves und die Metapher Software-Wald	7
2.2 Begriffsklärung	10
2.3 Vergleich mit anderen Konzepten	11
2.4 Anforderungen an CodeLeaves	16
3 Datenmodell	23
3.1 Trennung in Schichten	23
3.2 Software Meta-Modell	24
3.3 Internes Datenmodell	27
3.4 UI-Datenmodell	28
4 Modellierung und Layout des Waldes	31
4.1 Grundlegender Ansatz	31
4.2 Verteilung von Blättern mit dem Sonnenblumen-Algorithmus	34
4.3 Verteilung innerer Knoten mit Circle-Packing	37
4.4 Verwendete Algorithmen in der Praxis	43
5 Grundlagen der Interaktion in der AR	45
5.1 Interaktionsmöglichkeiten der HoloLens	45
5.2 Herausforderungen	47
5.3 Technische Umsetzung von Interaktionen mit Reaktivem UI	49
6 Interaktion mit CodeLeaves	53
6.1 Basisinteraktionen	53
6.2 Das Kontextmenü	55
6.3 Manipulation-indicators	57
6.4 Das App-Menü	57
6.5 Interaktion mit Blättern, inneren Knoten und Wurzeln	63
7 Zusammenfassung und Ausblick	69
7.1 Zusammenfassung	69

7.2 Ausblick	70
A Anhang	73
Literatur	75

Abbildungsverzeichnis

1.1	Abgewandelte Darstellung des Reality-Virtuality-Kontinuums aus [29]	2
2.1	Vorteil der Dreidimensionalität bei der Darstellung von Abhängigkeiten	9
2.2	Bezeichnungen	12
2.3	Von Mitarbeitern der QAware gewünschte Informationen	13
2.4	CodeLeaves und alternative Modelle	14
3.1	Schichten der Datenstrukturen	24
3.2	Meta-Modell einer Software	26
3.3	Internes Datenmodell	27
3.4	UI-Datenmodell	29
4.1	3D Objekte für den 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 Knotens	36
4.5	Sonnenblumen-Algorithmus zur Verteilung von Blättern	38
4.6	Wang's Circle Packing Algorithmus (nach [4])	40
4.7	Berechnung der Position eines Kreises, tangent zu zwei anderen Kreisen .	41
4.8	Layout des Waldes	44
5.1	Air-tap [5]	46
5.2	Verdeckung eines Buttons	48
5.3	Shine-through-Effect von UI Elementen in HoloStudio [20]	49
5.4	Anzeige des Klassennamens durch Klick auf ein Blatt	50
5.5	Datenstrom des Labels	51
6.1	Bounding Box zum Platzieren, Skalieren und Rotieren von Objekten [10] .	54
6.2	Spacial Mapping beim Platzieren des Waldes	56
6.3	Kontextmenü des Waldbodens im HoloLens Prototyp	56
6.4	Manipulation-indicators am Beispiel des Interaktions-Modus Skalieren .	57
6.5	High-Fi Prototyp des App-Menüs	59
6.6	Low-Fi Prototyp der Projektauswahl im App-Menü	60
6.7	Low-Fi Prototyp für die Einstellungen im App-Menü	61
6.8	Visualisierung der Kreise für leichtere Interaktion	62
6.9	Darstellung der Anzahl, Richtung und Hervorhebung der aggregierten Verbindungen einer Wurzel	66
6.10	Kontextmenü für ein Blatt	67

A.1	Interaktion bei dem Import eines neuen Projekts anhand des Beispiels SonarQube	73
A.2	Kontextmenü für einen inneren Knoten	74
A.3	Kontextmenü für eine Wurzel	74

Tabellenverzeichnis

2.1 Akzeptanzkriterien zu User-Story 1	17
2.2 Akzeptanzkriterien zu User-Story 2	18
2.3 Akzeptanzkriterien zu User-Story 3	18
2.4 Akzeptanzkriterien zu User-Story 4	19
2.5 Akzeptanzkriterien zu User-Story 5	19
2.6 Akzeptanzkriterien zu User-Story 6	20
2.7 Akzeptanzkriterien zu User-Story 7	21
4.1 Eckdaten des Beispielprojekts Air	32
6.1 Mögliche Aktionen und deren Erreichbarkeit	64

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 Billionenhöhe unterstützt [25]. Laut einem Cover-Artikel der Zeitschrift *Wire* ist die noch unter Verschluss gehaltene Technologie den Konkurrenzprodukten allen 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 wird und möglicherweise die Digitalisierung revolutionieren könnte. Diese Arbeit wird sich mit einer Anwendung für die AR beschäftigen – der Softwarevisualisierung. Die Spezialisierung auf AR kann nach der Abgrenzung von AR und 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 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].

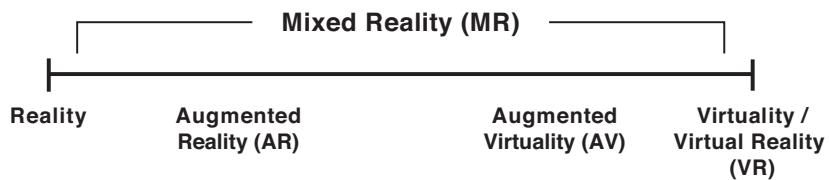


Abbildung 1.1 Abgewandelte Darstellung des Reality-Virtuality-Kontinuums aus [29]

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, da, wie er in einem Interview sagt, diese Technologie nicht wie die VR die wirkliche Welt ausschließt, sondern die Realität erweitert und Teil von zwischenmenschlicher Kommunikation sein kann [34].

Wir stellen uns ein Hologramm vor, dass auf einem Konferenztisch Gestalt annimmt und ein Software-System repräsentiert. Entwickler, Projektleiter oder auch Kunden versammeln sich um den Tisch und können miteinander interaktiv die Software betrachten, evaluieren und wichtige 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 Stand heute am weitesten ausgereiften Technologie der AR gearbeitet – der *HoloLens* von Microsoft.

1.2 Motivation dieser Arbeit

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 Wissenschaften wie Physik, Chemie oder Biologie, nimmt Visualisierung auch besonders in der Informatik eine wichtige Rolle ein. In vielen Bereichen müssen Informationen in eine visuelle Form gebracht werden, die für das menschliche Auge besser zu lesen sind.

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 nur in den Source-Code zu schauen.

So wie UML-Diagramme, war die Darstellungsform der Softwarevisualisierung bislang meist zweidimensional. Mit AR wird dieser Disziplin der Visualisierung jedoch wortwörtlich ein neuer Raum an Möglichkeiten eröffnet und in diese Arbeit soll diesen Raum ausfüllen.

1.3 Zielsetzung

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]. Wird der Fokus mehr auf die Ziele, d.h. den Nutzen für den Betrachter gelegt, lässt sich Softwarevisualisierung wie folgt definieren:

Definition 1.2: Softwarevisualisierung

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

In dieser Arbeit soll das neue Konzept *CodeLeaves* für eine solche Softwarevisualisie-

1 Einleitung

rung in der AR vorgestellt und im Detail ausgearbeitet werden.

Dabei soll CodeLeaves, im Vergleich zu andern 3D Softwarevisualisierungen, die Vorteile der Dreidimensionalität optimal ausnutzen.

Im Vorfeld dieser Arbeit wurden in einer Studie Metriken gesammelt, die eine gute Softwarevisualisierung bzw. CodeLeaves unterstützen sollte.

Es sollen dynamische und statische Metriken zur Erkennung von Anomalien in einer Software unterstützt werden. Ebenfalls soll die Darstellung der Struktur und der darauf abgebildeten Abhängigkeiten innerhalb einer Software möglich sein.

Durch weitere Expertengespräche sollen User-Storys erstellt werden um den Mehrwert des neuen Konzepts validieren zu können.

Um diesen Anforderungen gerecht zu werden, soll für CodeLeaves ein sprachunabhängiges Datenmodell entworfen werden, dass alle geforderten Metriken unterstützt.

Der Praktische Teil dieser Arbeit soll die prototypische Entwicklung von CodeLeaves für die HoloLens sein.

1.4 Aufbau der Arbeit

Im Kapitel 2 wird das Konzept von CodeLeaves vorgestellt. Dabei wird zunächst unter Betrachtung alternativer Ansätzen begründet, wieso ein neues Konzept sinnvoll ist, um dann in Abschnitt 2.1 genauer auf das Konzept einzugehen. Die Befragung von Experten der Softwareanalyse und die daraus abgeleiteten Anforderungen an CodeLeaves in Abschnitt 2.4 schließen das erste Kapitel ab.

Das Kapitel 3 beschäftigt sich mit der Entwicklung eines geeigneten Datenmodells für CodeLeaves. Es werden vorhandene Datenmodelle auf Tauglichkeit für CodeLeaves überprüft und Rücksprache mit erfahrenen Software-Ingenieuren gehalten.

Aufbauend auf das entwickelte Datenmodell, wird das Konzept von CodeLeaves in Kapitel 4 theoretisch weiter ausgearbeitet. Darunter fällt die Positionierung der Bäume auf einer Grundfläche und die Länge, Dicke und der Winkel der einzelnen Äste. Parallel zur Theorie wird aufgezeigt, wie sich CodeLeaves in Unity für die HoloLens modellieren lässt.

Die Interaktion mit CodeLeaves soll Thema des Kapitel ?? sein. Besonders die Abhängigkeiten von Artefakten einer Software sollen mithilfe von CodeLeaves interaktiv exploriert werden können.

Das Kapitel 7 fasst die Ergebnisse der Arbeit zusammen und ein Ausblick auf zukünftige Verwendung und weiterführende Arbeiten runden die Arbeit ab.

1.5 Abgrenzung

Die Beschaffung der Informationen über eine Software soll hier nur am Rande betrachtet. Für die Analyse einer Software müssen statische und dynamische Informationen gesammelt und aggregiert werden. Dafür gibt es verschiedenste Tools, wie inhouse Entwicklungen der QAware mit dem *Software-EKG* und dem *QValidator* oder proprietäre und Open-Source-Software. Die Anbindung an CodeLeaves würde den Rahmen dieser Arbeit jedoch sprengen.

1.5 Abgrenzung

Stattdessen soll der Fokus besonders auf das grundlegende Konzept und das Frontend mit der Generierung und Interaktion des Waldes gelegt werden. Diese Arbeit stellt jedoch den Anspruch, dass mit realistischen Daten gearbeitet wird, um eine valide Einschätzung des Mehrwerts von CodeLeaves geben zu können.

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 auch bei der Struktur einer Software. Jede Software mit einer geschachtelten Paketstruktur ist hierarchisch und kann in einer Baumstruktur 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. Daraus folgt, dass die Darstellung von Software als Baum oder auch Bäume sinnvoll ist.

Der Baum in der Informatik zeugt von einer ursprünglichen Metapher – der Baum, wie er draußen in der Natur wächst. Da dieser unbestritten dreidimensional ist, liegt eine Softwarevisualisierung für die Dreidimensionalität mit einer realitätsnäheren Interpretation der Baum-Metapher nahe. Werden die Bäume, die im zweidimensionalen meist von oben nach unten gezeichnet wird, 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, dass sich die Metapher des Software-Waldes zu nutze macht und wird im Folgenden auf High-Level-Ebene skizziert und danach genauer erläutert:

Konzept: CodeLeaves
<ol style="list-style-type: none">1. Die Struktur der Software wird mit nach oben wachsenden Bäumen dargestellt.2. Jedes Paket im <i>Root-Verzeichnis</i> wird als einzelner Baum auf einer Ebene – dem <i>Waldboden</i> – dargestellt.3. Die Blätter der Bäume entsprechen den kleinsten betrachteten <i>Softwareartefakten</i> (siehe Definition ??) und können durch ihre Farbe eine beliebige Metrik visualisieren.4. Zwischen den Bäumen entsteht ein <i>Wurzelgeflecht</i>, was die aggregierten Abhängigkeiten zwischen den Paketen darstellt.5. Abhängigkeiten oder Aufrufe zwischen einzelnen Softwareartefakten werden aggregiert über die Elternpakete als farbige bzw. dicke Äste dargestellt oder alternativ als

2 Das Konzept CodeLeaves

direkte *Spinnweben* zwischen den Bäumen.

Punkt 1 ist dafür verantwortlich, dass die Softwarevisualisierung nahe an der tatsächlichen Software ist, wie sie ein Entwickler in seiner Code-Base gewöhnt ist. Das heißt, diejenigen, die auch tatsächlich mit der Software arbeiten, finden sich aufgrund der bekannten Baumstruktur auch in der Visualisierung schnell zurecht, ohne jedes Softwareartefakt auszuwählen, um zu sehen, mit welchem sie es zu tun haben. 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 [45].

Punkt 2 ist weitgehend selbsterklärend. Durch Darstellung der Software in mehreren Bäumen entsteht erst der Software-Wald und die Pakete im Root-Verzeichnis als Wurzeln der Bäume zu verwenden bietet sich an. Das schließt jedoch nicht aus, dass Unterpakete als neuer Waldboden verwendet wird. Interaktion mit dem Waldboden soll Teil des Kapitel ?? sein.

Punkt 3 bedeutet, dass in CodeLeaves durch die Farbe der Blätter ein *Laubdach* entsteht, dass eine gute Übersicht über eine ausgewählte Metrik der Software gibt. Beispielsweise kann der Farbe der Blätter die Code-Coverage der einzelnen Softwareartefakte (siehe Definition 2.1) 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 Coverage eher in den Herbst übergehen. Die Färbung des Laubdachs ist flexibel auf jegliche Metrik anwendbar, die bei der betrachteten Software zur Verfügung steht.

Definition 2.1: Softwareartefakt

Ein *Softwareartefakt* wird in dieser Arbeit als Überbegriff für eine definierte Einheit einer Software verwendet. Das sind typischerweise Pakete und Klassen. Das kleinsten betrachteten Softwareartefakte sind die Einheiten, die nicht weiter unterteilt wird. In den meisten Softwarevisualisierungen sind dies Klassen, könnten bei feinerer Granularität aber auch Methoden oder sogar Blöcke sein.

Punkt 4 bietet einen großen Vorteil gegenüber der Zweidimensionalität. In Abbildung 2.1 wird rechts das Prinzip des Wurzelgeflechts mit einem minimalistischen Beispiel illustriert.

Die Abhängigkeiten zwischen den Bäumen wird auf einen Blick ersichtlich. Betrachtet man die Bäume in 2D, wie es im linken Teil der Abbildung dargestellt ist, verschwinden die Abhängigkeiten hintereinander und sind nicht zuzuordnen.

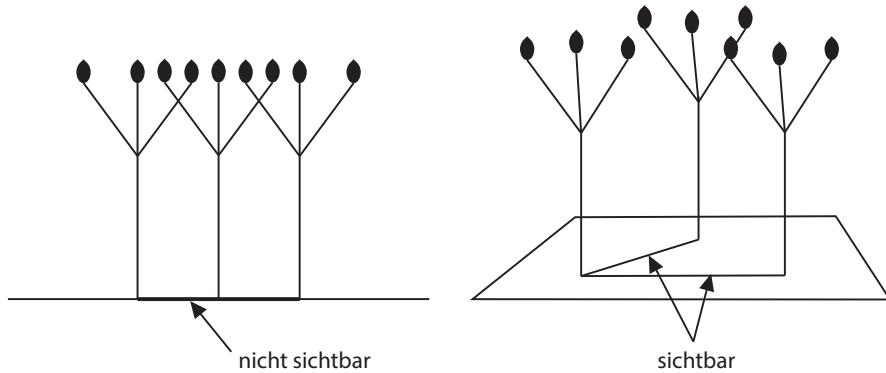
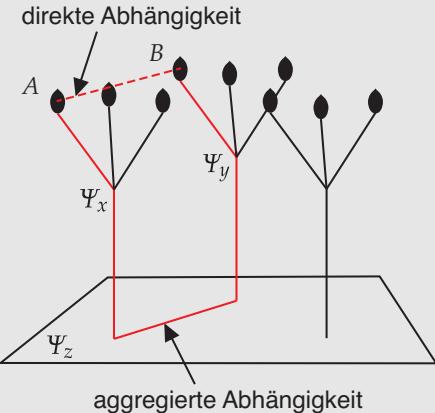


Abbildung 2.1 Vorteil der Dreidimensionalität bei der Darstellung von Abhängigkeiten

Punkt 5 heißt, dass die Abhängigkeiten einer Software sehr übersichtlich auf die Struktur der Software abgebildet werden können, ohne diese zu beeinflussen. Für das besser Verständnis bedarf es einer Definition der Aggregation von Abhängigkeiten.

Definition 2.1: Aggregation von Abhängigkeiten

Bei der Aggregation von Abhängigkeiten zwischen Softwareartefakten werden die Abhängigkeiten nicht direkt dargestellt, sondern über die Eltern-Pakete geleitet. Seien x, y Pakete und Softwareartefakt $A \in x$ besitze eine Abhängigkeit zu Softwareartefakt $B \in y$, dann geht die Abhängigkeit von A zu einem zusätzlich Konstrukt Ψ_x , das das Pakets x repräsentiert. Angenommen x und y befinden sich zudem im Paket z , dann geht die aggregierte Abhängigkeit entweder direkt von Ψ_x zu Ψ_y , oder weiter über Ψ_z zu Ψ_y und schließlich B .



Auf der rechten Seite der Definition 2.1 ist eine aggregierte Abhängigkeit am Beispiel von CodeLeaves zu sehen. Die Ψ in CodeLeaves sind die Knoten der Bäume, an denen die Äste zusammen laufen und im Spezialfall des Root-Verzeichnisses, der Waldboden.

Bei mehreren Abhängigkeiten zwischen Nachbar-Paketen, überlagern sich die aggregierten Abhängigkeiten zwangsläufig. Falls in unserem Beispiel eine weitere Abhängigkeit von Paket x zu Paket y bestünde, würden sich die Abhängigkeiten von Ψ_x bis Ψ_y überlagern. Daraus ergibt sich, dass nicht jede aggregierte Abhängigkeit ohne Interaktion zwangsläufig eindeutig zuzuordnen ist.

Wird aber bei jeder Kante, sei es ein Ast, Stamm, oder Wurzel, die Anzahl an

2 Das Konzept CodeLeaves

überlagernden Abhängigkeiten als Dicke der Kante dargestellt, bekommt der Betrachter eine gute Übersicht über die Gesamtheit der Abhängigkeiten. Durch Interaktion mit einzelnen Kanten, oder sogar des ganzen Waldbodens, soll eine fein granulärere Analyse der Abhängigkeiten möglich sein.

Das zweite Element von Punkt 5 sind die Spinnweben. Damit lassen sich die Abhängigkeiten direkt darstellen. Um bei großen Software-Systemen aber die Übersicht über den Wald nicht zu verlieren, sollten diese mithilfe der Interaktion des Nutzers flexibel aktivierbar sein.

Nachdem das grundlegende Konzept von CodeLeaves verstanden ist, muss eine Begriffsdefinition ein Einheitliches Verständnis der verwendeten Komponenten schaffen. Die eingeführten Bezeichnungen dienen dem Verständnis der verwendeten Datenmodelle im nächsten Kapitel, sowie die entwickelten Layout-Algorithmen in Kapitel 4.

Die Struktur von CodeLeaves baut stark auf der Datenstruktur eines klassischen Baumes auf – „*the most important nonlinear structures that arise in computer algorithms*“ [26] – die zum Beispiel in [26, 16, 23] definiert wird.

Da aber die Struktur von CodeLeaves zum Teil von der klassischen Definition abweicht und teilweise neue Bezeichnungen benötigt, betrachten wir daher einige Definitionen zur Baumstruktur, um diese an die Gegebenheiten von CodeLeaves anzupassen und führen neue Begriffe ein.

2.2 Begriffsklärung

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

2.3 Vergleich mit anderen Konzepten

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*

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*) sind alle nachfolgenden Kanten und Knoten des Knotens A ausschließlich des Knotens A selbst.

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

- Der Kronenansatz ist durch die Kante namens *Stamm* (engl.: *trunk*) mit den *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* besteht aus einer disjunkten Menge an Bäumen und dem Waldboden.
- Ein Waldboden eines Waldes mit n Bäumen besitzt 0 bis $n!$ *Wurzeln* (engl. *roots*), die jeweils zwei Stammbasen miteinander verbinden.

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 3D Visualisierungen und andere mögliche Konzepte miteinander verglichen. Die Ergebnisse dieser Untersuchung, soll in Folgenden vorgestellt werden.

Um herauszufinden welchen Mehrwert sich Nutzer einer Softwarevisualisierung von dieser versprechen, wurde eine Umfrage in der QAware GmbH 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

2 Das Konzept CodeLeaves

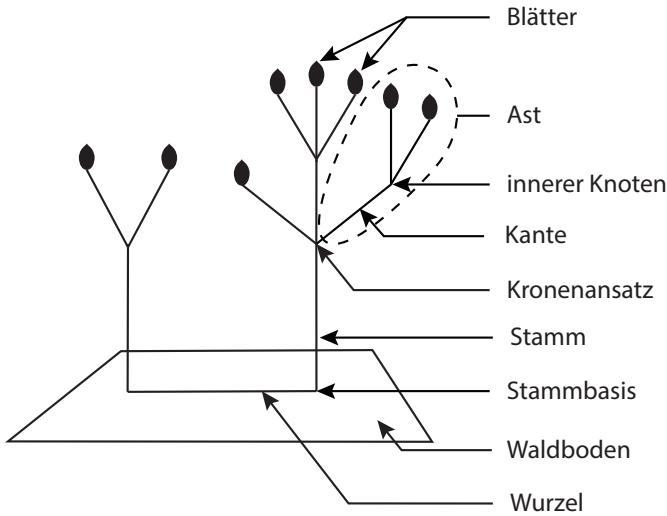


Abbildung 2.2 Bezeichnungen

insgesamt 22 Mitarbeiter mit unterschiedlichen Rollen in der Softwareentwicklung befragt.

In Abbildung 2.3 ist zu sehen, wie oft welche Metriken genannt wurden. Alle Metriken lassen sich in die drei Kategorien der Softwarevisualisierung aus [14] einordnen und sind farbig entsprechend gruppiert.

Statik sind die Informationen, die ohne die Ausführung der Software generiert werden können [14]. Darunter fallen die Metriken, die als Zahl zu jedem Softwareartefakt zugeordnet werden können und damit zueinander in Relation gesetzt werden können. Genannt wurden LOC, Komplexität, Coverage und Code-Violations. Letzteres sind beispielsweise Verletzungen von vereinbarten *Code-Conventions*. Die Informationen, die komplizierter zu visualisieren sind, stellen die Struktur und die Abhängigkeiten dar. 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 von Softwareartefakten und Anzahl von Aufrufen genannt. Damit sind beispielsweise Bottlenecks identifizierbar. Auch die Darstellung der Laufzeitfehler einer fehlerhaften Software sind für deren Analyse wichtig. Die Resourcen-Auslastung ist dabei auch 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. Mit der Evolution eines Themas ist gemeint, dass anhand

2.3 Vergleich mit anderen Konzepten

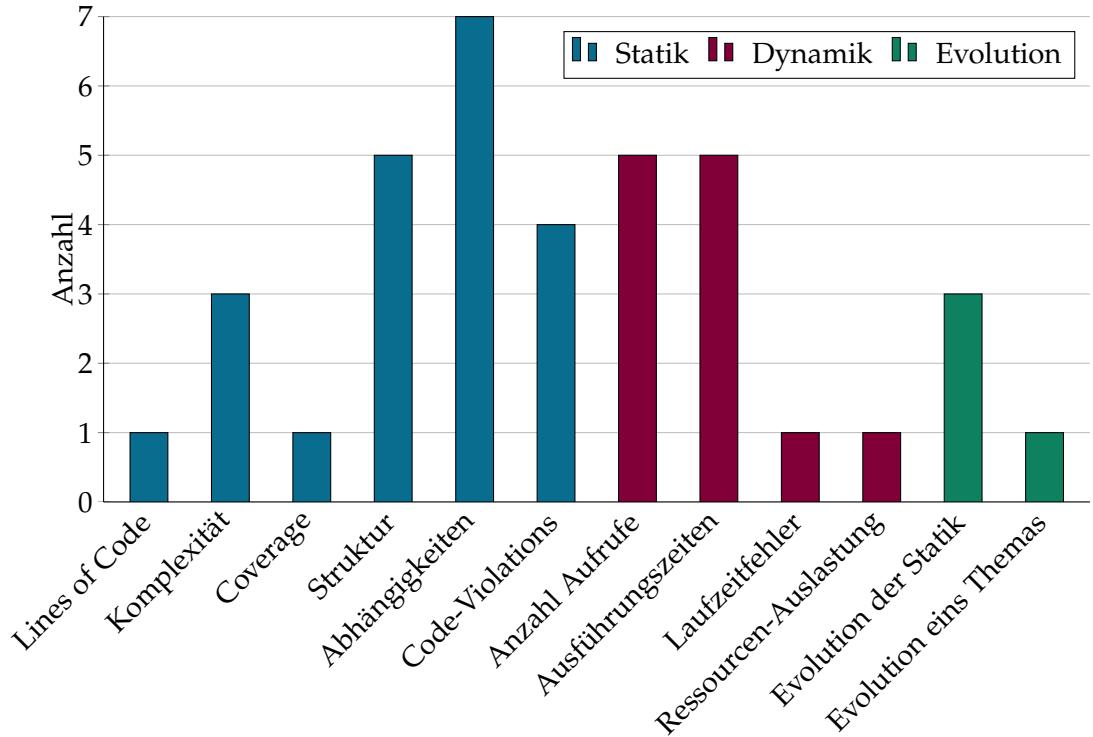


Abbildung 2.3 Von Mitarbeitern der QAware gewünschte Informationen

die Entwicklung eines bestimmten Themas nachverfolgt werden kann.

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

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

2 Das Konzept CodeLeaves

CodeCity

2007 stellten Wettel et al. CodeCity vor, die mithilfe der *Stadt-Metapher* dreidimensionale Städte visualisiert, in denen Klassen als Gebäude und Pakete als Stadtviertel dargestellt werden [45, 46, 47]. 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)*) der visualisierten 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 [47].

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].

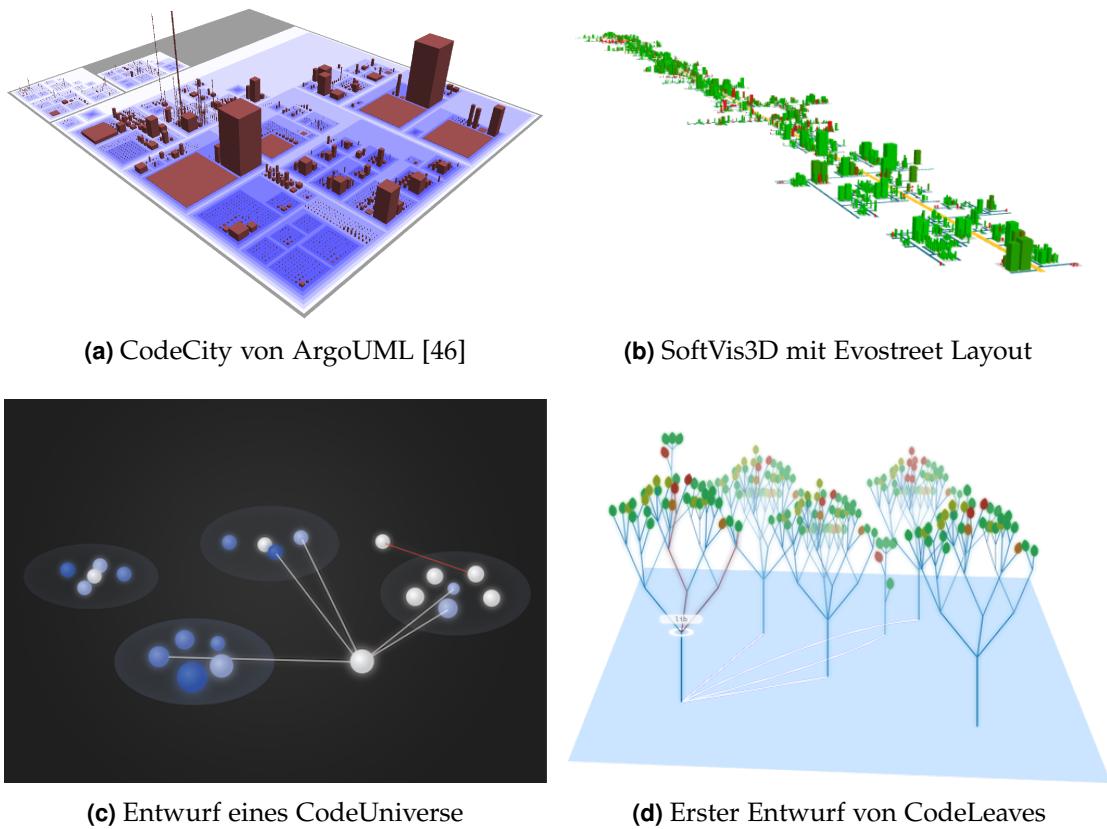


Abbildung 2.4 CodeLeaves und alternative Modelle

2.3 Vergleich mit anderen Konzepten

SoftVis3D

SoftVis3D greift das Konzept der CodeCity auf und visualisiert als Plugin für SonarQube¹ 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. 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 Straßen dargestellt.

Die Evolution wird von SoftVis3D trotz Evostreet-Layout jedoch nicht unterstützt. Bei Die Abhängigkeiten wurde in früheren Versionen aggregiert dargestellt. Dafür wurden Pakete im Distrikt-Layout in übereinander liegenden Ebenen abgebildet und für Ψ (vgl. Definition 2.1) ein Hilfsgebäude benutzt, dass zu der darüberliegenden Ebene führt. Dadurch ging jedoch die Stadt-Metapher und die Übersichtlichkeit 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 der Arbeit [22, 3], wird die Software als Universum dargestellt. Die Softwareartefakte in Paketen 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 andere Konzepte. Bei der Visualisierung der Abhängigkeiten stößt das CodeUniverse aber an seine Grenzen. Durch direkte Verbindungen zwischen den Sternen lassen sich zwar Abhängigkeiten darstellen, 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. Deshalb wird in [3] ein Konzept entworfen, dass die Softwareartefakte mit einem „hierarchischen Netz“ verbindet. Dieses ist nichts anderes als die vorhandene Baumstruktur der Software und hat mit der Metapher des Universums auch 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 weitere, haben gemein, dass sie zum einen Struktur, Dynamik und Evolution nicht vereinen. Zum anderen können Abhängigkeiten oder dynamische Aufrufe zwischen Softwareartefakten nicht ohne Verlust der Übersichtlichkeit angezeigt werden. CodeLeaves soll alle drei Kategorien der Softwarevisualisierung

¹SonarQube ist eine open-source Plattform für statische Code-Qualität, <https://www.sonarqube.org/>

2 Das Konzept CodeLeaves

unterstützen und ist bei der Visualisierung der Struktur und der Abhängigkeiten den Alternativen überlegen. Durch die Baumstruktur, wie er auch in der Code-Base vorhanden ist, wird die Paket-Struktur eins zu eins wiedergegeben. Die aggregierten Abhängigkeiten lehnen sich an die Struktur an und beeinflussen diese nicht negativ. Durch das Wurzelgeflecht und die Spinnweben wird die Dreidimensionalität optimal ausgenutzt.

2.4 Anforderungen an CodeLeaves

Die Umfrage, die in Abschnitt 2.3 vorgestellt wurde, ergab einen guten Stimmungsbarometer über die Wünsche im Bezug auf zu visualisierende Informationen. In diesem Abschnitt soll genauer auf die Anforderungen an einen Code-Leaves eingegangen werden. Dazu wurden von der QAware zwei Experten der Softwareanalyse befragt, um User-Storys und Akzeptanzkriterien auf zu stellen.

Aus der dynamischen Analyse wurde der promovierende Performance-Analyst F. Lautenschlager befragt. 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 Sprecher auf internationalen Konferenzen geladen.

Als Experte der statischen Analyse wurde J. Weigend befragt. 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 und in den Tabellen 2.1 – 2.5 zu den dazugehörige Akzeptanzkriterien zusammengefasst. Bei den Akzeptanzkriterien dient die ID der Nachverfolgbarkeit und der Bezugnahme im weiteren Text. Die letzte Spalte gibt die Priorität des Akzeptanzkriteriums an, welche besonders für das Prototyping von Bedeutung ist.

Prototypen

Für das Prototyping wird überwiegend mit einem *High-Fi Prototyp*² für die HoloLens gearbeitet. Dieser Prototyp kann in der Augmented Reality erlebt werden und auch die Interaktion mit der HoloLens können erprobt werden. Die Inhalte, die in Kapitel 4 beschrieben werden, und einige Interaktionen aus Kapitel 6 sind im High-Fi Prototyp enthalten.

Weitere Elemente werden in Kapitel 6 mit einem Papier *Low-Fi Prototyp*³ ergänzt.

²englisch 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].

³englisch 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].

2.4 Anforderungen an CodeLeaves

Prioritäten

Bei den Akzeptanzkriterien wird zwischen drei verschiedenen Prioritäten unterschieden:

Priorität 1: Grundfunktionalität, die für die Beurteilung des Mehrwertes von CodeLeaves unerlässlich ist und im High-Fi Prototyp implementiert wird.

Priorität 2: Funktionalität die konzeptionell ausgearbeitet wird und im Low-Fi Prototyp enthalten ist.

Priorität 3: Funktionalität, die für die Beurteilung des Mehrwertes von CodeLeave nicht essentiell ist.

Im Folgenden werden nun User-Storys und Akzeptanzkriterien aufgeführt.

Userstory 1: Code-Qualität

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

Tabelle 2.1 Akzeptanzkriterien zu User-Story 1

ID	Beschreibung	Prio
1.1	Ich kann statische Metriken der Code-Qualität, wie z.B. Code-Coverage, auf die Farbe der Blätter anwenden.	1
1.2	Ich kann Ausreißer der ausgewählten Metrik gut erkennen.	1
1.3	Ich kann mir die genaue Zahl der aktuellen Metrik für eine bestimmte Klasse anzeigen lassen	1
1.4	Ich möchte möglichst viele Blätter gleichzeitig betrachten können.	1

Userstory 2: Kopplung und Struktur

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

2 Das Konzept CodeLeaves

Tabelle 2.2 Akzeptanzkriterien zu User-Story 2

ID	Beschreibung	Prio
2.1	Die Struktur der Bäume entspricht der der Software.	1
2.2	Die Struktur der Bäume ist eindeutig und hat keine Überschneidungen.	1
2.3	Wenn ich ein Blatt auswähle, dann kann ich die ein- bzw. ausgehenden Abhängigkeiten hervorheben.	2
2.4	Die Dicke einer Kante oder Wurzel zeigt mir die Anzahl der dazugehörigen Abhängigkeiten.	2
2.5	Wenn ich eine Kante oder Wurzel auswähle, dann kann ich die von den Abhängigkeiten betroffenen Blättern hervorheben.	2
2.6	Wenn ich eine Kante oder Wurzel auswähle, dann kann ich sehen wie viele Abhängigkeiten in welche Richtung fließen.	2
2.7	Ich kann den Zeitpunkt auswählen, auf die sich die dargestellten Daten beziehen.	2
2.8	Zyklische Verbindungen können hervorgehoben werden.	3

Userstory 3: Drilldown

Als *Softwarearchitekt*
möchte ich *einen intuitiven Drilldown haben,*
um die Zusammenhänge auf verschiedener Pakete-Ebene zu verstehen.

Tabelle 2.3 Akzeptanzkriterien zu User-Story 3

ID	Beschreibung	Prio
3.1	Wenn ich eine Kante auswähle, dann kann ich das repräsentierte Softwareartefakt als neuen Waldboden festlegen.	2

Userstory 4: Dynamische Metriken

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.

2.4 Anforderungen an CodeLeaves

Tabelle 2.4 Akzeptanzkriterien zu User-Story 4

ID	Beschreibung	Prio
4.1	Ich kann Antwortzeiten und Laufzeitfehler auf die Blattfarbe anwenden.	1
4.2	Ich kann Ausreißer der ausgewählten Metrik gut erkennen.	1
4.3	Ich kann mir die genaue Zahl der aktuellen Metrik für eine bestimmte Klasse anzeigen lassen	1
4.4	Wenn ich ein Blatt auswähle, dann kann ich die ein- bzw. ausgehenden Aufrufe hervorheben.	2
4.5	Die Dicke einer Kante oder Wurzel zeigt mir die Anzahl der dazugehörigen Aufrufe.	2
4.6	Wenn ich eine Kante oder Wurzel auswähle, dann kann ich die von den Aufrufen betroffenen Blättern hervorheben.	2
4.7	Wenn ich eine Wurzel oder Kante anklicke, dann kann ich sehen wie viele Verbindungen in welche Richtung gehen.	2
4.8	Ich kann den Zeitraum auswählen, auf die sich die dargestellten Daten beziehen.	2

Userstory 5: Zustand der Software

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

Tabelle 2.5 Akzeptanzkriterien zu User-Story 5

ID	Beschreibung	Prio
5.1	Wenn ich die Laubfarbe des Waldes betrachte, dann möchte ich eine möglichst gute Gesamtübersicht über die ausgewählte Metrik haben.	1
5.2	Die Struktur der Bäume ist möglichst übersichtlich und hat so wenig Überschneidungen wie möglich.	1

Es wird deutlich, dass sich viele Akzeptanzkriterien aus der statischen und dynamischen Analyse überschneiden. Das lässt erkennen wie sinnvoll es ist diese beiden Sichten miteinander zu kombinieren. Ob es sich um statische oder dynamische Metriken handelt, spielt für die Visualisierung keine Rolle. Genauso funktioniert die Aggregation und die Interaktion mit Kanten und Wurzeln für beide Sichten gleich.

2 Das Konzept CodeLeaves

Für den High-Fi Prototyp wird deshalb beispielhaft nur mit einer statischen Metrik gearbeitet. Mit dem Datenmodell, dass im Kapitel 3.1 entworfen wird, und entsprechender Datenbeschaffung können aber genauso dynamische Metriken und Verbindungen visualisiert werden.

Ist also ein Akzeptanzkriterium aus der statischen Analyse erfüllt, kann aus Sicht der Visualisierung auch das Pendant der dynamischen Analyse als erfüllt angesehen werden.

User-Storys zur Benutzbarkeit von CodeLeaves

Aus anfänglichem Prototyping mit der HoloLens sind weitere User-Story entstanden, ohne die die anderen nicht sinnvoll umsetzbar sind. Somit lautet die 6. User-Story wie folgt:

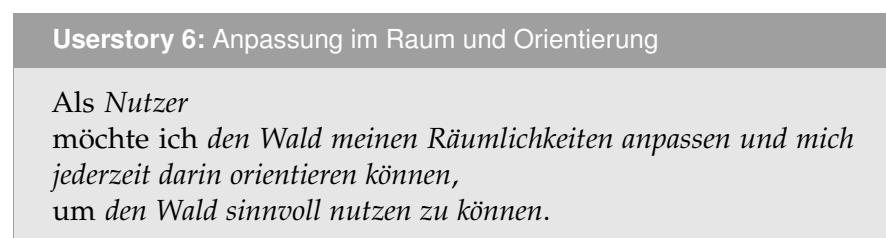


Tabelle 2.6 Akzeptanzkriterien zu User-Story 6

ID	Beschreibung	Prio
6.1	Ich kann den Wald im Raum platzieren.	1
6.2	Ich kann den Wald skalieren.	1
6.3	Ich kann den Wald rotieren.	1
6.4	Ich kann Elemente im Wald mit dem Namen der repräsentieren Softwareartefakte identifizieren.	1

Akzeptanzkriterium 6.1 aus Tabelle 2.6 ist vor allem am Anfang wichtig, um den Wald dort 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 die Betrachtung von Details ist auch Akzeptanzkriterium 6.2 essentiell. Akzeptanzkriterium 6.4 ist eine Funktionalität, die durchgängig benötigt wird. Zwar können Pakete durch die Struktur der Bäume Pakete erkannt werden, jedoch ist die Anzeige der Namen der Softwareartefakte unerlässlich, um genau zu wissen, welches Element des Waldes was repräsentiert.

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

2.4 Anforderungen an CodeLeaves

2.7 deren Akzeptanzkriterien.

Userstory 7: Auswahl eines Projekts
Als Nutzer möchte ich <i>ein bestehendes Projekt öffnen und neue importieren können,</i> <i>um das Projekt zu betrachten, das mich interessiert.</i>

Tabelle 2.7 Akzeptanzkriterien zu User-Story 7

ID	Beschreibung	Prio
7.1	Ich kann bereits gespeicherte Projekte öffnen.	1
7.2	Ich kann ein neues Projekt aus unterstützten Datenquellen importieren.	2
7.3	Ich kann Daten zu einem bereits gespeicherten Projekt aus unterstützten Daten hinzufügen.	2

Alle Akzeptanzkriterien mit Priorität 1 werden im High-Fi Prototyp umgesetzt und ergeben gleichzeitig einen technischen Durchstich für CodeLeaves. Dabei wird besonders Wert auf das Layout des Waldes gelegt, das in Kapitel 4 erarbeitet wird, da ohne die übersichtliche Struktur jegliche Weiterentwicklung von CodeLeaves sinnlos wäre.

Im weiteren Verlauf des Textes wird bei der Umsetzung eines Akzeptanzkriteriums darauf in folgender Form verwiesen: (Akz. <ID>).

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

3 Datenmodell

3.1 Trennung in Schichten

Das Datenmodell für CodeLeaves nimmt einen zentralen Baustein ein, auf den sich die Implementierung stützt. Bei dem Entwurfsmuster sind einige Prinzipien zu beachten. Wie Buschmann et al. in [19] beschreibt, 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 es eine Software ist, 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. Im Backend Layer werden die Daten über eine Software aggregiert, seien es statische oder dynamische Informationen und in ein Sprach-agnostisches Format gebracht. Die Konnektoren, d.h. die Verbindungen zu den verschiedenen Datenquellen, können dann leicht ausgetauscht werden.

Wie in Abschnitt 1.5 bereits festgehalten, soll der Schritt des Transfers der realen Software in deren Repräsentation jedoch nicht Schwerpunkt dieser Arbeit sein. Das Datenmodell ist hingegen sehr wohl zu definieren, um die Daten darstellen zu können.

Application logic

Die Einteilung in Schichten geht nach der Transformation der Software in ein Meta-Modell aber noch weiter. CodeLeaves könnte neben Software prinzipiell auch jegliche anderen, hierarchisch strukturierten Informationen darstellen. Für die interne Datenhaltung wird deshalb in der Schicht *Application logic* das Datenmodell weiter abstrahiert und in ein Format gebracht, dass sich stark an der Baumstruktur der Informatik orientiert. So könnten später auch leicht gänzlich andere Daten angebunden werden.

Presentation

Die letzte Schicht ist die der *Presentation*. Das Datenmodell für hierarchische Informationen muss für die Darstellung der Bäume weiter verarbeitet werden. Zum Beispiel

3 Datenmodell

muss für die rekursive Generierung der Bäume die Höhe der Knoten bekannt sein. Nur so kann das Paket mit der tiefsten Schachtelung auch sinnvoll als Verlängerung des Stammes dargestellt werden. Auch die Aggregation der Verbindungen zwischen einzelnen Blättern findet in diesem Schritt statt, sodass in den Zeichenalgorithmen, mit denen die Bäume in Unity generiert werden, keine fachliche Logik mehr benötigt wird. Das entstehende UI-Datenmodell wird nur zum Rendern in Unity verwendet und ist unveränderlich.

Das bedeutet auch diese Schicht könnte wieder ausgetauscht werden und unterschiedliche Zeichenalgorithmen könnten unterstützt werden.

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

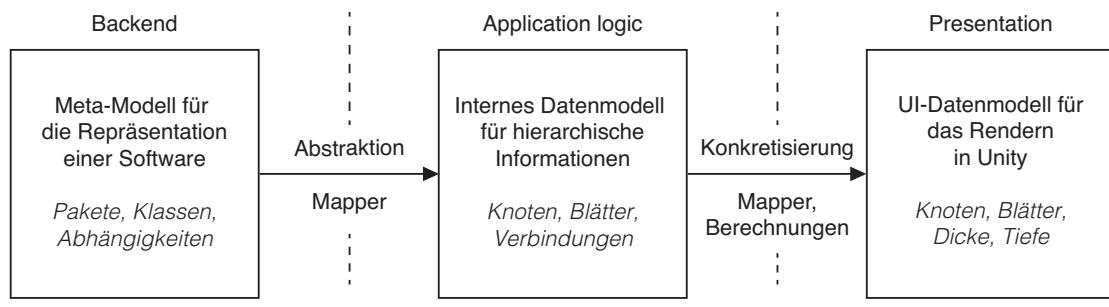


Abbildung 3.1 Schichten der Datenstrukturen

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 heut in Version 3.0 verfügbar und in [28] beschrieben. Das Meta-Modell ist sehr umfangreich und beinhaltet 51 verschiedene Entities. Es wird in einem eigenen Dateiformat namens MSE, vergleichbar mit JSON, 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 simpel 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.

3.2 Software Meta-Modell

Aus den User-Storys 1 und 4 ergeben sich folgende Metriken, die auf die Blattfarbe angewandt werden kann:

- Statische Metriken zur Code-Qualität
- Antwortzeiten
- Laufzeitfehler

Aus User-Story 1 und 3 lassen sich folgende mögliche Verbindungen ableiten:

- Statische Abhängigkeiten
- Dynamische Aufrufe

Diese Metriken und Verbindungen müssen auf die Struktur der Software, die in User-story 3 gefordert wird, 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 *SoftwareArtifact*, *Metric* und *Association* zu finden. Die *Metric*-Klasse ist ein Schlüssel-Wert-Paar, das durch *CodeSnippets* ergänzt werden kann. So ist es zum Beispiel möglich bei Laufzeitfehler zu sehen, an welcher Stelle die Fehler aufgetreten sind. Der Schlüssel identifiziert die Metrik, wie zum Beispiel statische Testabdeckung oder dynamische Laufzeitfehler und der Wert 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 mit der einzelnen Klasse *SoftwareArtifact*, das eine Liste von Kindern hat. Diese Liste ist für ein Blatt leer. Grund für diese Modellierung ist die bessere Möglichkeit der Sterilisierung. Das Kompositum-Muster, wie es in Abbildungen 3.3 und 3.4 zu sehen ist, beinhaltet eine abstrakte Klasse, die das Blatt und das Kompositum spezialisieren. Für die Serialisierung 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.

Bei einem Softwareartefakt spielt es keine Rolle, wie die Software organisiert ist. Egal ob ein ganzes Projekt, Paket oder eine einzelne Methode, alles wird zu einem Softwareartefakt.

Jedes Objekt der Klasse *SoftwareArtifact* hat einen eindeutigen Schlüssel, mit dem das Softwareartefakt eindeutig identifiziert werden kann und einen Namen, der für den Nutzer lesbar ist. Daneben kann bei einem Softwareartefakt beliebig viele Metriken gespeichert werden. Für die weiter Verarbeitung in CodeLeaves ergibt dies jedoch nur bei Softwareartefakten ohne weitere Kinder Sinn.

3 Datenmodell

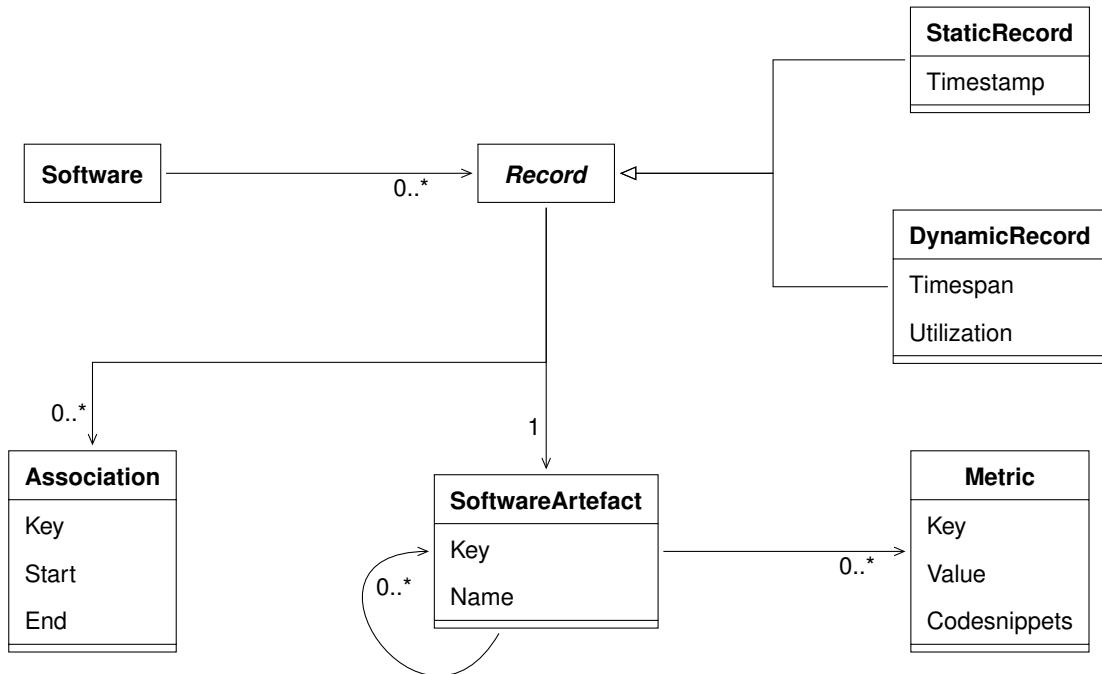


Abbildung 3.2 Meta-Modell einer Software

Mit der *Association*-Klasse können jegliche Verbindungen wie statische Abhängigkeiten oder dynamische Aufrufe gespeichert werden. Es besteht keine direkte Assoziation mit der Klasse *SoftwareArtifact* um zyklische Abhängigkeiten zu vermeiden. Stattdessen werden Start- und Endpunkt der Verbindung mit dem Schlüssel der zugehörigen Softwareartefakte angegeben.

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

Neben der Struktur, Metriken und den Verbindungen einer Software muss der Zeitraum bzw. der Zeitpunkt angegeben werden können, um die Software in unterschiedlichen Stadien betrachten zu können und somit die Evolution zu unterstützen.

Deshalb werden Struktur und Verbindungen in einem Datensatz – der *Record*-Klasse gespeichert. Eine Software wiederum kann beliebig viele Datensätze enthalten. Bei einem Datensatz muss zwischen einem statischen und einem dynamischen 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 werden.

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 dem

3.3 Internes Datenmodell

Attribut *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*, ein *SoftwareArtifact* zum *Node* und eine *Metric* zu *LeafData*.

Neben der semantischen Transformation wird das *SoftwareArtifact* durch ein Kompositum-Pattern ersetzt. Dies reduziert die Logik von Algorithmen, indem nicht immer auf eine leere Liste getestet werden muss, sondern Logik in den Spezialisierungen realisiert werden kann. Die Metriken sind im internen Datenmodell auch nur noch bei Blättern verfügbar.

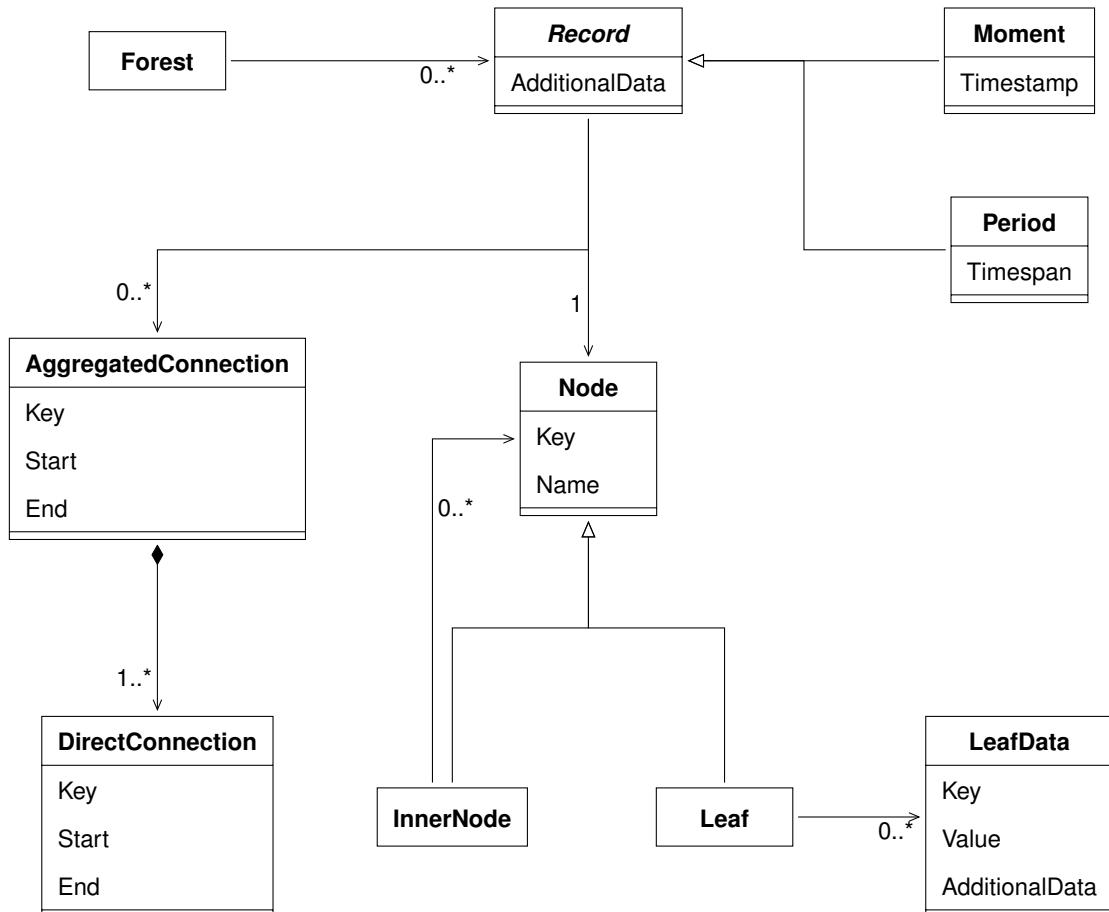


Abbildung 3.3 Internes Datenmodell

Beim Transfer in das interne Datenmodell werden ebenfalls die einzelnen Verbindungen aggregiert. Eine *AggregatedConnection*, die genau eine Kante lang ist, besteht

3 Datenmodell

aus mindestens einer direkten Verbindung.

Die *Utilization* aus dem *DynamicRecord* wandert als *AdditionalData* in die Generalisierung, da bei andern hierarchischen Daten eventuell auch für einen Zeitpunkt zusätzliche Daten relevant sein können.

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 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 beeinflusst. Es wurde darauf geachtet, dass das Modell nur Daten beinhaltet, das auch zwangsläufig für das Rendern der Visualisierung nötig ist.

Das ist bei einem Blatt die Farbe und bei einem inneren Knoten die Dicke der dazugehörigen Kante. Beide Spezialisierungen eines Knotens 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 technisch sinnvoll umgesetzt wird, ist mit reaktiver Programmierung 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 in der Farbe des Waldbodens wieder. Da diese zusätzlichen Informationen keine hohe Priorität haben, werden sie im Prototyping in Kapitel 4 und 6 keine Relevanz finden.

Mit der Methode *Find* der *UiNode*-Klasse kann jeder Knoten im Ui-Modell anhand seines Schlüssels gefunden werden. Dies ist für die Interaktion mit Konten essentiell. Auf das Attribut *Circle* bzw. dem *Kreis* eines Knotens, wird im Zuge des Layout des Waldes im nächsten Kapitel eingegangen. Für das Layout ist es außerdem nötig die Knoten sortieren zu können. Die abstrakten Methoden der *UiNode*-Klasse sind dazu erforderlich und werden entsprechend in der *UiInnerNode*- und *UiLeaf*-Klasse realisiert.

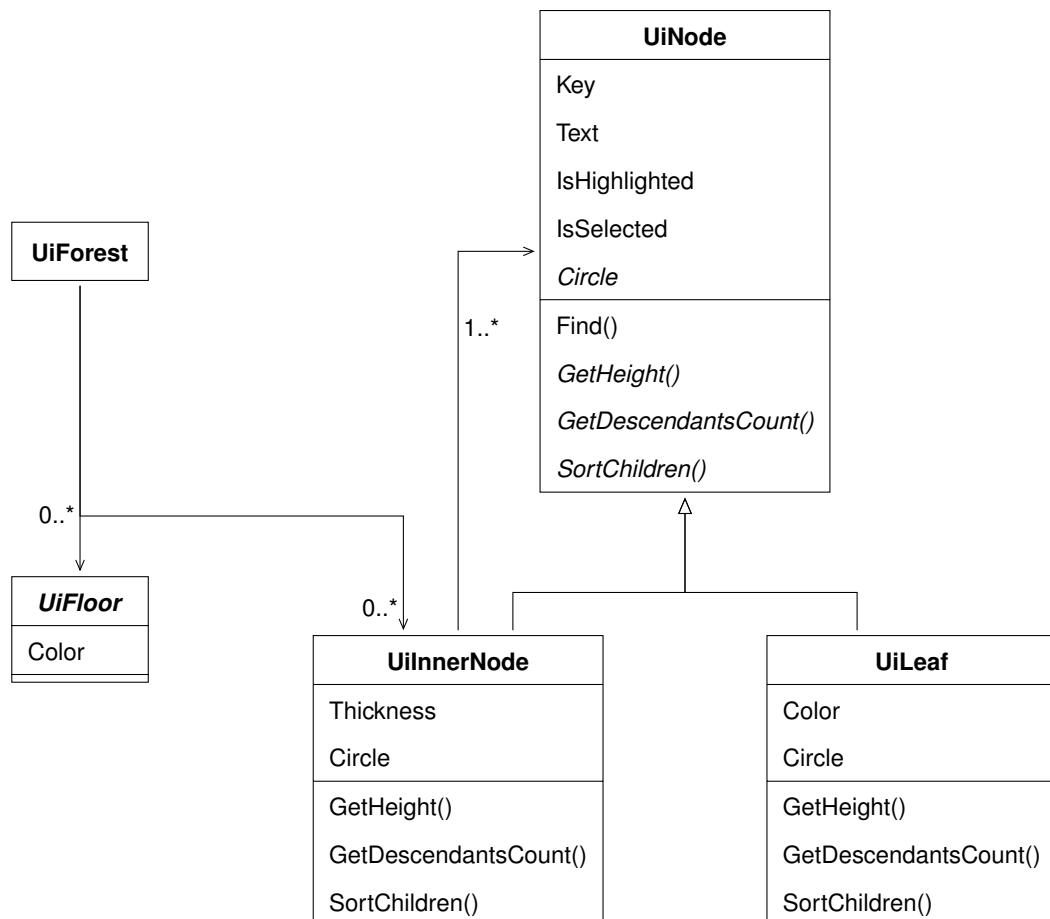


Abbildung 3.4 UI-Datenmodell

4 Modellierung und Layout des Waldes

4.1 Grundlegender Ansatz

CodeLeaves steht und fällt mit der korrekten und übersichtlichen Darstellung der Baumstruktur. Wegen Akzeptanzkriterien 1.4 und 5.1 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 2.2 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 ermöglicht. Für das Rendern in 3D müssen zunächst 3D Objekte entworfen werden, die die Knoten und Blätter repräsentieren. Dies wurde mithilfe der professionellen 3D Modellierungs-Software Maya¹ von Autodesk erreicht.

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. Ein Blatt - im Sinne der Begriffsdefinition von Abschnitt 2.2 - wird demnach vom dem dazugehörigen Blatt-Objekt und dem Kanten-Objekt zum Elternteil repräsentiert.

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

Blatt-Objekt Das Blatt-Objekt wurde in Anlehnung an ein echtes Blatt modelliert und ist in Abbildung 4.1b zu sehen.

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 jeder Postion möglich sein.

¹<https://www.autodesk.de/products/maya/overview>

4 Modellierung und Layout des Waldes

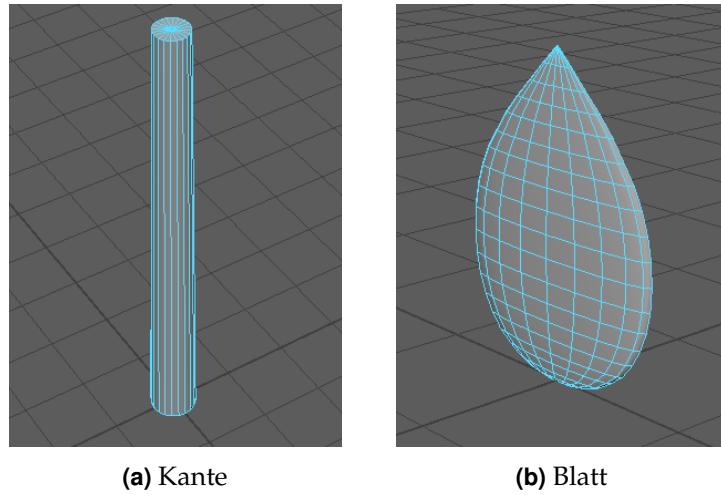


Abbildung 4.1 3D Objekte für den Prototyp

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 eine Script hinzugefügt, dass das Blatt zur Laufzeit immer in die Richtung des Betrachters rotieren lässt. Ein 3D-Objekt mit einer solchen Verhalten wird *Billboard* genannt. Mithilfe des Billboard-Verhaltens ist gewährleistet, dass der Nutzer von jedem Winkel aus immer eine optimale Übersicht über das Laubwerk besitzt.

Beschaffung von realistischen Beispieldaten

Für die Generierung der Struktur im Prototyp wurde mit einem Beispielprojekt gearbeitet. Das Projekt lautet *Air* und ist eines der größten Software-Projekte der QAware und wird seit rund 5 entwickelt.

In Tabelle 4.1 sind hinsichtlich der Struktur Eckdaten von Air angegeben.

Tabelle 4.1 Eckdaten des Beispielprojekts 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 Farben der Blätter abzufragen. Die Struktur wird bei der Abfrage der API rekursiv zusammengebaut, da SonarQube immer nur die direkten Kinder eines Softwareartefakts liefert.

4.1 Grundlegender Ansatz

Das Format von SonarQube wird dabei schon in das generische Software-Metamodell von CodeLeaves transferiert, was durch andere Daten später ergänzt werden kann. Da bei Air sehr viele HTTP-Requests entstehen, wird das entstandene Metamodell lokal in eine Datei geschrieben, sodass dieses als Beispiel-Datensatz für den Prototypen dient. Prinzipiell können aber auch dynamisch andere Projekte geladen werden.

Das Datenmodell des Backends wird in das Datenmodell der Application logic und weiter in das UI-Datenmodell transferiert.

Nun liegt eine Struktur vor, die es gilt zu Bäumen zusammen zu bauen. Der Grundlegende Ablauf des Algorithmus zur Generierung eines Baumes mit Kronenansatz K und dessen Kindern $\{(K_i) \mid i = 1, 2, \dots, n\}$ lässt sich wie folgt beschreiben:

- (i) Instanziere eine 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 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 werden und deren Kanten zum Elternteil rotiert und gestreckt werden soll im Folgenden erarbeitet werden.

Es stellt sich zunächst die Frage auf Grundlage welcher geometrischen Figur die Kinder verteilt werden. In Frage kommt eine Kugel und eine Kreisfläche.

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 Aststücke konstruiert. Ein solcher Fractal tree ist in Abbildung 4.2 dargestellt. Das Konzept ließe sich problemlos in die Dreidimensionalität übertragen. Das Problem dabei ist, dass bei diesem Ansatz die Aststücke 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 den kleiner werdenden Kanten und Blättern kaum möglich wäre. Auch die Überschneidungen, die in Abbildung 4.2 zu sehen sind, wären bei größeren Bäumen nur durch Neigung der Kanten nicht zu vermeiden. Die Kollisionen, die dadurch im 3D Raum entstünden, würden sich negativ auf die Übersichtlichkeit der Bäume auswirken und müssen deswegen vermieden werden.

Verteilung von Kindern auf einer Kreisfläche

Werden die Kinder stattdessen auf gleicher Höhe auf einer Kreisfläche überhalb des Elternteils positioniert, muss die Länge der Kanten entsprechend angepasst werden. Dadurch entstehen drei Vorteile.

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

4 Modellierung und Layout des Waldes

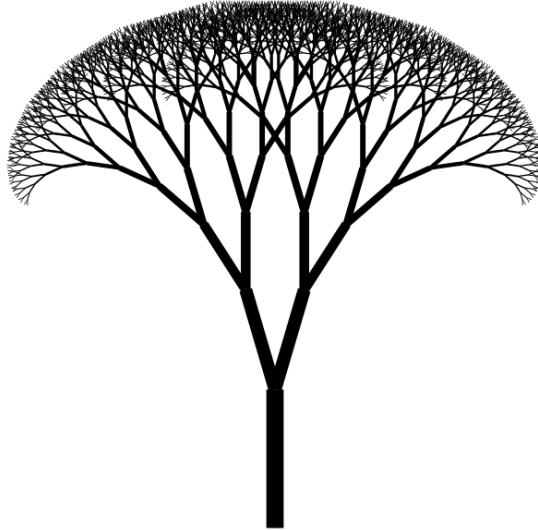


Abbildung 4.2 Fractal tree [39]

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

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

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

4.2 Verteilung von Blättern mit dem Sonnenblumen-Algorithmus

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

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 Knotens mit inneren Knoten, wird im 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 Befruchtung die Sonnenblumenkerne ausbilden, nutzen den Platz innerhalb der gelben Zungenblüten optimal aus [48]. Die spiralförmige Anordnung ist nicht nur ästhetisch, sondern folgt auch einem genauen Muster.

4.2 Verteilung von Blättern mit dem Sonnenblumen-Algorithmus



Abbildung 4.3 Spiralförmige Anordnung der Röhrenblüten einer Sonnenblume [17]

Der Goldene Winkel

Ausgehend von dem Mittelpunkt der Sonnenblume ist jede nachfolgender Blüte um rund 137.5° um den Mittelpunkt rotiert. Dieser sogenannte *Goldener 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 wie folgt definiert:

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

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

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

Die Ergänzung zum Vollwinkel ist der Goldene Winkel:

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

Dieser hat die Eigenschaft, dass er aufaddiert nie den selben Winkel ergibt. Diese Eigenschaft machen sich viele Pflanzen zu Nutze und kann auch in CodeLeaves Anwendung finden.

Berechnungen für das Hinzufügen eines Blattes

Mit dem Vielfachen des goldenen Winkels ist der Winkel φ gegeben, um den eine neue Kante um die y-Achse rotiert wird. Für die Rotation und Skalierung einer Kante sind aber noch andere Größen zu berechnen.

Wird eine Kante mit der Länge l an einen Knoten angefügt, um die x-Achse mit dem Winkel θ geneigt und anschließend um die y-Achse mit den Winkel φ rotiert, entsteht ein Kugelkoordinatensystem mit der Position des Elternteils als Ursprung

4 Modellierung und Layout des Waldes

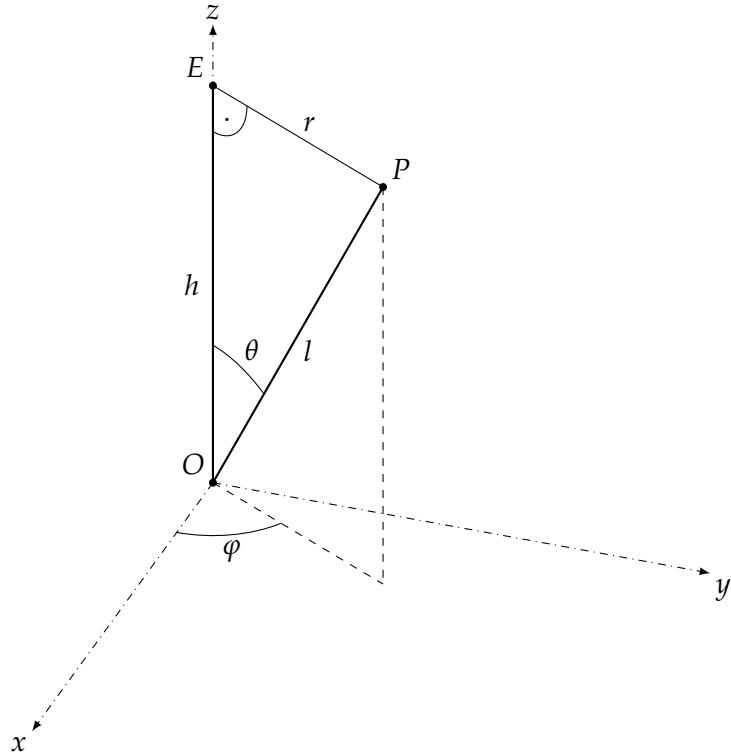


Abbildung 4.4 Kugelkoordinatensystem mit den zu berechnenden Größen für das Hinzufügen eines neuen Knotens

O , der y -Achse als Polachse, der Position des n -ten Kindknoten als Punkt P , dem *Polarwinkel* θ und dem *Azimutwinkel* φ [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.

Der Abstand d_n zwischen dem Punkt E und P berechnet sich für den n -ten Knoten nach [42] mit

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

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

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

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

Der Polwinkel θ lässt sich mithilfe von h und r berechnen.

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

Die Länge l der Kante kann mit

4.3 Verteilung innerer Knoten mit Circle-Packing

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

berechnet werden.

Für die Positionierung des Kindknotens muss der Punkt P von den Kugelkoordinaten (l, θ, φ) in das kartesische Koordinatensystem umgerechnet werden. Dies wird durch die folgende 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.

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

In Unity und C# ist dieser Algorithmus in vereinfachter Form in Listing 4.1 zu sehen ist.

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 kleinem c sichtbar.

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 Modellierung und Layout des Waldes

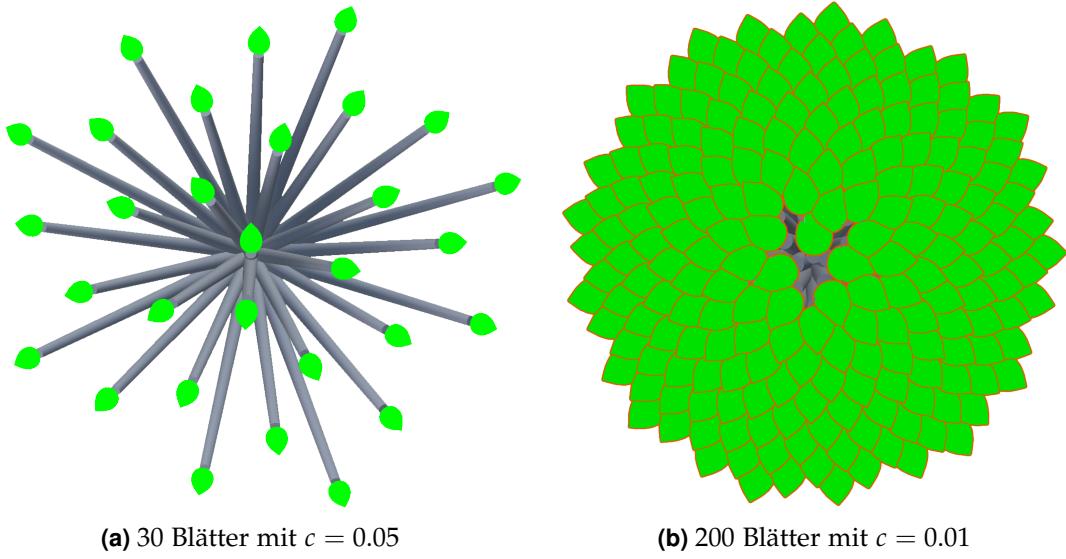


Abbildung 4.5 Sonnenblumen-Algorithmus zur Verteilung von Blättern

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 mit einer minimalen Höhe von 1, führen wir ein weiteres Merkmal ein.

Definition 4.1: Kreis und Radius eines inneren Knotens

Der Radius eines inneren Knotens ist der Radius des Kreises, aus dem alle Nachfahren des Knotens von oben betrachtet nicht hinaus ragen.

Bei einem inneren Knoten, dessen Kinder nach dem Sonnenblumen-Algorithmus verteilt wurden, kann dessen Radius leicht berechnet werden. Der Radius r des inneren Knoten ist gleich dem Abstand a_n des n -ten Kindes, der nach Formel 4.4 mit $a_n = c\sqrt{n}$ 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 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 Wang in [43] 2006 einen Algorithmus entworfen, der seitdem von vielen adaptiert worden ist. Zum Beispiel in der populären JavaScript Library *d3.js* zur Datenvisualisierung verwendet Bostock, der Schöpfer von *d3.js* diesen Algorithmus [4].

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 ein Hauptstamm für einen Baum 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.

Im Ganzen ist der leicht modifizierte Algorithmus für eine Menge von Kreisen $\{C_i \mid i = 1, 2 \dots n\}$ im Folgenden beschrieben:

Abgewandelte Form von Wang's Circle-Packing-Algorithmus (nach [43])

- (i) Falls $n >= 1$ setzte den Mittelpunkt von C_0 in den Ursprung und füge C_0 zur Front-Chain hinzu.
- (ii) Falls $n >= 2$ 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 , der Kreis dessen Mittelpunkt am nächsten zum Ursprung ist. C_n ist der Kreis in der Front-Chain nach C_m .
- (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 ist, 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 ist, 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).

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 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 natürlichem Wuchs am

4 Modellierung und Layout des Waldes

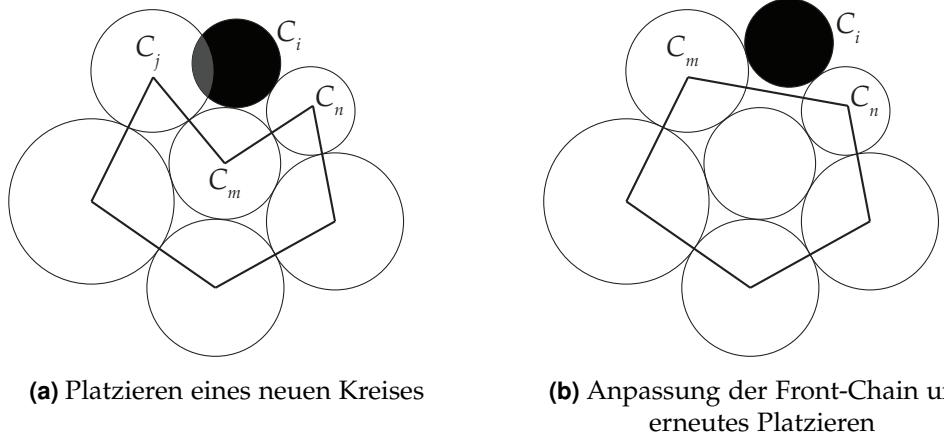


Abbildung 4.6 Wang's Circle Packing Algorithmus (nach [4])

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.

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 Polarwinkel α 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 Abbildung 4.7 abgebildet. Gesucht ist M_3 . Gegeben sind die Mittelpunkte M_1 und M_2 sowie die Radien der drei Kreise.

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

$$h = \sqrt{r_1^2 - |d_1|^2} = \sqrt{r_2^2 - |d_2|^2} \quad (4.11)$$

und D_1 lässt sich durch Umformen der Gleichung 4.11 mit

4.3 Verteilung innerer Knoten mit Circle-Packing

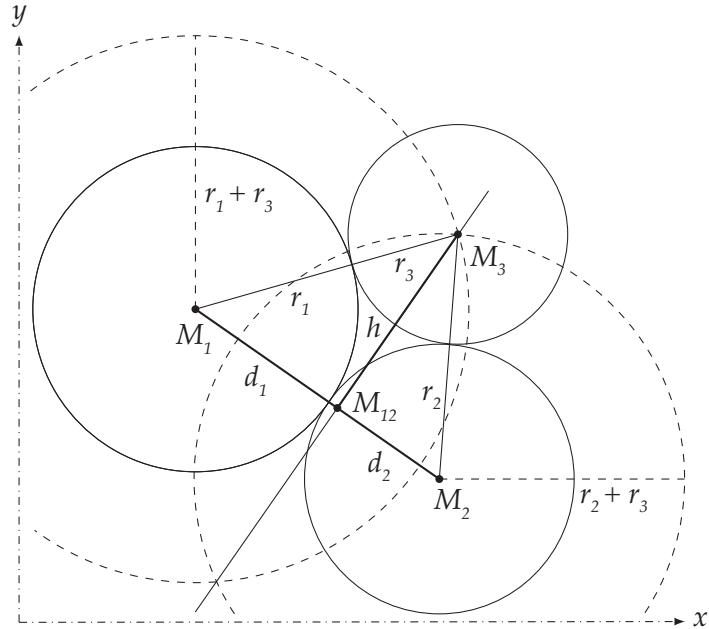


Abbildung 4.7 Berechnung der Position eines Kreises, tangent zu zwei anderen Kreisen

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

darstellen.

Mit diesen Gleichungen kann der Circle-Packung 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-Packung-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 Radius von K nötig, dass in Ebene $n - 1$ die Generation von K verteilt werden kann.

4 Modellierung und Layout des Waldes

Der Radius des Kreises von Knoten K (mit Mittelpunkt an der Position von K), der alle Kreise der Kinder von K umschließt und mindestens zu einem Kreis der Kinder tangential ist, ist der Abstand vom 0-ten Kind 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. Wir nennen den Kreis dieser Methode den *größten umschließenden Kreis*.

Für die vollständige hierarchische Anwendung des Circle-Packing wird für jeden Knoten rekursiv das Circle-Packing durchgeführt und anschließend der eigene Radius 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.

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

Verteilung von Bäumen

Die Verteilung der einzelnen Bäume kann durch das Circle-Packing sehr leicht umgesetzt werden. Nachdem der Kronenansatz einen Radius besitzt, ist damit auch die Radius des ganzen Baumes bekannt. Damit können die Bäume beliebig platziert werden, ohne dass sich deren Äste überschneiden würden. Es wäre zum Beispiel eine Verteilung in einem Gitter möglich, was dann an eine Baumschule mit Baumreihen erinnern würde. Es ist jedoch auch ganz einfach möglich die Bäume ebenfalls mit Circle-Packing anzuordnen. 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. Ein Auszug aus einem so entstandenen Wald 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.

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, deutlich.

Um das zu umgehen müsste anstelle des größten umschließenden Kreises, der minimal 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 CodeLeaves 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.

Rückführung auf Akzeptanzkriterien

In Abbildung 4.8 sind auch die unterschiedlichen Blattfarben zu sehen. Dargestellt wird die Testabdeckung im Projekt Air, womit das Akzeptanzkriterium 1.1 aus Abschnitt 2.4 erfüllt ist. Die Testabdeckung wird mit einem stufenlosen Farbverlauf von Rot (0%) über Gelb (50%) bis hin zu Grün (100%) dargestellt.

Hotspots sind als ganze Zweige und Bäume sehr schnell erkennbar, was Akzeptanzkriterium 1.2 erfüllt. Die roten Stellen, die in Abbildung 4.8 schon gut zu erkennen sind, stechen mit der Hololens sofort ins Auge.

4.4 Verwendete Algorithmen in der Praxis

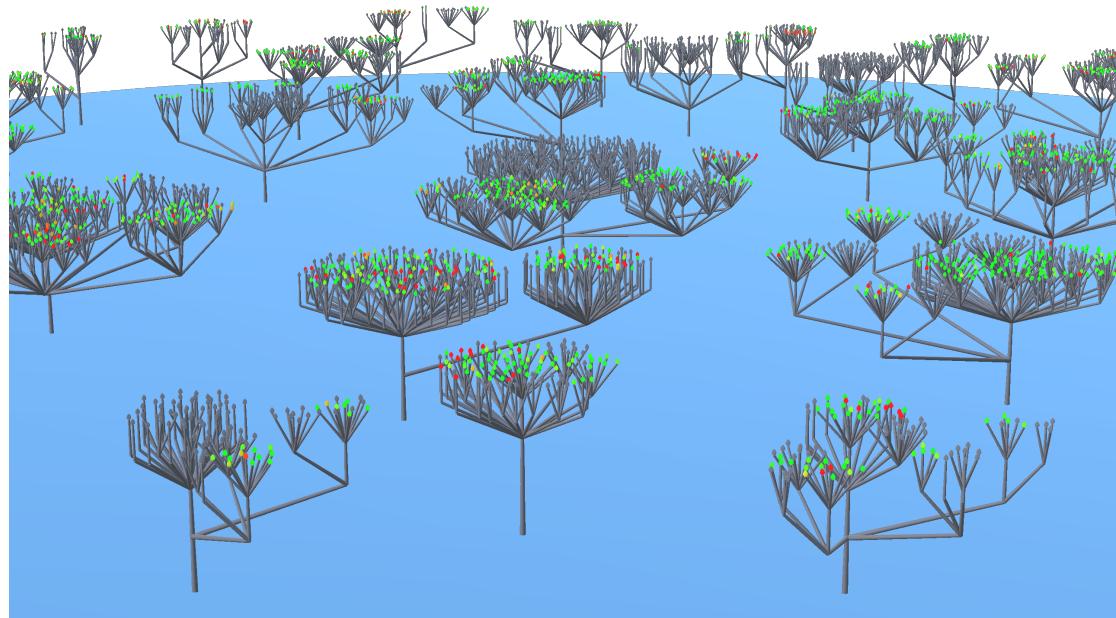
Für die Praxis in Unity ist bei der Verteilung der Bäume noch zu beachten, dass deren Radien erst verfügbar sind, nachdem der Circle-Packing-Algorithmus für alle Bäume vollständig durchlaufen wurde. Das heißt, dass bei der Generierung der einzelnen Bäume noch nicht bekannt ist, an welcher Position die Bäume letztendlich gesetzt werden müssen und zunächst zum Beispiel an der gleichen Stelle generiert werden müssen. Das bedeutet wiederum, dass bei asynchroner Programmierung 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.

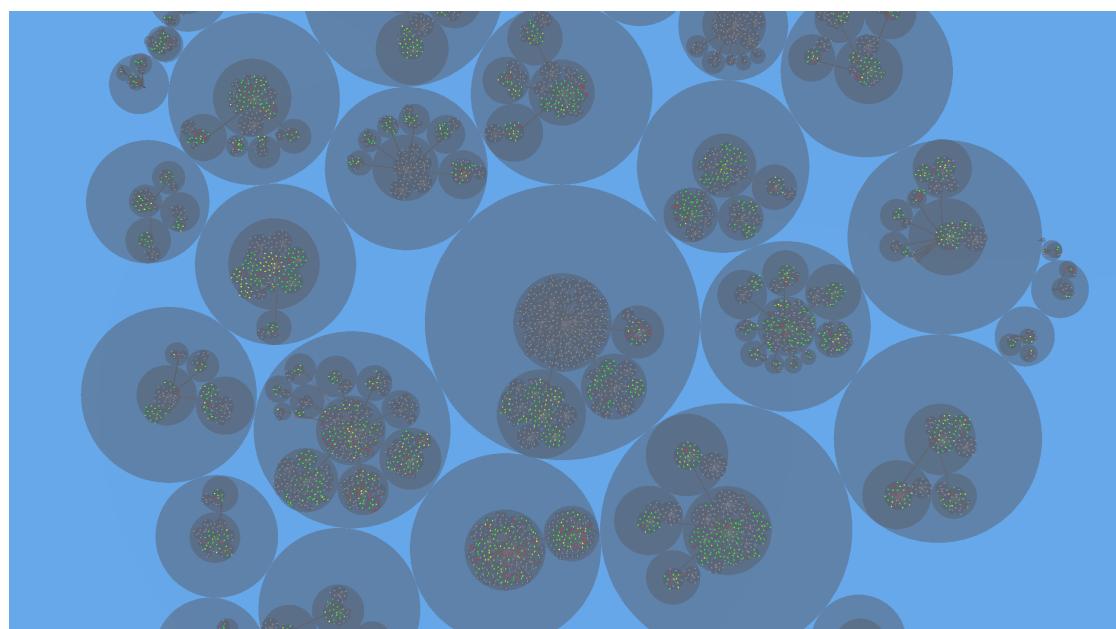
Soll vermieden werden, dass der Nutzer zunächst die überlagernde Generierung mit ansieht, dürften die Bäume erst sichtbar werden, nachdem alle Bäume fertig generiert und verteilt worden sind. Jedoch ist zu beachten, dass währenddessen dem Benutzer angezeigt wird, dass der Aufbau des Waldes im Hintergrund läuft. Mehr zu diesem Thema ist im nächsten Kapitel zu finden

Ein weiterer zu beachtender Punkt ist, dass bei beschriebener Vorgehensweise einzelne Blätter theoretisch auf die gleiche Position gelangen könnten. Grund dafür ist, dass der Radius eines Knotens mit ausschließlich Blätter gleich a_n gesetzt wurde. Daraus resultiert, dass bei zwei Geschwister solcher Knoten deren n -ten Kinder genau auf dem Schnittpunkt der Kreise liegen könnten. In diesem Fall würden sich die Blätter überschneiden. In der Praxis ist dies jedoch sehr unwahrscheinlich, besonders wenn für den Azimutwinkel des Sonnenblumen-Algorithmus ein zufälliger initialer Wert gesetzt wird, auf den dann ein Vielfaches des goldenen Winkel aufaddiert wird. In unserem Beispielprojekt mit 5373 Blättern, trat dieser Fall nicht auf.

4 Modellierung und Layout des Waldes



(a) Seitenansicht



(b) Vogelperspektive mit visualisierten Circle-Packing

Abbildung 4.8 Layout des Waldes

5 Grundlagen der Interaktion in der AR

5.1 Interaktionsmöglichkeiten 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 Computer nicht mehr weg zu denken.

Neben der zeigerbasierten Eingabe hat sich mit dem Einzug von Smartphones und Tablets die touchbasierte Eingabe etabliert. Diese kann für Monitor basierte AR verwendet werden. Bei dieser Art von Mit dem AR-Kit von Apple ab der iOS Version 11 sind solche Applikationen einfach zu realisieren.

Für die AR mit Head Mounted Displays (HMD) Devices, wie es die HoloLens ist und die wie der Name schon sagt, vom Nutzer getragen werden, ergeben sich neue Möglichkeiten mit Anwendungen zu interagieren. Auch wenn z.B. die HoloLens die Eingabe durch eine Bluetooth-Maus unterstützt, bieten sich in AR zusätzliche Eingabemöglichkeiten an.

Es ergeben sich folgende Eingabemöglichkeiten für HMD-Devices:

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

Blickrichtung

Bei der HoloLens ist die *Blickrichtung* (engl.: *Gaze*) ein elementarer Bestandteil der Bedienung der HoloLens [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. Dieses Ziel der Blickrichtung nennen wir *fokussiertes Objekt*.

Gesten

Nachdem bei HDM-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 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 Ausgabe. 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 der Hände verwendet werden. Mit Produkten wie zum Beispiel Kinetic¹ können solche Gesten auch erkannt werden. Bei der HoloLens sind die unterstützen Gesten jedoch stark begrenzt. Es existieren insgesamt drei verschiedene Gesten, die von der HoloLens als solche identifiziert werden können. Die sogenannte *Bloom*-Geste ruft das Windows Menü auf. 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.

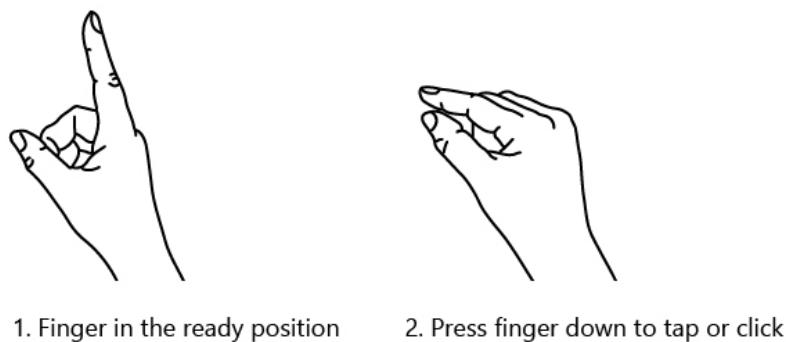


Abbildung 5.1 Air-tap [5]

Mit der *Manipulation* Geste stellt Microsoft eine kontinuierliche Geste zur Verfügung. Wird ein Air-tap gehalten und die Hand anschließend bewegt, so kann als Ausgabe 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 eine Interaktion mit der Verwendung von beiden Händen möglich ist.

¹<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 Sprachbefehle 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 an, die Spracherkennung zur Unterstützung der Gestensteuerung zu verwenden. 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 kein Controller. Bei anderen Windows Mixed Reality Geräten wie die *Immersive* Headsets von Acer und HP² kann mit den *Motion* Controllern, ähnlich wie bei Gaming Konsolen, mit verschiedenen Tasten unterschiedliche Eingaben tätigen. Bei der HoloLens steht lediglich der *Clicker* zur Verfügung, ein kleines Bluetooth-Gerät, dass durch physisches Klicken den Air-tap ersetzen kann. Bei Dauerhafter Eingabe von Air-tap-Gesten ist der Einsatz vom Clicker angenehm, da man 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 der Senior Holographic Designer 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 es *Tag-along*-Objekte zu verwenden. Diese Objekte bleiben immer im Blickfeld des Betrachters. Schaut der Nutzer in eine andere Richtung, wird das Objekt neu positioniert, sodass es wieder im Blickfeld gelangt. Innerhalb des Blickfelds bleiben Tag-along-Objekte aber stationär, sodass der Nutzer mit der

²https://developer.microsoft.com/en-us/windows/mixed-reality/immersive_headset_hardware_details

5 Grundlagen der Interaktion in der AR

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

In der Case Study aus [20] wurde festgestellt, dass die Probanden sich von solchen Tag-along-Objekten in vielen Fällen bedrängt fühlen und instinktiv versuchen davon weg zu kommen.

Eine andere Möglichkeit ist es die Hologramme stationär im Raum zu platzieren und die Aufmerksamkeit des Nutzers gezielt zu lenken. Das kann mit verschiedenen Techniken erreicht werden. Eine davon ist die Verwendung von *Spacial 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 Content wie CodeLeaves, sind Geräusche nicht besonders intuitiv.

Es kann auch mit zusätzlichen Objekten gearbeitet werden, die dem Nutzer darauf hinweisen, wo sich etwas befindet, was seine Aufmerksamkeit erfordert. Solche Hilfs-Objekte nennen wir *Direction Indicators*. 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 das Menu zu Interaktion mit dem Wald der „Move“-Button von einem Stamm verdeckt.



Abbildung 5.2 Verdeckung eines Buttons

Analog zum 2D Popups ist es natürlich 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. Die UI Elemente würden dadurch vom eigentlichen interagierten Objekt losgelöst werden, was eine wenig

5.3 Technische Umsetzung von Interaktionen mit Reaktiver UI

intuitive Interaktion darstellt. Dies kann aber zu Folge haben, dass der Nutzer irritiert wird, weil er mit etwas weiter weg interagiert und die UI deutlich weiter vorne erscheinen kann.

Eine Abhilfe kann ein *Shine-through*-Effekt schaffen, wie er bei HoloStudio Verwendung findet. Dabei bleiben UI Elemente an dem Hologramm angeheftet, mit dem interagiert wird, jedoch scheinen sie durch davor liegende Hologramme durch.

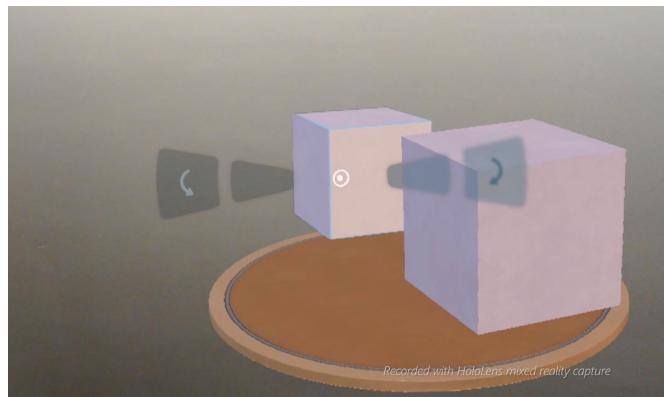


Abbildung 5.3 Shine-through-Effect von UI Elementen in HoloStudio [20]

Diese Methode funktioniert jedoch schlecht, wenn die UI Elemente Texte sind und von davor liegenden Texten verdeckt werden. Dadurch werden beide Texte unleserlich. Dieses Problem tritt z.B. zwangsläufig bei der Beschriftung von Blättern in CodeLeaves auf und lässt sich nur vermeiden, indem einer der beiden Texte verschwindet oder zumindest verblasst.

Die betrachteten Herausforderungen von 3D Interaktion und deren mögliche Lösungsansätze finden sich im Interaktionskonzept von CodeLeaves wieder und werden in Abschnitt ?? aufgegriffen.

5.3 Technische Umsetzung von Interaktionen mit Reaktiver UI

Motivation

Eine der offensichtlichen Interaktionen mit CodeLeaves ist das Anklicken von Knoten, seien es Blätter oder die dazugehörigen Kanten. Der Nutzer erwartet damit nähere Informationen zu einem Paket oder einer Klasse zu erfahren. Betrachten wir zunächst eine sehr einfache 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 der Klassename gespeichert werden kann. Das Anzeigen Namens ist demnach kein Problem und würde wie folgt funktionieren:

5 Grundlagen der Interaktion in der AR

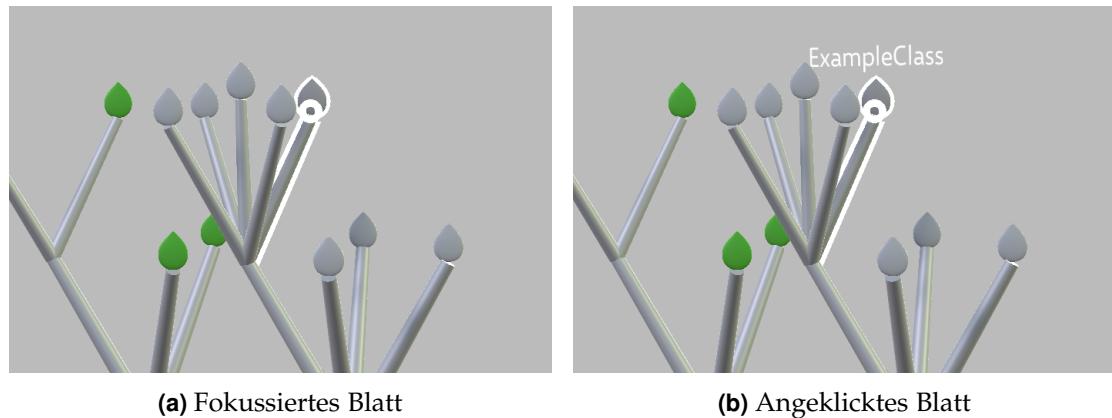


Abbildung 5.4 Anzeige des Klassennamens durch Klick auf ein Blatt

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

```

Im traditionellen 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 Businesslogik aber komplexer, kommt dieser Ansatz schnell an seine Grenzen.

Eventuell könnte sich zum Beispiel folgende Einstellung in der Applikation als sinnvoll erweisen:

Der Nutzer kann einstellen, ob bei einem Klick auf ein Blatt der Klassenname oder die Zahl der gerade visualisierten Metrik angezeigt wird.

Woher weiß die Funktion jetzt was angezeigt werden soll?

Das eigentliche Problem bei der beschriebenen Situation ist, dass Businesslogik und UI eng gekoppelt wurde. Dies sollte tunlichst vermieden werden. Bei einem Klick auf einem Blatt sollte nicht die UI entscheiden was zu tun ist, sondern die Businesslogik. Die Entkopplung von UI und Logik, ist mit reaktiver Programmierung möglich.

5.3 Technische Umsetzung von Interaktionen mit Reaktiver UI

Was ist reaktive Programmierung? Stoltz beschreibt es in [40] als „Programmierung mit asynchronen Datenströmen.“ Dies trifft den Kern. Es geht darum auf Änderungen von Werten zu reagieren, die durch *Streams* propagierte 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 den Text, der im Label eines Blattes steht.



Abbildung 5.5 Datenstrom des Labels

Über die Zeit hinweg, kann sich durch die Businesslogik der darzustellende Text verändern. Im Beispiel soll der Klassenname, dann vielleicht Laufzeitfehler, Coverage und dann wieder der Klassenname 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*³ 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 UI-Knotens 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 {
```

³<https://github.com/neuecc/UniRx>

5 Grundlagen der Interaktion in der AR

```
8     ...
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 nicht mehr direkt auf die UI zugegriffen, sondern es wird ein Funktion aufgerufen, die sich darum kümmert, dass das UI-Model editiert.

6 Interaktion mit CodeLeaves

CodeLeaves kann ohne Interaktion die Struktur einer Software, eine Metrik und Verbindungen zwischen Softwareartefakten darstellen. Damit kann bereits einen Überblick über die Software geschaffen werden. Der wirklichen Mehrwert von CodeLeaves wird aber erst erreicht, wenn mit dem Software-Wald interagiert wird. Die intuitive Bedienung von CodeLeaves ist demnach essentiell für eine gute User Experience und damit 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 Interaction Design 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. Dennoch werden

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 (siehe Akzeptanzkriterium 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. Wenig hilfreich wäre es aber wenn er über dem Benutzer schwebt.

Das Skalieren ist auch unerlässlich. Gerade bei großen Systemen kann auch der Wald sehr groß werden. Daher muss die Größe des Waldes auf die Gegebenheiten der Räumlichkeiten angepasst werden können.

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 selbst ist bequemer.

6 Interaktion mit CodeLeaves

Bounding Box

Das Repository „Mixed Reality Design Labs“ bietet für genau diese Interaktionen eine Lösung. Die *Bounding Box*, wie sie in Abbildung 6.1 zu sehen ist, rendert ein 3D Rahmen um das zu manipulierende Objekt. Das Platzieren des Objekts kann mit einer Manipulation auf den gesamten Bereich der Bounding Box erreicht werden. Mit den kleinen Würfeln an den Ecken der Bounding Box kann skaliert werden und die kleine Kugeln auf den vertikalen Kanten der Bounding Box ist für das Rotieren vorgesehen.

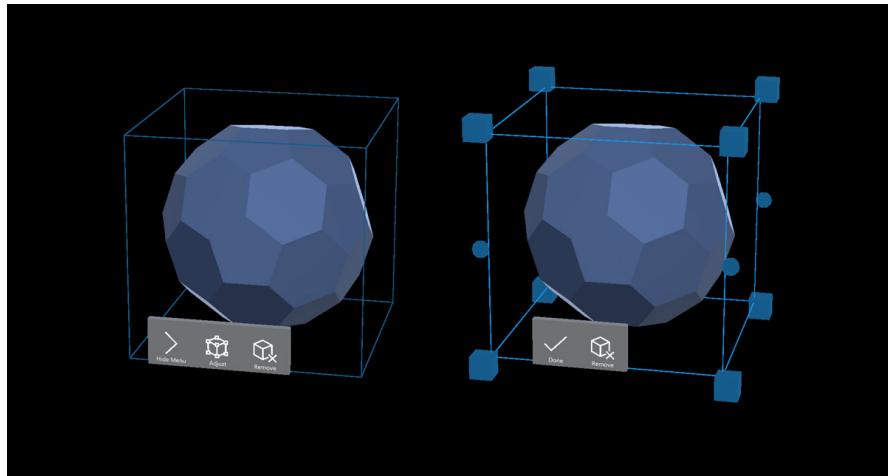


Abbildung 6.1 Bounding Box zum Platzieren, Skalieren und Rotieren von Objekten [10]

Dieser Ansatz hat den Vorteil, dass dem Nutzer ein Interface angeboten wird, das ihm aus dem Zweidimensionalen wohl bekannt ist. In den meisten Applikationen mitunter in den Bereichen 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 ein 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 repräsentiert. Der Waldboden als Antwort auf diese Frage leuchtet ein. Er ist auch bei näherer Betrachtung einzelner Teile von CodeLeaves 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 mit dem gleichen Ziel 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 und fokussiertem Waldboden den Wald entsprechend manipulieren. Die Ausführung dieser Geste werden näher im Abschnitt 6.3 ausgeführt.

Interaktion mit Spacial Mapping

Für den Plazierungs-Modus bietet sich für die HoloLens die Nutzung des *Spacial Mapping* an. Die HoloLens ist fähig 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 ist es zum Beispiel möglich den Wald auf einen Tisch oder auf den Boden zu positionieren.

Bei aktivem Platzieren des Waldes wird des Spatial Mapping Mesh angezeigt, sodass der Nutzer sieht, wo genau der Waldboden mit der Umgebung kollidiert. Die Positionierung 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 eines Deltas geeignet. Für den Input 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, da die Hand des Nutzers nicht dauerhaft oben gehalten werden muss, wie es bei der Manipulation nötig wäre.

Der Modus des Platzieren 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 aber 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 User Interfaces ist dies mit einem Rechtsklick zu vergleichen. In 3D kann um den Punkt, den der Nutzer beim Aufruf des Kontextmenüs fokussiert hatte, das Kontextmenü als 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 vom Cursor gleich weit entfernt und daher schnellstmöglich mithilfe der Blickrichtung erreichbar.

6 Interaktion mit CodeLeaves

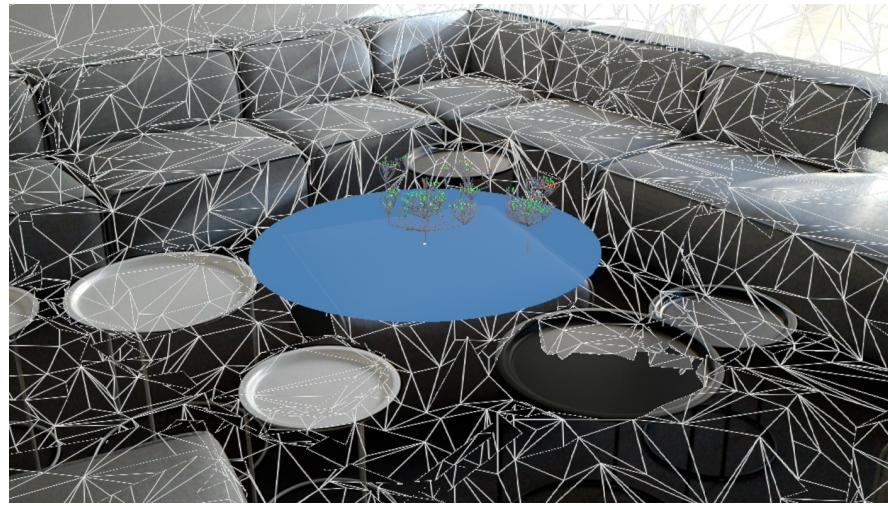


Abbildung 6.2 Spacial Mapping beim Platzieren des Waldes

Die Distanz des Kontextmenüs ist den Umständen entsprechend anzupassen. Ist der fokussierte Punkt bereits näher als die Distanz, in der eine Darstellung von Interaktionselementen empfohlen wird¹, sollte das Kontextmenü nicht noch näher an dem 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.

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 Prototypen ist in Abbildung 6.3 zu sehen. Mit dem vierten Button von links ist das App-Menü aufrufbar, dass in ?? vorgestellt wird.

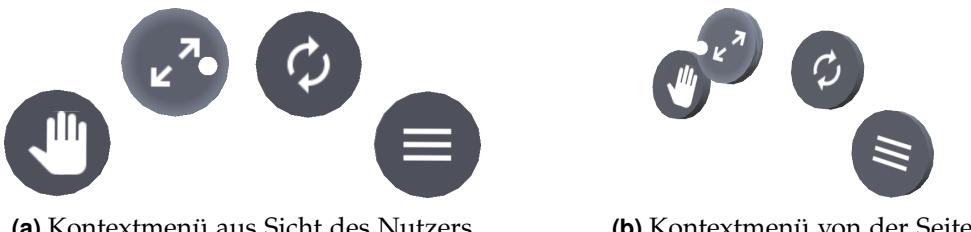


Abbildung 6.3 Kontextmenü des Waldbodens im HoloLens Prototyp

In Abbildung 6.3b ist zu erkennen, dass der fokussierte Button räumlich hervorgehoben ist. Dies ist eine vor der Augmented Reality noch nicht dagewesene Möglichkeit

¹Die Distanz, in der es empfohlen ist Hologramme zu platzieren, liegt bei 2,0 Metern [6]

6.3 Manipulation-indicators

dem Nutzer 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 wird einem Button ein haptisches Feedback verliehen und der Nutzer kann ihn räumlich „eindrücken“. Das Prinzip findet auch beim Windows-Menü der HoloLens Anwendung und zieht sich durch die gesamte UI des Prototypen.

6.3 Manipulation-indicators

Mit dem Kontext-Menü des Waldbodens kann zwischen den beiden Interaktionsmodi umgeschalten werden, die mit der Manipulation-Geste ausgeführt werden. Bei der Manipulation-Geste weiß der Nutzer aber ohne weiter Hilfe nicht, wie er seine Hand bewegen muss, um die gewünschte Manipulation zu erreichen. Möchte der Nutzer beispielsweise den Wald verkleinern, weiß er nicht in welche Richtung der Wald vergrößert wird und in welche verkleinert. Zudem sind mehrere Bewegungsachsen wie horizontal, vertikal oder diagonal denkbar. Deswegen ist es unerlässlich dem Nutzer Feedback zu geben. Dies wird mit den – wie wir sie nennen – *Manipulation-indicators* erreicht. Diese tauchen auf, sobald der Air-tap gehalten wird und indizieren die möglichen Richtungen und deren Auswirkung.

In Abbildung 6.4 sind Manipulation-indicators für das Beispiel des Skalieren dargestellt.

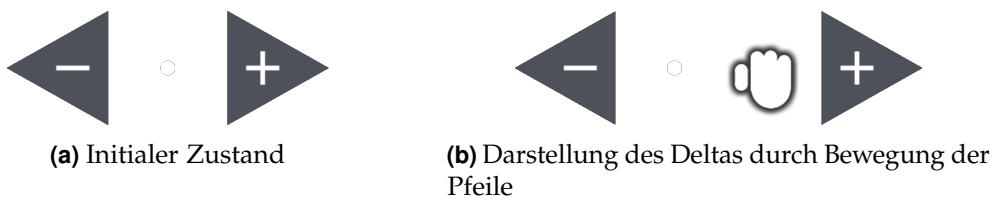


Abbildung 6.4 Manipulation-indicators am Beispiel des Interaktions-Modus Skalieren

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 Soft-

6 Interaktion mit CodeLeaves

ware er überhaupt visualisieren möchte. Auch müssen die dargestellten Informationen nach der Auswahl eines Projekts konfiguriert werden können. Dafür wird ein zentrales *App-Menü* eingeführt.

Für dieses Menü wurde sich an herkömmlichen Fenstern orientiert, 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 auszuwählen. Deshalb sollte das Menü nach dem Start der App in jedem Fall im Fokus des Nutzers platziert werden, sodass der Nutzer 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 würde.

Nach der Auswahl eines Projekts muss der Tag-along-Effekt aber 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 zu werden oder ganz geschlossen zu werden.

Zweiteres kann mit einem Schließen-Button in einer Titelleiste realisiert werden, was dem Nutzer aus bekannten Systemen vertraut ist. Das erneute Öffnen kann und muss durch das Kontextmenü 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.

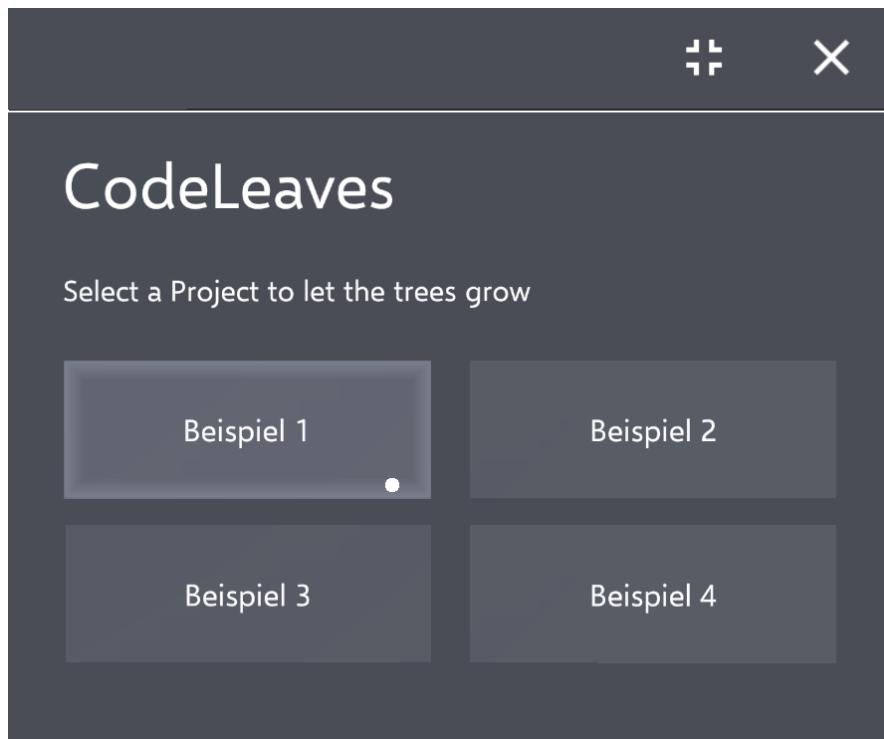
Die Funktion des stationären Platzierens kann ebenfalls in der Titelleiste untergebracht werden. Analog zum „Vollbild“ eines Desktop-Fensters, bei dem der Nutzer sich ausschließlich auf dieses konzentrieren möchte, wird neben dem Schließen-Button ein neuer *Tag-along-Button* eingeführt, der für das Ein- und Ausschalten des Tag-along-Effekts zuständig ist. Damit kann der Nutzer das Fenster jederzeit dort platzieren, wo es sich gerade befindet. Wurde das App-Menü zuvor geschlossen, wird beim erneuten Öffnen des Fensters durch das Kontextmenü der Tag-along-Effekt aktiviert, sodass der Nutzer das Fenster nicht suchen muss, sondern 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 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 und Faktor, kann der Drag-and-drop im 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

Auswahl eines Projekts beschränkt sich dabei auf Beispielprojekte (siehe Akzeptanzkriterium 7.1).

Projektauswahl



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



(b) Titelleiste mit deaktivierem Tag-along

Abbildung 6.5 High-Fi Prototyp des App-Menüs

Für die produktive Einsetzung von CodeLeaves ist ein Import von neuen Projekten nötig (siehe Akzeptanzkriterium 7.2). In dem Low-Fi Papier Prototyp in Abbildung 6.6 ist die Interaktion mit zusätzlicher Import-Möglichkeit dargestellt. Bei den hier vorgestellten Low-Fi Prototyp Darstellungen besteht kein Anspruch auf Vollständigkeit. Vielmehr müsste der Low-Fi Prototyp eine Evaluations-Phase mit Probanden durchlaufen, um diesen zu bewerten und bei Bedarf zu überarbeiten. Auf diese Phase wird im Rahmen dieser Arbeit jedoch verzichtet, da es vor allem um das Design neuer Konzepte in der AR und das Grobkonzept gehen soll.

Bei der Projektauswahl hat der Nutzer die Wahl zwischen zwei Haupt-Tasks: Der Auswahl eines bereits betrachteten Projekten und den Import von neuen Projekten. Die

6 Interaktion mit CodeLeaves

Interaktion bei einem Import ist für SonarQube beispielhaft im Anhang zu finden.

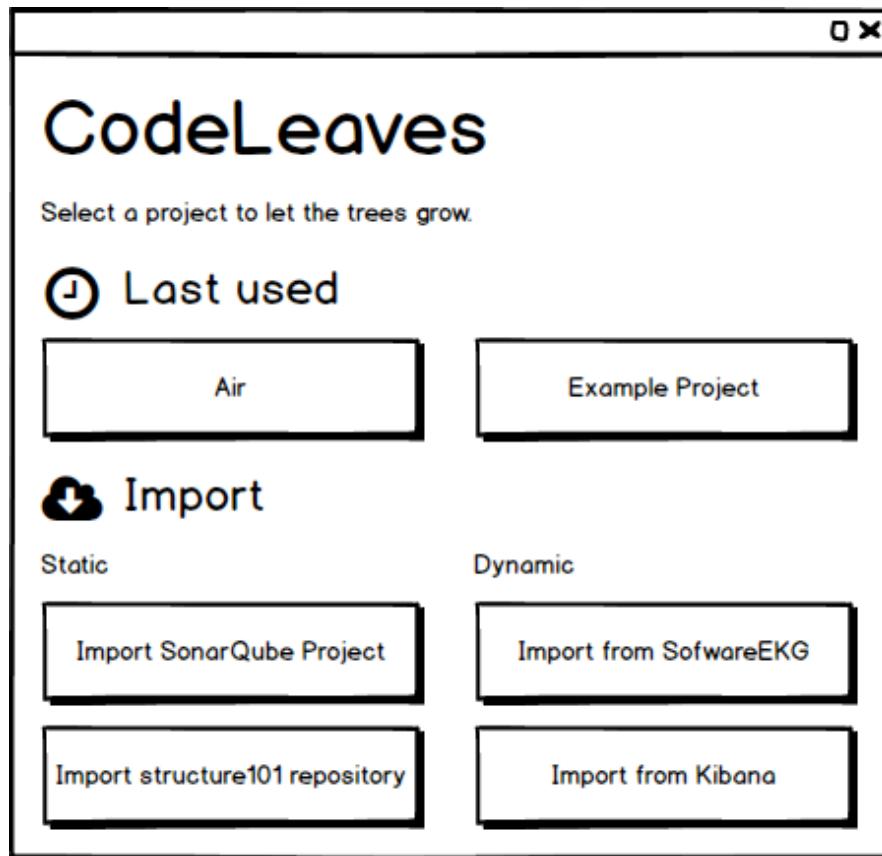


Abbildung 6.6 Low-Fi Prototyp der Projektauswahl im App-Menü

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.

Einstellungen

Dies kann er ebenfalls mit dem App-Menu 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 (siehe Akzeptanzkriterium 1.1 und 4.1) und ein Verbindungs-Typ für die Dicke der Wurzeln und Kanten (siehe Akzeptanzkriterium 2.4 und 4.5) 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

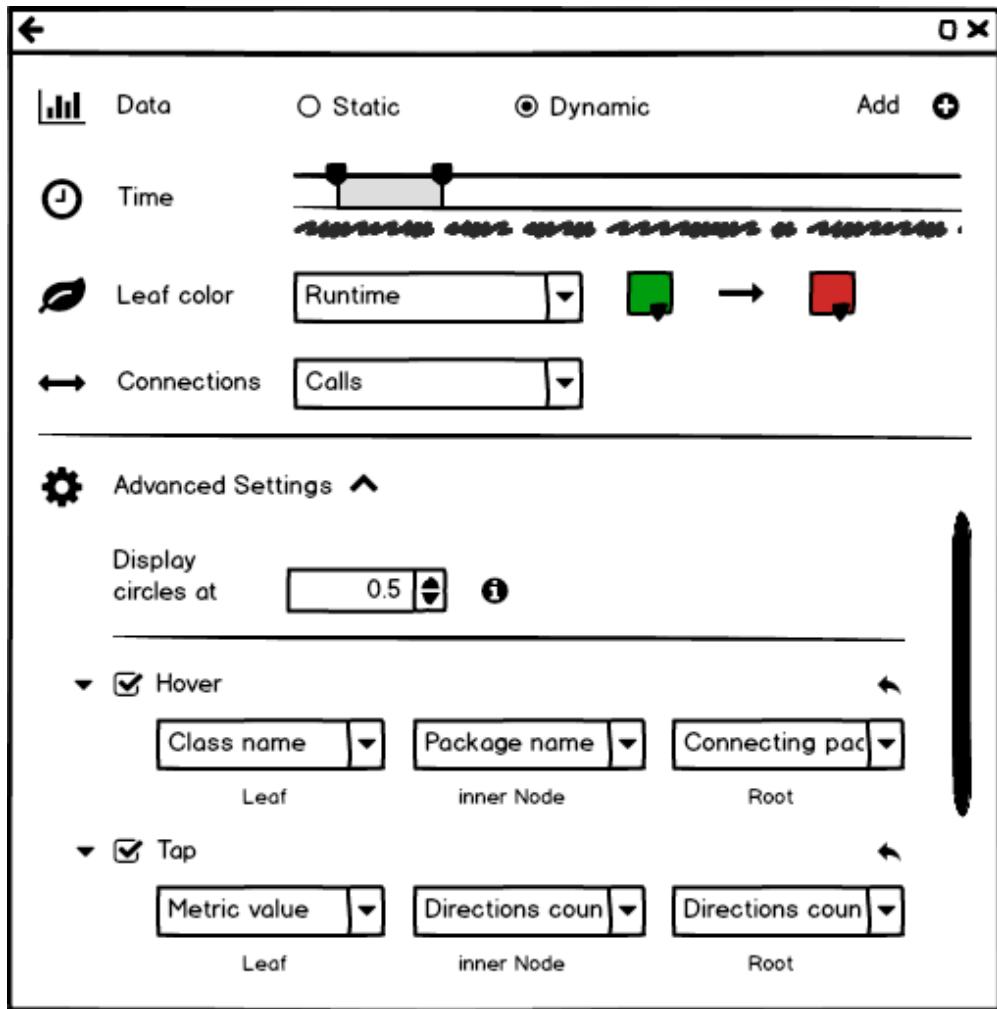


Abbildung 6.7 Low-Fi Prototyp für die Einstellungen im App-Menü

dynamische oder statische Daten betrachten möchte. Dynamische Daten beziehen sich immer auf einen bestimmten Zeitraum, wohingegen statische Informationen den Zustand eines Systems zu einem bestimmten Zeitpunkt widerspiegeln.

Deshalb wird als erstes die Wahl angeboten, ob statische oder dynamische Informationen betrachtet werden sollen. Mit dem „Add“ neben dieser Auswahl kann der Nutzer Daten zu dem geladenen Projekt hinzufügen (siehe Akzeptanzkriterium 7.3). Je nach dem welche Auswahl getroffen ist, kann der Nutzer darunter entweder einen Zeitpunkt – in diesem Fall wird nur **ein** Schieberegler angezeigt – oder einen Zeitraum angeben (siehe Akzeptanzkriterium 2.7 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 die aktuell dargestellten Daten interaktiv erforschen.

Norman stellt im Interaction Design Klassiker *The Design of Everyday Things* sieben allgemeingültige Prinzipien für gutes Interaction Design auf. Nach dem siebten Prinzip

6 Interaktion mit CodeLeaves

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. Dort hat der fortgeschrittene Nutzer zum einen die Möglichkeit die Aktionen, die in Abschnitt 6.5 und insbesondere in Tabelle 6.1 vorgestellt werden, anzupassen.

Zum anderen ist in den „Advanced Settings“ die Einstellung „Display circles at“ zu finden, die durch das Prototyping des High-Fi Prototyp 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 ist, desto schwieriger wird die Interaktion mit einzelnen Blättern oder Kanten.

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

Abbildung 6.8 zeigt die Fokussierung eines Knotens mithilfe der Interaktion mit seinem Kreis. Auf die Fokus-Aktion wird im nächsten Abchnitt eingegangen.

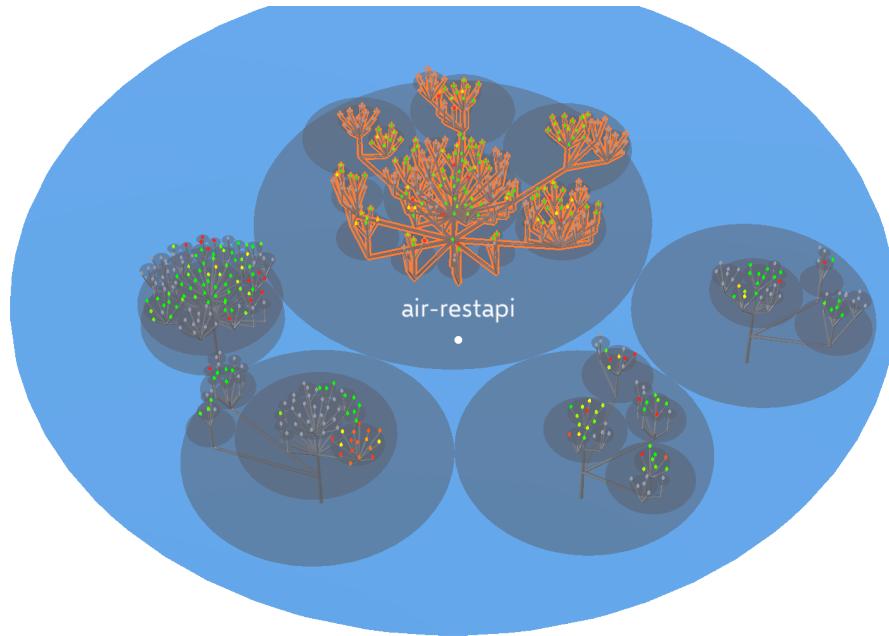


Abbildung 6.8 Visualisierung der Kreise für leichtere Interaktion

6.5 Interaktion mit Blättern, inneren Knoten und Wurzeln

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

Cooper ist der Meinung, dass “constant [...] feedback is exactly what users need” [9]. 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 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 Knoten und Wurzeln nur kurz aufflackern. Bei der Entwicklung des High-Fi Prototyps konnte festgestellt werden, dass diese Art der Hervorhebung zu schnell und damit unangenehm ist.

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

6 Interaktion mit CodeLeaves

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 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 und 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

In der Tabelle 6.1 können alle verfügbaren Aktionen betrachtet werden. Das Icons in der ersten Spalte befinden sich zur Wiedererkennung zum Teil auch in den nachfolgenden Abbildungen des Low-Fi Prototypen. 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.

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	Fokus	Fokus
2		Kontextmenü aufrufen	Tap-and-hold	Tap-and-hold
3		Wert der aktuellen Metrik anzeigen	Tap	
4		Liste mit allen Metriken anzeigen	KM	
5		Anzeigen wie vielen Verbindungen in welche Richtung gehen		Tap
6		Zugeordnete Verbindungen hervorheben	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		KM

Die Zuordnung der Aktionen zu den vorhanden Eingabemöglichkeiten kann, wie in Abschnitt 6.4, Abbildung ?? zu sehen, bei Bedarf vom Nutzer angepasst werden. Dies ist sinnvoll, wenn der Nutzer mit der einer anderen Zuweisung schneller oder komfortabler an gesuchte Informationen gelangt.

Möchte der Nutzer zum Beispiel nur Klassen mit niedriger Coverage identifizieren, ist es sinnvoll wie in Tabelle 6.1 angegeben bei einem Air-tap auf ein Blatt den Wert der Coverage anzuseigen. Ist der Nutzer jedoch ausschließlich an statischen Abhängigkeiten interessiert, kann er anstatt den 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 um zu schauen. 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 immer zwischen unterschiedlichen

6 Interaktion mit CodeLeaves

Raumtiefen hin- und herschauen 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 sie dem Nutzer durch Touch-basierten Eingabegeräte bereits vertraut ist.

● 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.

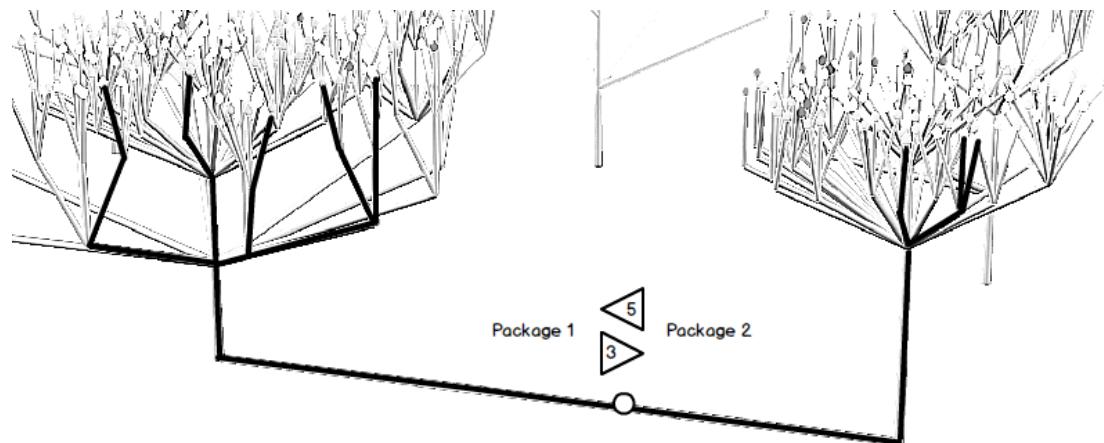


Abbildung 6.9 Darstellung der Anzahl, Richtung und Hervorhebung der aggregierten Verbindungen einer Wurzel

● 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 Package 1 nach Package 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 wieder finden 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 A.2 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.

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

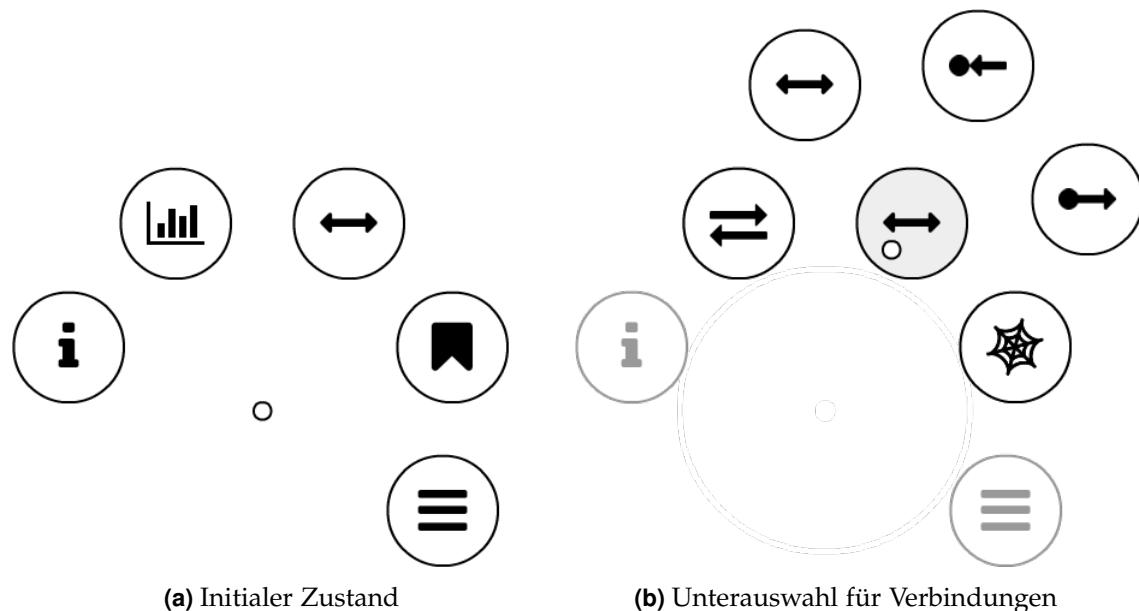


Abbildung 6.10 Kontextmenü für ein Blatt

7 Zusammenfassung und Ausblick

7.1 Zusammenfassung

Zu Beginn haben wir uns dem Thema AR genähert und evaluiert, wieso AR besonders gut für 3D Softwarevisualisierung geeignet ist. Auf dieser Basis wurde im Kapitel 2 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 werden zu Blättern.

Auf das Laubdach lassen sich ausgezeichnete 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 Stamm- und Ast-Dicke der Bäume. Je mehr aggregierte Verbindungen durch ein Stamm oder Ast fließen, desto dicker wird dieser. Durch diese Darstellung kann schon ohne jegliche Interaktion eine gute Übersicht über die Kopplung einer Software geschaffen werden. Durch Wurzeln können die aggregierten Verbindungen auch zwischen einzelnen Bäumen dargestellt werden.

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

In Kapitel 3 werden Datenmodelle entwickelt, die einerseits Schichtentrennung zwischen Backend, Business Logik und Frontend ermöglichen 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 wird in 4 beschrieben. Es wird gezeigt wie sich Verhalten in der Natur mit Fibonacci-Zahlen und dem Goldenen Schnitt auf die Bäume von CodeLeaves übertragen lassen. Mit dem Sonnenblumen-Algorithmus werden Blätter eines Elternteils gleichmäßig auf einer Kreisfläche verteilt, sodass sie den gleichen Abstand zueinander haben.

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

7 Zusammenfassung und Ausblick

sichtbar sind.

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 Interaktionen in Unity sinnvoll umgesetzt werden.

Zu guter Letzt wurden die im High-Fi Prototypen bereits umgesetzten Interaktionen in Kapitel 6 aufgegriffen und durch ein Low-Fi Prototypen ergänzt. Mit dem universellen Kontextmenü und den Manipulations-Indikatoren wurden Interaktions-Elemente eingeführt, die auch in anderen Anwendungen ausgezeichnet 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 Schritte zu gehen.

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 weiter Import-Möglichkeiten anzubieten. Für statische Abhängigkeiten bietet sich *strucute101*¹ an. Mit structure101 sind Abhängigkeiten auf Klassenebene verfügbar. Der Export der Daten von structure101 wurde bereits erfolgreich getestet und muss als nächsten 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 [44]. 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.

Umsetzung der Aggregation Liegen ausreichend Daten für die aggregierte Verbindungen in Form des Software Meta-Modells vor, müssen die Anzahl der Einzelverbindungen auf die Dicke der Kanten und Wurzeln übertragen werden. Die Erweiterung des Prototypen für Unterstützung dieser Funktionalität ist gering, da der Prototyp bereits unterschiedliche Dicken von Kanten und Wurzeln generieren kann.

Evaluierung des Interaktionskonzepts Das Interaktionskonzept, das im Rahmen dieser Arbeit erarbeitet wurde, muss mit Test-Nutzern evaluiert werden. In diesem Zuge kann

¹Structure101 ist eine agile Architektur-Entwicklungs-Umgebung (ADE), mit der das Software-Entwicklungsteam eine Codebasis organisieren kann (aus dem Englischen aus [27]).

7.2 Ausblick

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 Interaktionskonzept 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.

Nutzung für mehrere Nutzer Die Möglichkeit holographische Inhalte gleichzeitig mit anderen Nutzern teilen zu können, ist einer der Vorteile der AR. Diese muss jedoch auch in CodeLeaves 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 NASA, um das Rover-Planungstool der NASA durch die Nutzung von HoloLens Geräten zu erweitern. Die Apps laufen auf mehreren HoloLens Geräten gleichzeitig und kommunizieren und synchronisieren sich in Echtzeit, sodass Nutzer gemeinsam an einer Aufgabe arbeiten können [11].

Das Teilen erfolgt so, dass die App, die als erstes startet, einen Welt-Ankerpunkt definiert. Der Nutzer muss eine Session öffnen, der andere Nutzer beitreten können. Die Verbindung kann mit visuellem Pairing durch QR-Codes erfolgen. Nach dem Beitreten einer Session wird der bestehende Welt-Ankerpunkt geladen und so die Position von geteilten Hologrammen synchronisiert.

Der Zustand einer App, die Sharing unterstützt, kann ebenfalls geteilt werden. Dies wäre für CodeLeaves besonders wichtig, da nur so zum Beispiel Gemeinsam Verbindungen genauer untersucht werden können.

Für die Synchronisation des Zustandes müssen spezielle Datentypen verwendet werden.

Der Einbau von Sharing wäre eine überaus spannende weiterführende Aufgabe, die CodeLeaves noch attraktiver machen würde.

A Anhang

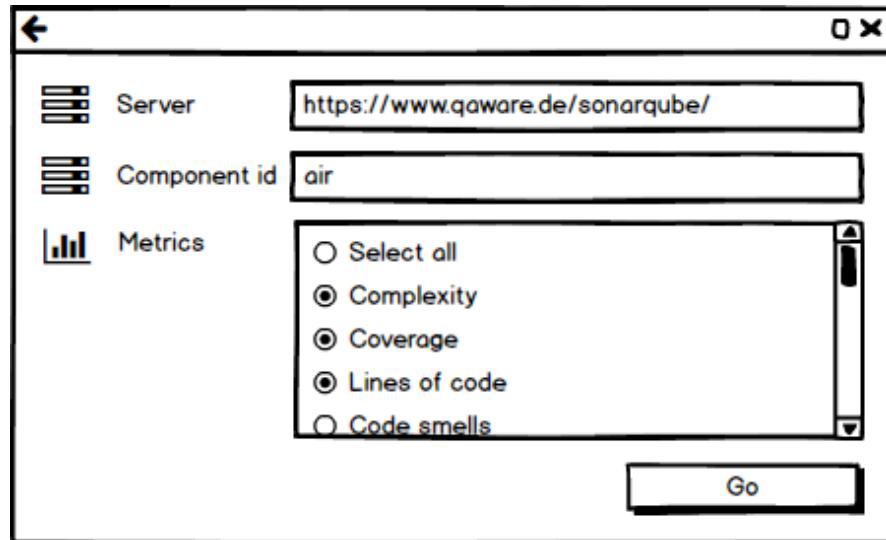


Abbildung A.1 Interaktion bei dem Import eines neuen Projekts anhand des Beispiels SonarQube

A Anhang

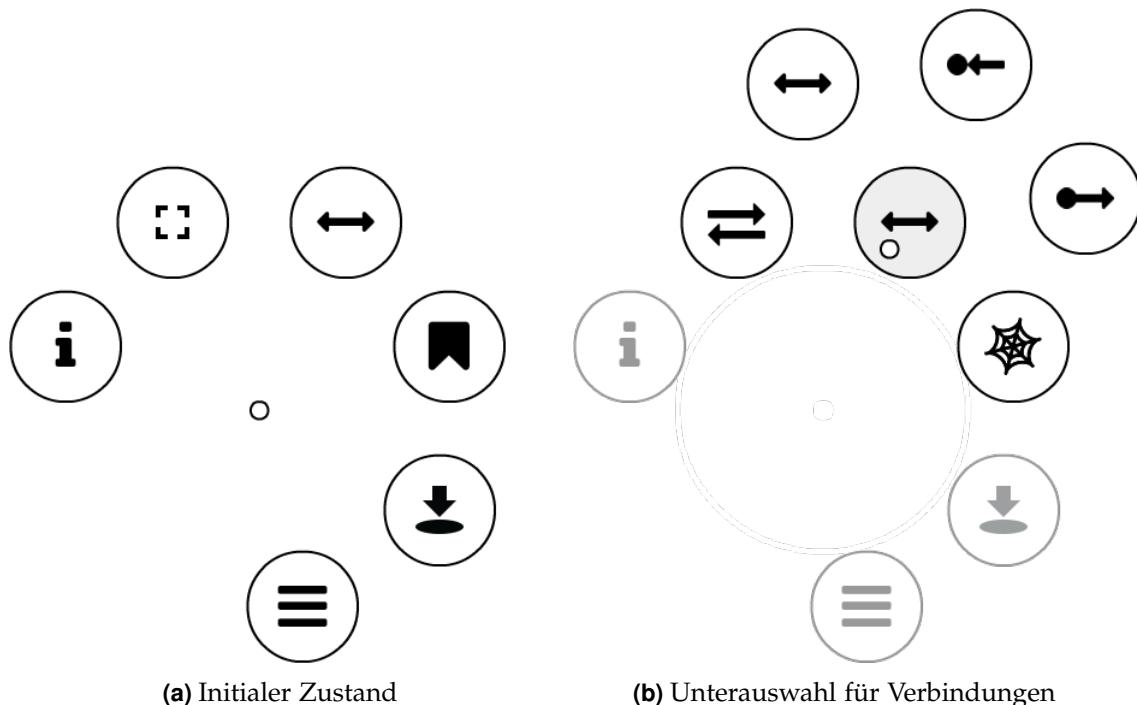


Abbildung A.2 Kontextmenü für einen inneren Knoten

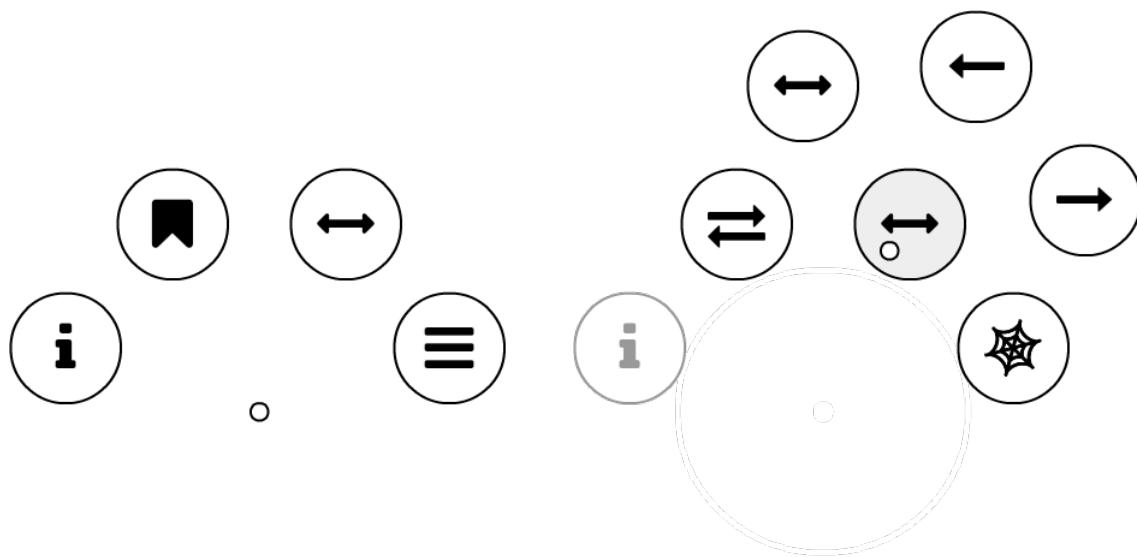


Abbildung A.3 Kontextmenü für eine Wurzel

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 (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 (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 (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 (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 (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: Aufl. 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-brexite-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 (besucht am 04.09.2017).
- [40] André Stultz. *The Introduction to Reactive Programming You've Been Missing*. Lizenziert unter CC BY-NC 4.0. 2016. URL: <https://gist.github.com/stultz/868e7e9bc2a7b8c1f754>.
- [41] Frank Steinbrück. „Consistent Software Cities: supporting comprehension of evolving software systems“. Diss. Cottbus, Brandenburgische Technische Universität Cottbus, 2013.
- [42] Helmut Vogel. „A better way to construct the sunflower head“. In: *Mathematical biosciences* 44.3-4 (1979), S. 179–189.
- [43] 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.
- [44] Johannes Weigend, Johannes Siedersleben und Josef Adersberger. „Dynamische Analyse mit dem Software-EKG“. In: *Informatik-Spektrum* 34.5 (2011), S. 484–495.

Literatur

- [45] 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).
- [46] 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).
- [47] 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.
- [48] 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).