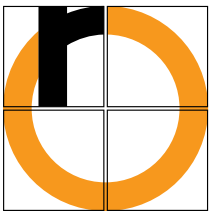
Hochschule **Rosenheim**
University of Applied Sciences



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 selbständig angefertigt, nicht anderweitig für Prüfungszwecke vorgelegt, keine anderen als die angegebenen Quellen oder Hilfsmittel benützt sowie wörtliche und sinngemäße Zitate als solche gekennzeichnet habe.

Rosenheim, den 14. Juni 2017

Marcel Pütz

Kurzfassung

here comes the abstract

Schlagworte: Softwarevisualisierung, Wald, Baum, 3D, Augmented Reality, Virtual Reality, Statische Codeanalyse, Abhängigkeiten

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

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

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 [10].

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 [8]. 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 [9].

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 Milgrim in [9] ein vollständigeres Verständnis, weshalb sich die Begrifflichkeiten in dieser Arbeit daran anlehnen sollen. Nach Milgrim 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 [7].

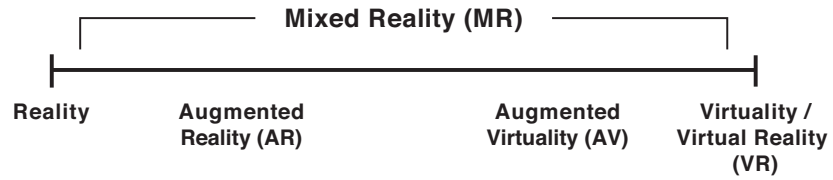


Abbildung 1.1 Abgewandelte Darstellung des Reality-Virtuality-Kontinuums aus [9]

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, 7] 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 [10].

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 heut 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“ [4]. 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 einfachere Analyse von Software zu ermöglichen.

In dieser Arbeit soll das neue Konzept *CodeLeaves* für eine solche Softwarevisualisierung 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 Userstories 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 2.3 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 5 sein. Besonders die Abhängigkeiten von Artefakten einer Software sollen mithilfe von CodeLeaves interaktiv exploriert werden können.

Das Kapitel 6 fasst die Ergebnisse der Arbeit zusammen und ein Ausblick auf zukünftige Verwendung und weiterführende Arbeiten runden die Arbeit ab.

2 Das Konzept CodeLeaves

2.1 CodeLeaves und die Metapher Software-Wald

"Hierarchies are almost ubiquitous [...]" [13] 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."* [5] stellen auch Ernst *et al.* fest. Draus 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

1. Die Struktur der Software wird mit nach oben wachsenden Bäumen dargestellt.
2. Jedes Paket im *Root-Verzeichnis* wird als einzelner Baum auf einer Ebene – dem *Waldboden* – dargestellt.
3. Die Blätter der Bäume entsprechen den Softwareartefakten (z.B. Klassen) und können durch ihre Farbe eine beliebigen Metrik visualisieren.
4. Zwischen den Bäumen entsteht ein *Wurzelgeflecht*, was die aggregierten Abhängigkeiten zwischen den Paketen darstellt.
5. Abhängigkeiten oder Aufrufe zwischen einzelnen Softwareartefakten werden aggregiert über die Elternpakete als

farbige bzw. Dicke der Äste dargestellt oder alternativ als 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 [15].

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 5 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 die kleinste betrachtete Einheit der Software verstanden. Das können bei objektorientieren Sprachen typischerweise Klassen, aber auch bei feinerer Granularität einzelne Funktionen innerhalb einer Klasse sein. Auch die Betrachtung von Dateien, die mehrere Klassen enthalten können, sind denkbar.

Punkt 4 bietet einen goßen 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.

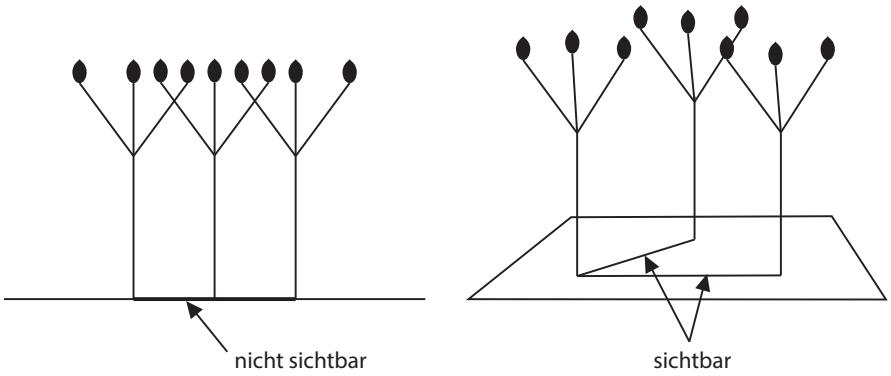
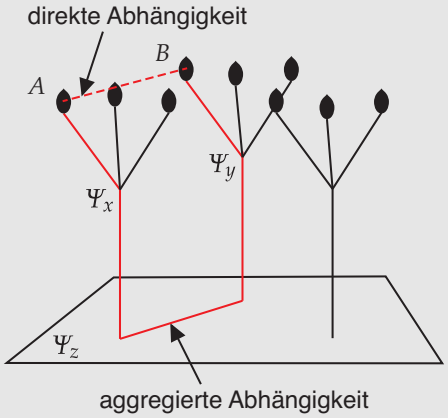


Abbildung 2.1 Vorteil der Dreidimensionalität bei der Darstellung von Abhängigkeiten

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

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.

2.2 Vergleich mit anderen Konzepten

Im Vorfeld dieser Arbeit wurde in [11] 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 GbmH durchgeführt. Die QAware ist ein Projekthaus mit den Kerngeschäften Diagnose, Sanierung, Exploration und Realisierung von Software [12]. Durch die Erfahrung in Projekten für namhafte Kunden, zeichnen sich die Mitarbeiter durch fundiertes Wissen und Expertise aus. Es wurden insgesamt 22 Mitarbeiter mit unterschiedlichen Rollen in der Softwareentwicklung befragt.

In Abbildung 2.2 ist zu sehen, wie oft welche Metriken genannt wurden. Alle Metriken lassen sich in die drei Kategorien der Softwarevisualisierung aus [4] einordnen und sind farbig entsprechend gruppiert.

Statik sind die Informationen, die ohne die Ausführung der Software generiert werden können [4]. 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 2.2 geht hervor, dass diese Informationen gleichzeitig am meisten von Interesse sind.

Dynamik beschreibt die Informationen, die zur Laufzeit einer Software generiert werden können [4]. 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.

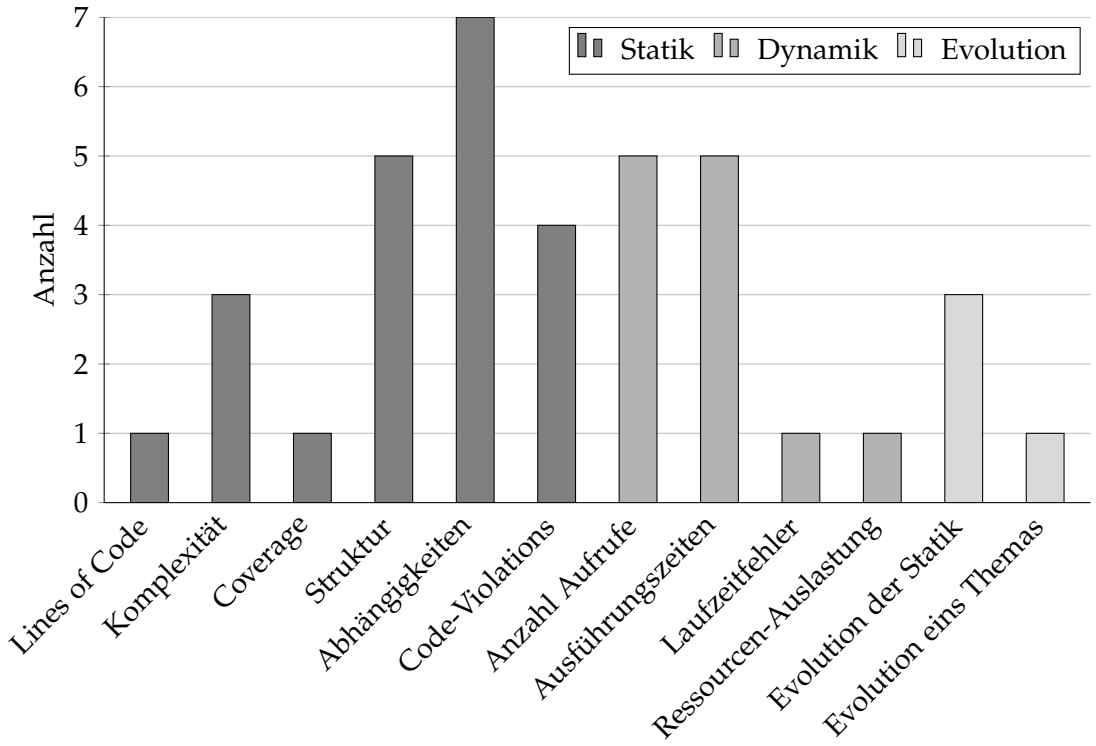


Abbildung 2.2 Von Mitarbeitern der QAware gewünschte Informationen

Evolution beschreibt den zeitlichen Verlauf einer Software und stellt den Entwicklungsprozess in den Vordergrund [4]. Beispielsweise kann die Entwicklung statischer Metriken verfolgt werden. Mit der Evolution eines Themas ist gemeint, dass anhand die Entwicklung eines bestimmten Themas nachverfolgt werden kann.

Aus den von den Mitarbeiten gewünschten Informationen und weitern Rahmenbedingungen wurde in [11] 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.

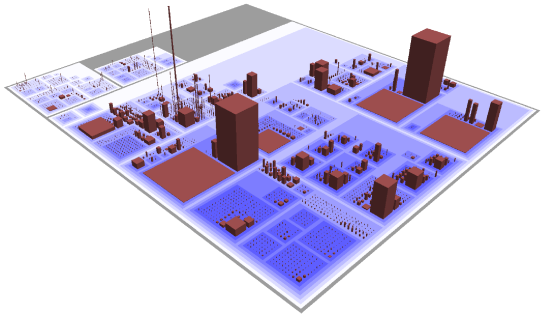
Bei der technischen Machbarkeit wurde berücksichtigt, ob eine existierende Softwarevisualisierung für die HoloLens verwendbar ist. In Abbildung 2.3 sind die untersuchten Alternativen abgebildet, darunter auch ein erster Entwurf von CodeLeaves.

CodeCity

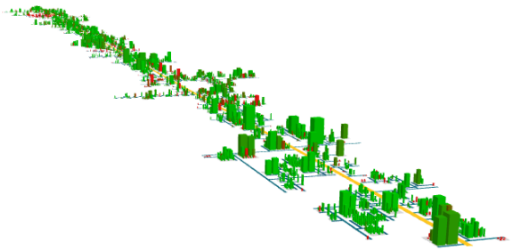
2007 stellten Wettel et al. CodeCity vor, die mithilfe der *Stadt-Metapher* dreidimensionale Städte visualisiert, in denen Klassen als Gebäude und Pakete als Stadtviertel dargestellt werden [15, 16, 17]. Für die Breite und Tiefe der Gebäude wurde für die Anzahl der Attribute (engl. *number of attributes (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 [17].

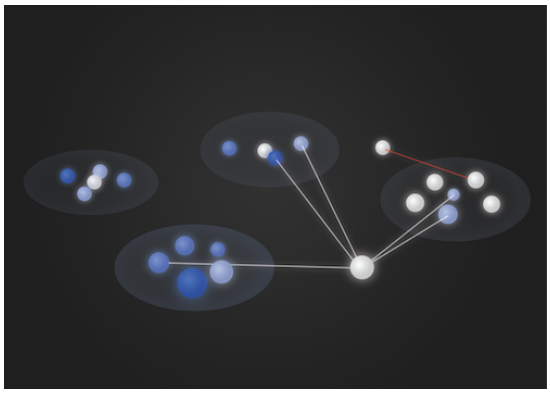
Jedoch unterstützt CodeCity keine Dynamik und die Abhängigkeiten sind nur als direkte Verbindungen darstellbar, was bei größeren Software-Systemen sehr unübersichtlich wird. Die verfügbaren statischen Metriken sind begrenzt und vor allem ist die Technologie Stand heute nicht mehr produktiv einsetzbar [11].



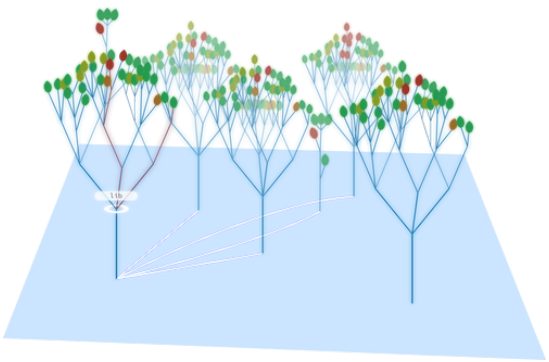
(a) CodeCity von ArgoUML [16]



(b) SoftVis3D mit Evostreet Layout



(c) Entwurf eines CodeUniverse



(d) Erster Entwurf von CodeLeaves

Abbildung 2.3 CodeLeaves und alternative Modelle

SoftVis3D

SoftVis3D greift das Konzept der CodeCity auf und visualisiert als Plugin für SonarQube[1] 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 [14] für die Evolution einer Software entworfen wurde. In diesem Layout, wie es in Abbildung 2.3b 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 [11].

CodeUniverse

Im Zuge der der Studie [11] wurde eine weitere Metapher evaluiert. Ähnlich wie in der Arbeit [6, 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 *„hierarchischem 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

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

dynamische Aufrufe zwischen Softwareartefakten nicht ohne Verlust der Übersichtlichkeit angezeigt werden. CodeLeaves soll alle drei Kategorien der Softwarevisualisierung 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.

2.3 Anforderungen an CodeLeaves

3 Datenmodell

3.1 FAMIX-Familie

4 Modellierung

5 Interaktionskonzept

6 Zusammenfassung und Ausblick

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] S. Diehl. *Software Visualization: Visualizing the Structure, Behaviour, and Evolution of Software*. Springer Berlin Heidelberg, 2007. ISBN: 9783540465041.

[5] 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.

[6] 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.

[7] 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.

[8] 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).

[9] 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.

[10] David Phelan. *Apple CEO Tim Cook: As Brexit hangs over UK, 'times are not really awful, there's some great things happening'*. 2017. URL: http://www.independent.co.uk/life-style/gadgets-and-tech/features/apple-tim-cook-boss-brexit-uk-theresa-may-number-10-interview-ustwo-a7574086.html#gallery (besucht am 30. 05. 2017).

[11] Marcel Pütz. „Softwarevisualisierung für Virtual Reality“. Masterseminar. Hochschule Rosenheim, 2017.

[12] QAware GmbH. *IT-Probleme lösen. Digitale Zukunft gestalten.* 2017. URL: http://www.qaware.de/leistung/#leistung-realisierung (besucht am 28. 03. 2017).

[13] 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.

[14] Frank Steinbrück. „Consistent Software Cities: supporting comprehension of evolving software systems“. Diss. Cottbus, Brandenburgische Technische Universität Cottbus, 2013.

[15] 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.

[16] 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.

[17] 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.