Technische Universität Dresden
Bereich Mathematik und Naturwissenschaften
Fakultät Psychologie
Professur für Ingenieurpsychologie und Angewandte Kognitionsforschung

# BACHELOR-ARBEIT

zum Thema

# Anforderungsanalyse für eine Softwarevisualisierung zur Einarbeitung neuer Mitarbeitenden in bestehende Softwareprojekte

in Kooperation mit

Deutsches Zentrum für Luft- und Raumfahrt
Simulations- und Softwaretechnik

| | |
|---|---|
| eingereicht von | Jana-Sophie Effert |
| geb. am | 24. Juli 1996 in Mutlangen |
| Matrikelnummer | 4520439 |

| | |
|---|---|
| 1. Gutachter und Betreuer | Prof. Dr. Sebastian Pannasch |
| 2. Gutachter | Prof. Dr. Stefan Scherbaum |

Dresden, den 06.September 2019

# Inhaltsverzeichnis

# 1 Zusammenfassung

In der Einrichtung Simulations- und Softwaretechnik des Deutschen Zentrums für Luft- und Raumfahrt soll zur Unterstützung neuer Mitarbeitenden bei der Einarbeitung in bestehende Softwareprojekte eine Softwarevisualisierung entwickelt werden. Das Ziel der vorliegenden Arbeit ist es, zu untersuchen, wie der Prozess der Einarbeitung in bestehende Softwareprojekte zu definieren ist, ob der Bedarf an Unterstützung im Einarbeitungsprozess vorhanden ist, und welche Anforderungen an eine Softwarevisualisierung zu diesem Zwecke vorliegen. Dazu wurden fünf durchschnittlich 45-minütige Interviews mit Softwareentwickler*innen der Einrichtung durchgeführt. Diese wurden transkribiert und mittels qualitativer Inhaltsanalyse ausgewertet.

Der Einarbeitungsprozess lässt sich in zwei Phasen untergliedern. Zunächst ist es Teil der Einarbeitung, den Anwendungsfall der Software zu verstehen und die Abteilung mit den zuständigen Kolleg*innen der jeweiligen Bereiche kennenzulernen. In der zweiten, immer wiederkehrenden Phase werden mehrere für neue Mitarbeitende gekennzeichnete Aufgaben gelöst. Durch Bearbeitung dieser Aufgaben lernen Mitarbeitende diverse Stellen des Codes kennen und entwickeln von innen heraus ein mentales Modell der Softwarestruktur. Für die beiden Phasen müssen entweder separate Visualisierungen erstellt werden oder eine Visualisierung, bei der sich die in der ersten Phase benötigten Zusatzinformationen deaktivieren lassen.

Die Nützlichkeit einer Softwarevisualisierung zur Unterstützung wurde zwar von manchen Interviewten angezweifelt, es sahen aber vor allem diejenigen Softwareentwickler*innen Potential in dieser Form der Unterstützung, die bereits mit Softwarevisualisierungen gearbeitet haben. Eine Unterstützung im Einarbeitungsprozess könnte den langen Prozess verkürzen und die Notwendigkeit aufheben, dauerhaft ein mentales Modell im Hinterkopf aufrechtzuerhalten.

Die aus den Interviews abgeleiteten und teilweise von den Interviewten selbst konkret formulierten Anforderungen stimmen zu großen Teilen mit den in der Literatur gefundenen Anforderungen überein. Vor allem die einfache Verbindung von Visualisierung und Quellcode bzw. Entwicklungsumgebung spielt eine zentrale Rolle. Außerdem ist das Einblenden von bestimmten Zusatzinformationen relevant, genauso wie die Möglichkeit des Ausblendens von für die aktuelle Aufgabe irrelevanten Bestandteilen. Die Verwendung von VR-Methoden wird zum aktuellen Zeitpunkt kritisch gesehen, da ein zu großer Zeitaufwand damit einhergeht.

Es wurden insgesamt 20 Anforderungen definiert, deren Realisierung in einer Softwarevisualisierung dazu beitragen kann, die Einarbeitung neuer Mitarbeitenden in ein Softwareprojekt zu unterstützen.

# 2 Einleitung

Bereits im Jahr 1972 beschrieb der Turing Award Gewinner Edsger W. Dijkstra in seiner Turing-Lecture „The humble programmer“ eine Entwicklung, die als „Software-Krise“ Bekanntheit fand. Die schnelle technische Weiterentwicklung von Prozessoren und Rechnern resultierte in einer enormen Komplexitätssteigerung in der Programmierung. Quellcodes wurden umfangreicher, Softwareprojekte unübersichtlicher. Unter anderem durch die daraus folgende, immer länger werdende Entwicklungsdauer überstiegen die Kosten für die Software erstmals die Kosten der Hardware (Dijkstra, 1972).

Immer schneller entwickeln sich technische Möglichkeiten weiter, und mittlerweile ist es der Regelfall, dass eine Vielzahl an Softwareentwickler*innen an jeweils unterschiedlichen Teilen eines Softwareprojekts arbeiten (Gerstl, 2018). Das sogenannte Brooks'sche Gesetz: „Adding manpower to a late software project makes it later“ (Brooks, 1974) beschreibt, dass mehr Mitarbeitende am Projekt nicht zwangsläufig zu einem früheren Abschluss führen. Das nachträgliche Hinzuziehen zusätzlicher Softwareentwickler*innen zum Projekt bringt eine Zeitverzögerung mit sich. Dies liegt unter anderem am Zeitaufwand der Einarbeitung neuer Mitarbeitenden, sowie dem mit der Anzahl der Beteiligten steigenden Kommunikationsbedarf während der Entwicklung. Je mehr Personen an dieser beteiligt sind, desto größer ist der Aufwand, sich gegenseitig abzustimmen und auf den neuesten Stand zu bringen (Brooks, 1974). Bei andauernden Projekten ist die Einarbeitung neuer Entwickler*innen allerdings unvermeidlich, weshalb nach Möglichkeiten gesucht wird, diese Phase so effizient wie möglich zu gestalten.

Eine Methode, die die Kommunikation über Daten und zugrundeliegende Modelle erleichtert, sowie für ein verbessertes Verständnis und eine einfachere Analyse sorgt, ist die Visualisierung (McCormick, 1988). Auch im Anwendungsfall der Softwareentwicklung wird an Visualisierungen gearbeitet. Die wahrgenommenen Vorteile von Softwarevisualisierungen liegen hauptsächlich in der Zeitersparnis bei Ausführung spezifischer Aufgaben im Entwicklungsprozess und im besseren Verständnis der Softwarearchitektur (Bassil & Keller, 2001). Fjeldstad und Hamlen (1982, zitiert nach Banker, Davis & Slaughter, 1998) zeigen auf, dass Wartungsprogrammierer die Hälfte ihrer Arbeitszeit damit verbringen, die Software verstehen zu lernen, für deren Wartung sie eingestellt sind. Bassil und Keller (2001) untersuchten verschiedene Softwarevisualisierungen, welche nicht alle denselben Anwendungsfall hatten. In der vorliegenden Arbeit wird dieser Anwendungsfall auf das Szenario der Einarbeitung in ein neues Softwaresystem begrenzt. Da auch hier vor allem der Verständnisgewinn im Vordergrund steht, liegt nahe, dass eine Visualisierung auch im Prozess der Einarbeitung in eine neue Software neue Mitarbeitende unterstützen kann.

Beim Deutschen Zentrum für Luft- und Raumfahrt in der Einrichtung für Simulations- und Softwaretechnik, Abteilung Intelligente und Verteilte Systeme, soll zu diesem Zweck eine neue Softwarevisualisierung entwickelt werden. Der erste Schritt der Entwicklung interaktiver Systeme besteht in einer Anforderungsanalyse, bei der die Aufgaben und Prozesse, die durch das zu entwickelnde System unterstützt werden sollen, charakterisiert werden (Preim & Dachselt, 2010). Unvollständigkeit bei der Formulierung der Anforderungen oder unzureichende Einbeziehung der Nutzer*innen sind Hauptursachen für das Scheitern von Software-Projekten (The Standish Group, 1994). Um sicherzustellen, dass die neue Softwarevisualisierung für die intendierte Nutzung geeignet ist, und sie alle Nutzer*innen in ihren Aufgaben ideal unterstützt, sollen im Rahmen dieser Arbeit die Anforderungen an die Softwarevisualisierung definiert werden.

Im folgenden Kapitel werden Erkenntnisse aus dem Bereich der Informationsvisualisierung zusammengefasst und verschiedene Klassifizierungsmöglichkeiten der Softwarevisualisierung präsentiert. Außerdem wird die Rolle der Anforderungsanalyse im Entwicklungsprozess eines interaktiven Systems aufgezeigt und welche verschiedenen Ansätze es bei einer solchen Analyse gibt. Anschließend wird im

Kapitel 4 die für die vorliegende Anforderungsanalyse gewählte Methode der Befragung vorgestellt, der Interviewleitfaden präsentiert und das Vorgehen der qualitativen Inhaltsanalyse erläutert. Die Ergebnisse, die aus den Interviews gewonnen wurden, werden im fünften Kapitel aufgezeigt und im abschließenden Kapitel der vorliegenden Arbeit diskutiert.

# 3 Theorie

## 3.1 Visualisierung

### 3.1.1 Definition und Qualitätseinflüsse

Visualisierungen verbinden die beiden leistungsfähigsten bekannten Informationsverarbeitungssysteme: den modernen Computer und das menschliche Gehirn (Gershon & Eick, 1997) – insbesondere die visuelle Wahrnehmung und damit die Fähigkeit des menschlichen Sehsystems, räumliche Strukturen und Muster zu erkennen (Robertson, 1990). Als Visualisierung wird der Prozess der Umwandlung von Daten in eine visuelle Form bezeichnet, die es den Nutzer*innen erlaubt, nicht direkt sichtbare Aspekte, wie Zusammenhänge und Strukturen in diesen Daten zu entdecken (Diehl, 2007). Dabei werden hauptsächlich die relevanten Aspekte der Daten dargestellt, die vom Beobachtenden leicht und intuitiv verstanden werden können. Es ist zunächst zweitrangig, ob diese Daten von vorneherein visueller Natur sind und nur neu strukturiert oder ins Visuelle transformiert werden (Haber & Wilkinson, 1982). Die Visualisierung ermöglicht, dass sich die Nutzer*innen ein mentales Modell der Daten und der mit diesen verbundenen Modelle und Prozesse bilden, welches die grundliegenden Zusammenhänge beschreibt (Schumann & Müller, 2013). Dies zeigt nach McCormick (1988) eine der beiden Hauptaufgaben von Visualisierungen auf: die Analyse von Daten. Er beschreibt, dass die Visualisierung das Verständnis fördert, da der Prozess über das reine Sehen hinausgeht und in Erkennen, Verstehen und Bewerten von vorliegenden Phänomenen resultieren kann. Visualisierungen werden sowohl in der explorativen Analyse eingesetzt, bei der noch keine Vorannahmen über mögliche Zusammenhänge vorliegen, als auch in der konfirmatorischen Analyse, also bei der Überprüfung von Hypothesen. Die zweite Hauptaufgabe von Visualisierungen ist laut McCormick (1988) die Unterstützung bei der Präsentation von Daten. Bestenfalls erzeugt die Visualisierung ähnliche mentale Modelle bei allen Betrachtenden und erreicht somit, dass die Kommunikation über Strukturen und Zusammenhängen erleichtert wird. Der Erfolg von Visualisierungen ist nach Robertson (1990) davon abhängig, inwieweit Nutzer*innen in der Lage sind, den Kontext der realen Welt aus der Abbildung zu rekonstruieren.

### 3.1.2 Informationsvisualisierung

Über Visualisierungen lassen sich verschiedene Arten von Daten abbilden. Man unterscheidet zwischen wissenschaftlicher Visualisierung und Informationsvisualisierung. Wissenschaftliche Visualisierung beschreibt die Abbildung physikalischer Daten, Maße oder Simulationsoutputs. Diese Daten rufen oftmals konkrete mentale Bilder bei den Nutzer*innen hervor, da beispielsweise das Strömungsverhalten an Flussgabelungen oder die Kräfteverteilung an einem Gebäude konkrete räumliche Bezüge aufweisen (Preim & Dachselt, 2010). Informationsvisualisierung wird im Gegensatz dazu von Card (1999) wie folgt definiert:

*Informationsvisualisierung ist die Nutzung computergenerierter, interaktiver, visueller Repräsentationen von abstrakten, nicht-physikalischen Daten zur Verstärkung des Erkenntnisgewinns.*

Der Fokus liegt auf Daten ohne inhärent räumlichen Bezug. Deshalb ist es eine bedeutende Aufgabe, passende visuelle Metaphern zu finden, die die Informationen geeignet repräsentieren und je nach Analyseaufgabe das Verständnis der Zusammenhänge oder Strukturen zu unterstützen (Preim & Dachselt, 2010). Aus dem Information Seeking Mantra nach Shneiderman (1996) „Overview first, zoom and filter, and then details-on-demand“ lassen sich drei charakteristische Aspekte von Informationsvisualisierungen ableiten (Schumann & Müller, 2004).

1. Um verschiedene visuelle Zugänge zu großen Informationsbeständen zuzulassen, werden unterschiedliche Bilder erzeugt.
2. Um zwischen diesen Bildern wechseln zu können, gibt es Interaktionstechniken, die die Auswahl der aktuell präsentierten Sicht zur Verfügung stellen.
3. Um Details abzurufen oder unwichtige Informationen ausblenden zu können, werden Algorithmen in die Visualisierung integriert, die das gezielte Zu- und Wegschalten von Informationen ermöglichen.

Diese Aspekte finden sich in nahezu allen Informationsvisualisierungen und demnach auch in der zugehörigen Unterform, der Softwarevisualisierung, die im Folgenden vorgestellt wird.

### 3.1.3 Softwarevisualisierung

#### 3.1.3.1 Definition und Arten

Wartungsprogrammierer verbringen etwa die Hälfte der Arbeitszeit an einem Projekt allein damit, die Software zu verstehen, die geändert werden soll (Fjeldstad & Hameln, 1982, zitiert nach Banker et al., 1998). Hier wird der Bedarf an einem Instrument deutlich, welches diesen Zustand verbessert und das Verständnis der Software erleichtert. Deshalb gibt es spätestens seit der ersten internationalen Konferenz zu diesem Thema im Jahr 2003 verstärkt Forschung zum Einsatz von Softwarevisualisierung (Gerstl, 2018). Auch hier werden Daten visualisiert, die sich durch ihre Abstraktheit auszeichnen, keine physische Form oder Größe haben. Deutlich wird der Unterschied zu wissenschaftlicher Visualisierung bei Betrachtung von Visualisierungsassoziationen. Die Veränderungshistorie eines Softwarearchivs weckt keine natürliche Assoziation. Durch solche Daten können höchstens erlernte Assoziationen hervorgerufen werden, wie beispielsweise UML-Diagramme (Preim & Dachselt, 2010). Softwarevisualisierung kann Softwareentwickler*innen helfen, mit der Komplexität des Softwaresystems umzugehen und die eigene Produktivität bei der Arbeit zu erhöhen. Die Visualisierungen versprechen ein erhöhtes Verständnis der Software, welches oftmals durch ein erhöhtes Abstraktionsniveau, eine Reduktion des Informationsumfangs auf den für die aktuelle Aufgabe relevanten Teil und eine Vereinfachung des Durchsuchens des Datenraums entsteht (Koschke, 2003).

In der Literatur werden verschiedene Arten der Einteilung von Softwarevisualisierungen vorgenommen. Meist haben diese als gemeinsame Grundlage die Einteilung nach Abstraktionsgrad der dargestellten Informationen. Ball und Eick (1996) unterscheiden drei grundlegende Eigenschaften von Software, die visualisiert werden können: Der Quellcode, die Softwarestruktur oder das Laufzeitverhalten. Panas, Epperly, Quinlan, Saebjornsen und Vuduc (2007) sprechen ebenfalls von drei Abstraktionsleveln: Dem Quellcodelevel, dem Architekturlevel und einem mittleren Level, welches problemspezifisch für die Lösung einer bestimmten Aufgabe genutzt wird, beispielsweise Sequenzdiagramme.

Price, Baecker und Small (1993) unterteilen Softwarevisualisierungen zunächst grob in zwei Kategorien: Programmvisualisierung, welche sowohl Code- als auch Datenvisualisierung beinhaltet, und Algorithmenanimation. Die Autoren bleiben allerdings nicht auf dieser Einteilungsebene sondern liefern Fragen, nach denen Visualisierungen klassifiziert werden können, die hier im Folgenden aufgelistet werden.

**1 Umfang:** Wie hoch ist der Programmumfang, den das Softwarevisualisierungs-System als Input für die Visualisierung nehmen kann?

**2 Inhalt:** Welche Teilinformationen über die Software werden vom Softwarevisualisierungs-System visualisiert?

**3 Form:** Welche Eigenschaften hat die Ausgabe des Softwarevisualisierungs-Systems (z.B. Medium, Detailgrad)?

4 **Methode:** Wie wird die Visualisierung konkretisiert (z.B. Verbindung Visualisierung und Software)?
5 **Interaktion:** Wie steuern die Nutzer*innen das Softwarevisualisierungs-System?
6 **Effektivität:** Wie gut übermittelt das Softwarevisualisierungs-System Informationen an die Nutzer*innen?

#### 3.1.3.2 Anforderungen

Bassil und Keller (2001), definieren Softwarevisualisierungen als Werkzeug, das das Verständnis von Softwaresystemen durch grafische und/oder textuelle Darstellungen unterstützt. In einer Fragebogenstudie untersuchten sie, welche Vorteile vorhandene Softwarevisualisierungen mit sich bringen und welche Aspekte davon den Nutzer*innen wichtig sind. Sie führten die erste quantitative Untersuchung von Softwarevisualisierungstools durch. Teilnehmende sollten 34 Aspekte von Softwarevisualisierungstools nach ihrer Nützlichkeit bewerten. Als wichtigste Vorteile von Softwarevisualisierung wurden im Rahmen der Studie die Zeitersparnis bei Ausführung bestimmter Aufgaben angegeben und ein besseres Verständnis der Softwarestruktur. Der bedeutendste Aspekt war ein einfacher Zugang vom verwendeten Symbol zum Quellcode. Die funktionalen Aspekte von Suchen und Browsing, einer hierarchischen Repräsentation und Navigation über Hierarchien waren erwünscht, sowie die Verwendung von Farben. Die geringste Nützlichkeit wurde Animationseffekten, 3D-Visualisierungen und VR-Techniken zugeschrieben.

Eine weitere Fragebogenstudie ergänzte die Bewertung bereits existierender Softwarevisualisierungen um Fragen, wie eine gute Visualisierung aussehen solle (Koschke, 2003). 82% der Teilnehmenden, Forschende in den Bereichen Softwarewartung, Reverse Engineering und Reengineering, gaben an, dass sie Softwarevisualisierungen für absolut notwendig oder zumindest wichtig in ihrer Domäne hielten. Auch hier wurde betont, dass die Verbindung von der Visualisierung zum Code aufrechterhalten werden muss, wobei sogar ergänzt wurde, dass diese auch in die andere Richtung bestehen solle. Um die Frage der Nützlichkeit beantworten zu können, bezogen sich viele Teilnehmende konkret auf die Softwarevisualisierungen, die ihnen bekannt waren. Bei Betrachtung der Ergebnisse ist im Hinterkopf zu behalten, dass es diverse Arten der Softwarevisualisierung gibt, deren Qualität von verschiedenen Einflussgrößen abhängt (Schumann & Müller, 2013). Je nach Anwendungsfall, üblicherweise genutzten Metaphern etc. sind Softwarevisualisierungen von unterschiedlich großem Nutzen.

Um eine geeignete Visualisierung zu entwickeln muss der Nutzungskontext zunächst definiert werden. Außerdem müssen an die Visualisierung vorliegende Anforderungen spezifiziert werden. Dies geschieht im Rahmen einer Anforderungsanalyse, die als Methode im nachfolgenden Abschnitt genauer beschrieben wird.

## 3.2 Anforderungsanalyse

Die Hauptursachen für das Scheitern von Software-Projekten sind Unvollständigkeit bei der Formulierung der Anforderungen oder unzureichende Einbeziehung der Nutzer*innen (The Standish Group, 1994). Dies macht deutlich, wie grundlegend ein präzises Verständnis von dem ist, was ein System am Ende leisten muss, bevor dieses System entwickelt werden kann. Eine Schwierigkeit besteht darin, die oftmals nicht greifbaren Anforderungen zu spezifizieren. Balzert beschreibt Anforderungen in ihrer Natur als „vage, verschwommen, unzusammenhängend, unvollständig und widersprüchlich“ und die Aufgabe des Definitionsprozesses sei es, „aus diesen ein vollständiges, konsistentes und eindeutiges Produkt-Modell zu erstellen“ (Balzert, 2000, S.98).

Ein starker Fokus auf die letztliche Interaktion der Nutzer*innen mit dem entwickelten Produkt

wird im Bereich des Usablity Engineering gesetzt. Nach ISO-Norm 9241-11 wird Usability definiert als das Ausmaß, in dem ein Produkt durch bestimmte Benutzer*innen in einem bestimmten Nutzungskontext genutzt werden kann, um bestimmte Ziele effektiv, effizient und zufriedenstellend zu erreichen. Um ein hohes Ausmaß an Usability zu erreichen, werden Usability-Methoden im gesamten Entwicklungsprozess eines neuen Produktes genutzt. Dieser umfasst sowohl die in dieser Arbeit durchgeführte Anforderungsanalyse als auch das Modellieren, die Spezifikation, Realisierung und Evaluation des Produkts. In der Praxis wird dieser Prozess häufig iterativ durchgeführt, wodurch eine klare Trennung der einzelnen Phasen schwierig ist (Richter & Flückiger, 2013).


Abbildung 3.1: Optimierung der Benutzbarkeit nach Richter und Flückiger (2013, S.6)

In der Phase der Analyse besteht die Hauptaufgabe darin, ein Verständnis für den Kontext des zu entwickelnden Systems zu erlangen. Wie in der Übersicht von Richter und Flückiger (2013) (Abb.3.1) dargestellt, ist es für die Optimierung der Usability grundlegend, die Ziele, Abläufe und Anwendungen des Produkts zu kennen und zu verstehen, wo und wie die Nutzer*innen durch das Produkt effektiv, effizient und zufriedenstellend unterstützt werden können. Darauf aufbauend können die Funktionalitäten festgelegt und die Benutzungsoberfläche gestaltet werden.

Die Situation, die eine Neuentwicklung oder Weiterentwicklung des Systems erforderlich macht, wird in einer initialen Analyse charakterisiert. Nach Newman, Lamming und Lamming (1995) besteht der erste Schritt in der Beschreibung dieser Situation of Concern (SoC), welche verändert werden soll. Auf der Definition der SoC aufbauend, wird ein Problem Statement erstellt, welches erste Angaben zur Lösungsform, Art der Unterstützung, unterstützten Aktivität und Charakteristika der Nutzer*innen beinhaltet (Newman et al., 1995). Dieses Statement beantwortet einfacher formuliert die Frage „Wer wird wie und in welchen Aktivitäten unterstützt?“ und bietet die Grundlage für eine detailliertere Analyse. Nutzer*innen und andere für die Entwicklung wesentliche Personen müssen dafür identifiziert, kontaktiert, beobachtet bzw. befragt werden. Durch den Kontakt zu diesen Personen sollen die von den Nutzer*innen ausgeführten Aufgaben charakterisiert werden. Ablaufende Aktivitäten, Abhängigkeiten, Häufigkeiten und Dringlichkeiten von Aufgaben werden erfasst, genauso wie kritische Fehlersituationen (Preim & Dachselt, 2010).

Um an diese Informationen zu gelangen, stehen mehrere Methoden zur Verfügung. Die Befragung von Expert*innen, was im Fall einer Aufgabenanalyse zunächst alle sein können, die diese Aufgabe häufiger ausführen, ist oftmals naheliegend. Diese Befragung hat die Rekonstruktion der Wissensbestände von Expert*innen als Ziel (Pfadenhauer, 2009). Im Fall der Anforderungsanalyse handelt es sich um das Wissen, wie aktuelle Aufgaben erledigt werden. Oftmals ist diese Methode allerdings aus zwei Gründen

weniger effektiv. Zum einen handelt es sich bei den Aufgaben meistens um eine für die Person mit Expertise automatisierte Vorgehensweise. Viele Schritte werden bei der Beschreibung der Tätigkeit vergessen, da über diese nicht mehr aktiv nachgedacht werden muss. Außerdem sind Expert*innen oftmals in Besitz von implizitem und damit schwer zu verbalisierendem Wissen, handeln nach Bauchgefühl und mit reichlichem Erfahrungsschatz als Entscheidungshilfe (Preim & Dachselt, 2010).

Um diesen Gefahren einer reinen Expert*innenbefragung entgegenzuwirken ist die vorzuziehende Vorgehensweise eine Beobachtung am realen Arbeitsplatz. Hier werden auch die Vorgänge deutlich, die bei einem Interview vergessen werden können, da sie den Befragten selbstverständlich erscheinen. Ergebnisse aus der Beobachtung können dazu beitragen, dass erkannt wird, welche Fragen in einem späteren Interview zu stellen sind (Preim & Dachselt, 2010). Hier wird deutlich, dass die Kombination mehrerer Erhebungsmethoden das Mittel der Wahl sein sollte, um alle Informationen für eine Anforderungsanalyse zu erheben. Auf die Beobachtung folgende Interviews erfassen, neben der Vorgehensweise der Personen bei der Bewältigung von Aufgaben, auch ihre Meinung. Eine Serie von Interviews ist zu empfehlen, da zur Entwicklung eines interaktiven Systems Einzelmeinungen weniger relevant sind. Für die ersten Interviews muss mehr Zeit eingeplant werden, da in späteren Interviews mit weniger neuen Einsichten zu rechnen ist (Nielsen, 1993). Für die Interviews sind teilstrukturierte Interviewleitfäden am ergiebigsten, da durch die Leitfragen sichergestellt wird, dass alle relevanten Themenbereiche angesprochen und bearbeitet werden. Gleichzeitig wird genügend Flexibilität geboten um auf die Interviewpartner*innen eingehen zu können, Nachfragen zu stellen und offen zu sein für unerwartete Informationen (Preim & Dachselt, 2010). Es ist notwendig, dass die interviewende Person sich gut in die Materie eingearbeitet hat und somit über genügend Hintergrundwissen verfügt, um Fachausdrücke zu verstehen und Informationen einzuordnen.

Die über Beobachtungen und Interviews gewonnenen Informationen müssen zu konkreten Anforderungen definiert werden. Preim und Dachselt (2010) beschreiben diesen Schritt als einen kreativen Syntheseprozess. Die Anforderungsspezifikation muss nach Glinz (2002) bestimmte Kriterien erfüllen: Adäquatheit, Vollständigkeit, Widerspruchfreiheit, Verständlichkeit, Eindeutigkeit und Prüfbarkeit. Damit soll sichergestellt werden, dass die Spezifikation auf alle Punkte eingeht, die von den Nutzer*innen gewünscht und erwartet werden, dass diese sich nicht gegenseitig ausschließen und in ihrer Art nachvollziehbar und somit umsetzbar sind. Nach der Implementierung kann dadurch evaluiert werden inwieweit diese Anforderungen umgesetzt wurden.

## 3.3 Anwendungsfall DLR

Die in den vorherigen Abschnitten beschriebenen Inhalte der Softwarevisualisierung und des Vorgehens bei Anforderungsanalysen sollen im Rahmen dieser Bachelorarbeit verbunden werden. In der Einrichtung Simulations- und Softwaretechnik des Deutschen Zentrums für Luft- und Raumfahrt wird eine neue Softwarevisualisierung entwickelt, welche neue Softwareentwickler*innen bei der Einarbeitung in die Softwaresysteme unterstützen soll. Die Einrichtung beschäftigt sich mit innovativen Software-Engineering-Technologien, wobei aktuelle Schwerpunkte in der Softwareentwicklung für verteilte und mobile Systeme, Software für eingebettete Systeme, Visualisierung und High Performance Computing liegen. Ein derzeitiges Projekt ist die Arbeit an der Remote Component Environment, kurz RCE, welche eine Integrationsumgebung zur Analyse, Optimierung und Konstruktion komplexer Systeme, wie beispielsweise beim Flugzeugbau, liefert. Dadurch wird es möglich, in verschiedenen Programmiersprachen geschriebene Tools zusammenzuführen. Dies dient vor allem der Zusammenarbeit von Mitarbeitenden unterschiedlichster Fachgebiete, da die separat entwickelten Tools in eine einheitliche Umgebung integriert werden.

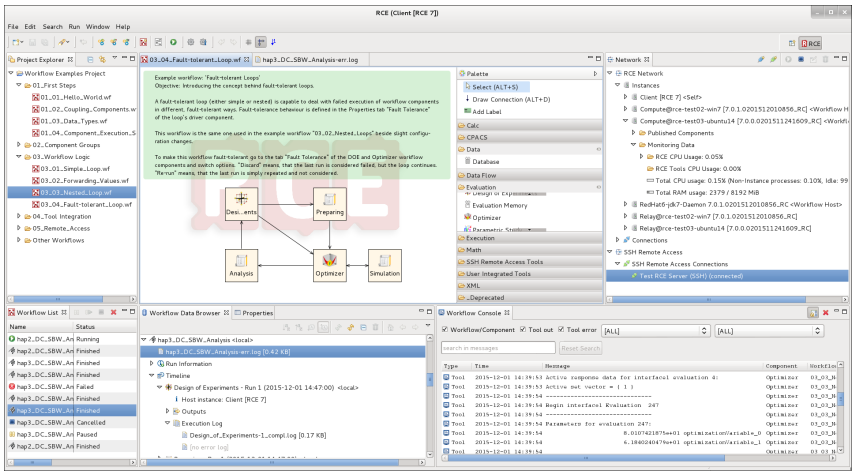


Abbildung 3.2: Screenshot der RCE-Oberfläche (veröffentlicht auf www.rcenvironment.de)

Zeitgleich wird an einer bestehenden Softwarevisualisierung geforscht, die die Architektur von OSGi-basierten Softwaresystemen darstellen kann. Die Visualisierung stellt mit Hilfe einer Inselmetapher die Komponenten einer OSGi-basierten Softwarearchitektur - Bundles, Packages und Klassen - dar und bietet die Möglichkeit, sich Abhängigkeiten anzeigen zu lassen (vgl. Abb.3.3). Dabei dienen Inseln der Darstellung von Bundles, die darin befindlichen Regionen und Gebäude repräsentieren Packages und Klassen. RCE basiert auf dem OSGi-Framework und dient deshalb als Grundlage der Visualisierung.

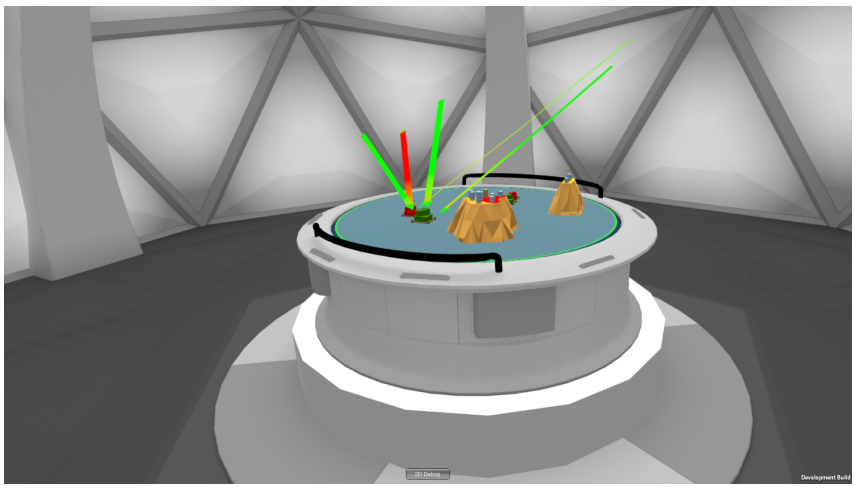


Abbildung 3.3: Visualisierung der RCE Software mithilfe einer Inselmetapher (DLR Abbildung)

Im Rahmen der Forschung an Softwarevisualisierungen in dieser Einrichtung wird nun eine weitere Visualisierung entwickelt, welche neue Mitarbeitende bei der Einarbeitung in eine Software unterstützen soll. Die vorliegende Anforderungsanalyse soll sicherstellen, dass die neue Softwarevisualisierung für die intendierte Nutzung geeignet sein wird.

Die nach Newman et al. (1995) zu definierende Situation of Concern als Ausgangspunkt der Anforderungsanalyse lässt sich für die vorliegende Arbeit wie folgt beschreiben:

*Das Verständnis des Aufbaus und der Funktion einer Software ist Voraussetzung für die weitere Arbeit mit dieser. Softwaresysteme wie RCE werden dabei immer größer und unübersichtlicher, was dazu*

*führt, dass sehr viel Zeit für die Einarbeitung in solche Systeme notwendig ist, bevor mit der eigentlichen Entwicklungsarbeit begonnen werden kann.*

Daraus folgt das Problem Statement des Anwendungsfalls, welches sowohl die Nutzer*innen, als auch die durch das System unterstützen Aktivitäten beschreibt und gleichzeitig die Art der Unterstützung und die grundliegende Lösungsform benennt (Newman et al., 1995):

*Neue Softwareentwickler*innen der DLR Einrichtung Simulations- und Softwaretechnik sollen bei der Einarbeitung in Softwaresysteme unterstützt werden. Dazu wird eine Softwarevisualisierung entwickelt, welche den Nutzer*innen hilft, für den Einarbeitungsprozess relevante Informationen abzurufen und mentale Modelle über den Aufbau und die Struktur der Software zu entwickeln.*

Ausgehend von diesem Problem Statement wird eine ausführliche Anforderungsanalyse erstellt, wofür im Rahmen dieser Bachelorarbeit mehrere Fragen geklärt werden:

1. Wie lässt sich der Anwendungsfall „Einarbeitung in eine neue Software“ definieren und welche Aufgaben werden dabei von den Softwareentwickler*innen in welcher Reihenfolge erledigt?
2. Bei welchen Aufgaben kann eine Softwarevisualisierung die Softwareentwickler*innen dabei unterstützen?
3. Welche Anforderungen liegen an eine solche Software vor, damit die Nutzung effektiv, effizient und zufriedenstellend ist?

# 4 Methoden

## 4.1 Teilstrukturiertes Interview

Nach Preim und Dachselt (2015) gibt es mehrere Methoden, die sich für eine Anforderungsanalyse eignen. Sie empfehlen die Beobachtung am Arbeitsplatz gefolgt von Interviews, um realistische Einblicke in aktuelle Vorgehensweisen zu bekommen und durch die Beobachtungen herauszufinden, was im Rahmen von Interviews noch erfragt werden muss. Beobachtet und befragt werden sollten im DLR in der Einrichtung Simulations- und Softwaretechnik Softwareentwickler*innen, da diese die zukünftige Nutzer*innengruppe der Visualisierung sind. Damit festgelegt werden kann, welche Teilaufgaben und Arbeitsschritte beobachtet werden können, muss der Prozess der Einarbeitung in eine neue Software definiert sein. Diese Definition ist in diesem Fall nicht von Beginn an vorgegeben, sondern soll im Rahmen dieser Arbeit erfolgen. Nur aufbauend auf diese Definition hätten Beobachtungen stattfinden können. Zum Zeitpunkt der Entstehung dieser Arbeit gab es allerdings keine neuen Softwareentwickler*innen in der Einrichtung. Auch deshalb war es nicht möglich, reale Beobachtungen bei der Einarbeitung in ein neues Softwaresystem durchzuführen.

Aus diesen Gründen wurde für die vorliegende Arbeit die Methode des Interviews gewählt, mit dem Ziel, den Anwendungsfall „Einarbeitung in ein neues Softwareprojekt“ und Anforderungen für eine neue Visualisierung zu definieren. Nach Sommerville (2011) bieten Interviews einen guten allgemeinen Überblick darüber, welche Aufgaben Nutzer*innen im Moment erledigen, welche Probleme dabei gegebenenfalls auftauchen und wie sie mit dem neuen System interagieren könnten. Teilstrukturierte Interviews erlauben die Anpassung der Interviewsituation an die jeweiligen Partner*innen und sorgen gleichzeitig dafür, dass alle relevanten Themen abgearbeitet werden. Dafür dient ein Interviewleitfaden, der die wichtigsten Fragen vorgibt (Döhring & Bortz, 2016). Die Interviewerin kann Reihenfolge der Fragen während des Interviews anpassen und Nachfragen stellen, die nicht vorgesehen waren (Kaiser, 2014).

Ein solcher Interviewleitfaden wurde für die vorliegende Arbeit erstellt. Nach Rücksprache und Diskussion mit Mitarbeitenden in der Einrichtung, die hauptsächlich für Themen der Mensch-Maschine-Interaktion zuständig sind, wurde der Leitfaden nochmals überarbeitet. Der Aufbau wird im folgenden Abschnitt erläutert und die genutzte Version des Leitfadens befindet sich im Anhang.

## 4.2 Interviewleitfaden

Zur Ausgestaltung eines Interviewleitfadens wird zunächst die leitende Forschungsfrage festgelegt (Mieg & Brunner, 2001). In diesem Fall lautet diese „Welche Anforderungen liegen an eine neue Softwarevisualisierung zur Einarbeitung neuer Mitarbeitenden vor?“.

Aus dieser Forschungsfrage lassen sich mehrere Themenblöcke ableiten, die bearbeitet werden müssen (Mieg & Brunner, 2001). Eine Eröffnungsfrage leitet einen neuen Themenblock ein und sorgt für einen allgemeinen Zugang zum neuen Thema. Nach der Eröffnungsfrage gibt es je nach Themenblock eine zusätzliche Anzahl an Fragen, die in diesem Block behandelt werden sollen. Hier steht es der Interviewerin frei, weitere Nachfragen zum Gesagten (beispielsweise „Warum ist das so?“, „Wie genau kommt es dazu?“ „Weshalb empfinden Sie das so?“) zu verwenden, die der Vertiefung dienen (Mieg & Brunner, 2001). Die Themenblöcke sind so anzuordnen, dass zunächst allgemeine Themen abgefragt und im Laufe des Interviews spezifischere Aspekte besprochen werden (Kaiser, 2014).

Der hier erstellte Leitfaden besteht aus fünf Blöcken, wobei der erste Block die Eröffnung des Interviews darstellt. Es wird der berufliche Hintergrund erfragt, und, ob der/die Softwareentwickler*in

bereits mit Softwarevisualisierungen gearbeitet hat. Dadurch ist später nachvollziehbar, auf welche Erfahrungen sich die Antworten und Einschätzungen stützen. Die drei inhaltlichen Blöcke werden im Folgenden ausführlicher skizziert. Sie sollen diese Fragen beantworten: Was unterscheidet die Arbeit mit einer Visualisierung von der Arbeit mit Quellcode? Wie sieht die Einarbeitung in ein neues Softwaresystem aus? Wie müsste eine Softwarevisualisierung aussehen, damit diese den Prozess der Einarbeitung unterstützt?

**Vergleich Visualisierung und Quellcode:** In der Einrichtung ist bekannt, dass eine neue Softwarevisualisierung entwickelt wird. Um die Interviewpartner*innen inhaltlich abzuholen, wird zunächst über Softwarevisualisierungen generell gesprochen. Es wird nach Aufgaben gefragt, bei denen die eine oder andere Methode nützlicher ist und den Gründen dafür. Genaue Fragen lauten beispielsweise: *„Bei welchen Aufgaben könnte die Visualisierung geeigneter sein als der Quellcode?“* und *„Was genau ist der Vorteil gegenüber dem Quellcode?“*. Hier lässt sich die Einstellung zu diesem Medium einschätzen und erkennen, wo die Nutzer*innen selbst mögliche Anwendungsbereiche einer Softwarevisualisierung sehen.

**Einarbeitung in ein neues Softwaresystem:** Vor der Definition von Anforderungen muss sichergestellt werden, dass der Bedarf an einer Unterstützung bei der Einarbeitung gesehen wird. Durch diesen Block soll der Prozess der Einarbeitung in ein neues Softwaresystem definiert und die aktuelle Vorgehensweise der Einarbeitung ermittelt werden. Dadurch werden mögliche Aspekte festgestellt, bei denen zusätzliche Unterstützung gegeben werden kann. Zunächst wird erhoben, wie oft eine solche Einarbeitung stattfindet, um die Anwendungshäufigkeit der Visualisierung einschätzen zu können. Außerdem werden die Interviewten aufgefordert, möglichst detailliert zu erläutern, welche Schritte sie nacheinander zur Einarbeitung absolvieren und mit welchen Mitteln dies geschieht. Beispielsweise lautet eine Frage: „Beschreiben Sie so kleinschrittig wie möglich den Prozess des Einarbeitens. Welche Informationen sind wann wichtig und warum?“ Hier werden Ansatzpunkte zur Unterstützung durch die Softwarevisualisierung festgestellt. Auch der Zielzustand und die übliche Dauer der Einarbeitungsphase werden abgefragt, um den Prozess besser zu verstehen. Fragen nach Zwischenschritten (beispielsweise Pausen zur Verschriftlichung bisheriger Ergebnisse) und der Bedeutung von Teamarbeit bei der Einarbeitung zielen vor allem auf technische Eingrenzungen ab, die berücksichtigt werden müssen. Konkret wird beispielsweise gefragt: „Wenn Sie sich bisher in neue Softwareprojekte eingearbeitet haben, arbeiteten Sie in Einzelarbeit oder Teamarbeit? Welche Arbeitsweise würden Sie bevorzugen und warum?“

**Softwarevisualisierung zur Einarbeitung:** Der letzte inhaltliche Themenblock stellt die Kombination der beiden vorherigen Blöcke dar. Hier soll herausgefunden werden, ob eine Visualisierung für den Anwendungsfall für sinnvoll erachtet wird und welche konkreten Anforderungen sich ergeben. Mangelnde Benutzer*innenbeteiligung ist eine der Hauptursachen von Fehlern in Anforderungsanalysen (Avci, 2008). Deshalb wird den Nutzer*innen in diesem Block ermöglicht, eigene Anforderungen bezüglich Interaktionsmöglichkeiten, Funktionen, genutzten Metaphern oder Medien mitzuteilen. Die Interviewten werden unter anderem gefragt: „Fallen Ihnen bestimmte Interaktionsmöglichkeiten und Funktionen der Visualisierung ein, die Sie bei der Einarbeitung hilfreich fänden?“. Es ist allerdings nicht zielführend, sich nur auf Anforderungen zu beziehen, die von Nutzer*innen genannt werden. Diese sind sich den durchgeführten Prozessen nur teilweise bewusst und können oft nicht einschätzen, welche Hilfestellungen durch Technologien möglich sind (Preim & Dachselt, 2015). Deshalb werden die Anforderungen der Nutzer*innen ergänzt durch die Anforderungen, die aus der Beschreibung der bisherigen Einarbeitungsvorgehensweise abgeleitet werden.

Der fünfte und abschließende Block bietet Gelegenheit, ein Gesamtfazit abzugeben und gegebenenfalls Nachfragen oder Anmerkungen zu einem im Interview aufgekommenen Thema zu stellen.

## 4.3 Datenerhebung

Insgesamt wurden im Juni und Juli 2019 fünf Interviews in den Besprechungsräumen oder dem VR-Labor der Einrichtung für Simulations- und Softwaretechnik des DLR in Köln durchgeführt. Die Anzahl der Interviews übersteigt die allgemeine Empfehlung von mindestens zwei bis drei Interviews zur Erhebung von Anforderungen einer Nutzer*innengruppe (Preim & Dachselt, 2015). Die Interviewpartner*innen haben sich vor unterschiedlich langer Zeit zuletzt in ein Softwaresystem eingearbeitet, weshalb die Befragung von mehr als drei Personen weitere Erkenntnisse versprach. Die Interviews dauerten im Durchschnitt 45 Minuten. Wie von Nielsen (1993) beschrieben, dauerten die ersten Interviews länger, da in späteren Interviews weniger neue Erkenntnisse hinzukamen. Interviewt wurden vier Softwareentwickler und eine Softwareentwicklerin, im Alter von 28 bis 41 Jahren, die seit einem halben bis 14 Jahren beim Deutschen Zentrum für Luft- und Raumfahrt arbeiten. Es wurde versucht, Softwareentwickler*innen für die Interviews zu gewinnen, die die vorhandene Visualisierung über eine Inselmetapher nicht im Detail kannten. Somit wurde sichergestellt, dass sich ihre Einschätzungen und Anforderungen nicht zu sehr darauf beziehen, was sie bereits gesehen haben. Zumindest im Gespräch mit Kolleg*innen und in Veröffentlichungen haben alle Interviewten die Inselmetapher-Visualisierung bereits gesehen, allerdings arbeitete niemand von ihnen aktiv an dieser Visualisierung mit. Alle Interviews wurden von der Autorin selbst durchgeführt. Zu Beginn wurden die Interviewten erneut über die Ziele des Interviews und der damit verbundenen Abschlussarbeit informiert. Da keine zusätzliche Person zur Protokollierung des Interviews anwesend war, wurden Tonaufnahmen der Interviews erstellt. Dies diente der Sicherstellung eines flüssigen Gesprächsablaufs und gleichzeitig der Vollständigkeit aller gegebenen Informationen. Den Interviewten wurde zugesichert, dass die Aufnahmen nach der anonymisierten Transkription gelöscht werden. Durch einen sehr ausführlichen Interviewleitfaden wurde sichergestellt, dass trotz fehlender Interviewerfahrung alle wichtigen Aspekte besprochen wurden. Im Rahmen der Interviews wurde allerdings häufiger von diesem Leitfaden abgewichen, beispielsweise wenn die Befragten schon früh den Bedarf einer Visualisierung infrage stellten und somit Fragen zu konkreten Anforderungen an eine Visualisierung nicht zielführend waren. Obwohl der Leitfaden in der Höflichkeitsform geschrieben wurde, werden die Interviewten von der Interviewerin wie im Arbeitsalltag geduzt.

## 4.4 Transkription und Auswertung

Die Tonaufnahme diente der vereinfachten Erfassung aller wichtigen Informationen, wurde anonymisiert niedergeschrieben und nach der Transkription gelöscht. Für die Transkription von Interviews gibt es verschiedene Ansätze und Regelwerke (u.a. Dresing & Pehl, 2018; Hagemann & Henle, 2014). Die Interviews der vorliegenden Arbeit werden mit hoher Wahrscheinlichkeit über die Festlegung von Anforderungen hinaus keiner wissenschaftlichen Verwendung dienen. Für die Definition der Anforderungen wäre es auch ausreichend gewesen, die Interviews mit einer zusätzlichen protokollführenden Person durchzuführen, um Hauptpunkte zu notieren. Dies war aus logistischen Gründen nicht möglich, weshalb Tonaufnahmen gemacht wurden. Deshalb ist es hier ausreichend, sich an den vereinfachten Transkriptionsregeln nach Dresing und Pehl (2018) zu orientieren. Die Transkripte wurden zur Förderung des Leseflusses und Erleichterung der inhaltlichen Auswertung geglättet. Das heißt, Doppelungen wurde gestrichen, genauso wie zustimmende Aussagen (beispielsweise „ja“, „mhm“,...) der Interviewerin während des Redeflusses des Interviewten. Außerdem wurden Sätze in manchen Fällen grammatikalisch angepasst, sodass das Lesen vereinfacht wird. Darüber hinaus wurde wörtlich transkribiert.

Die Auswertung der Interviews folgte den Ansätzen der qualitativen Inhaltsanalyse nach Mayring

(2010). Die eigentliche Analyse findet nach Bestimmung des Materials und der Spezifikation der Fragestellung statt. In diesem Fall sind diese Punkte durch die Interviews und die für den Leitfaden bereits definierte Leitfrage festgelegt. Es gibt drei Grundformen der qualitativen Inhaltsanalyse: Zusammenfassung, Extrahierung und Strukturierung (Mayring, 2010). Die zusammenfassende Inhaltsanalyse reduziert das vorliegende Material und schafft durch Abstraktionen eine Kurzform des Grundmaterials. Bei der Extrahierung werden bedeutende Textteile mit zusätzlichem Material angereichert, um das Verständnis zu erweitern. Die strukturierende Inhaltsanalyse hat das Ziel, einen Querschnitt des Materials abzubilden, indem Inhalte nach Kriterien sortiert und eingeordnet werden (Mayring, 2010). Da das Ziel der vorliegenden Analyse ist, bestimmte Aspekte zu ordnen und einzuschätzen, wird eine strukturierende Inhaltsanalyse nach Mayring (2010) angewandt. Als Unterform wird die inhaltliche Strukturierung gewählt, deren Ziel es ist, „bestimmte Themen, Inhalte, Aspekte aus dem Material herauszufiltern und zusammenzufassen“ (Mayring, 2010, S.98). Um festzulegen, welche Inhalte herausgefiltert werden, sind zuvor Kategorien zu bestimmen. Die hier verwendeten Kategorien lassen sich anhand des Interviewleitfadens erkennen: Definition der Einarbeitung; Bedarf der Unterstützung; Anforderungen an Softwarevisualisierung. Innerhalb der Hauptkategorien lassen sich in den Aussagen der Interviewpartner*innen induktiv Unterkategorien finden, die die Kategorien nach Themengebieten gliedern. Zu beachten ist, dass vergleichbar mit anderen Anforderungsanalysen (z.B. Zimmermann, Konrad & Nerdinger, 2009) die Erarbeitung eines Kategoriensystems für die vorliegende Studie eine weniger bedeutende Rolle spielt als von Mayring (2010) vorgesehen. Die Kategorien beschreiben in diesem Fall nicht selbst ein Phänomen sondern dienen der Gliederung der Aussagen. Nach der Bearbeitung des Textes mittels des Kategoriensystems werden die einzelnen kategorisierten Abschnitte paraphrasiert und pro Unterkategorie und Hauptkategorie zusammengefasst. Dies geschieht in zwei Reduktionsschritten. Zunächst werden sowohl nicht inhaltstragende als auch deutungsgleiche Paraphrasen gestrichen. In einem zweiten Schritt werden Phrasen gebündelt, die sich auf denselben Gegenstand beziehen (Mayring, 2010). Somit finden sich am Ende dieser Anforderungsanalyse jeweils die zusammenfassenden Einschätzungen der Interviewten zu den einzelnen Unterkategorien. Zitate aus den Interviews dienen der Untermalung dieser Zusammenfassungen.

# 5 Ergebnisse

Die im Rahmen der Interviews erhobenen Aussagen wurden in drei Hauptkategorien eingeteilt: Einarbeitung in neues Softwaresystem, Bedarf an Unterstützung im Einarbeitungsprozess und konkrete Anforderungen an eine Softwarevisualisierung zur Unterstützung der Einarbeitung. Die zu Unterkategorien gebündelten Aussagen werden in diesem Kapitel vorgestellt. Aus den Aussagen zur Definition der Einarbeitung und zum Bedarf an Unterstützung werden die von der Autorin abgeleiteten Anforderungen eine Softwarevisualisierung direkt in den jeweiligen Unterkategorien präsentiert. Die von den Interviewten konkret und direkt formulierten Anforderungen in der dritten Kategorie werden in Verbindung mit den abgeleiteten Anforderungen gesetzt und können diese bekräftigen, erweitern, ergänzen oder einschränken. Anforderungen, die in späteren Teilen des Kapitels bekräftigt werden, sind erneut aufgeführt und mit dem Zusatz „bestätigt“ gekennzeichnet. Anforderungen, die modifiziert werden, sind mit einem kleinen Buchstaben (beispielsweise Anforderung 1a) versehen und die Änderungen sind durch kursive Schrift gekennzeichnet.

## 5.1 Einarbeitung in neues Softwaresystem

Die Aussagen zur Beschreibung der Einarbeitung lassen sich in sieben Gruppen gliedern. Es wird die zeitliche Dimension, also Häufigkeit der Einarbeitung, der genaue Ablauf und die abgeschlossene Einarbeitung thematisiert. Danach wird die praktische Dimension anhand folgender Unterpunkte erläutert: zusätzliche Unterstützungsinstrumente, Notwendigkeit von Notizen und Pausen, Bedeutung der Softwarehistorie für die Einarbeitung und Teamarbeitsaspekt.

### 5.1.1 Zeitliche Dimension

**a) Häufigkeit:** Für die Arbeit als Softwareentwickler*in ist eine Einarbeitung in ein neues Softwareprojekt immer beim Wechsel in eine neue Abteilung nötig. Gerade im DLR ist normalerweise nicht vorgesehen, zwischen Softwareprojekten zu wechseln, es sei denn, man bekundet selbst starkes Interesse. Somit wird der gesamte Einarbeitungsprozess in ein neues Softwareprojekt von neuen Mitarbeitenden nur einmal durchlaufen. Die Visualisierung zur Einarbeitung wird demnach nur zu Beginn der Arbeit in der Abteilung genutzt und nur bis zur Beendigung des Einarbeitungsprozesses.

**Anforderung 1:** Einfache Bedienbarkeit und intuitive Nutzbarkeit ohne große Vorbereitung bieten.

**b) Ablauf:** Bevor sich neue Mitarbeitende mit der Software als solche beschäftigen, liegt der erste Schritt der Einarbeitung im Kennenlernen des gesamten Projektes. Dies schließt sowohl den Anwendungsfall der Software ein, als auch das Wissen, wer für welche Teile des Projektes verantwortlich ist. Die Aussage *„Je komplexer schon der Anwendungsfall ist, den man verstehen muss, desto schwieriger ist letztendlich auch die Einarbeitung in die Software, weil die Software ist ja nur eine Umsetzung der Anforderungen an die Software“* verdeutlicht die Bedeutung des Verständnisses der Anwendung. Die zitierte Person betont darüber hinaus, dass im besten Fall reale Anwender*innen bei der Interaktion beobachtet werden, sofern die Software bereits in Nutzung ist. Dies führt zum besseren Erkennen, wie mit der Software letztendlich umgegangen werden soll. Zusätzlich zur Anwendung der Software beschäftigen sich neue Mitarbeitende als erstes damit, herauszufinden, wo Dokumentationen zu finden sind und wer sich für welchen Teil des Softwareprojekts verantwortlich fühlt. In der Abteilung sind Schwerpunkte und Inhalte nicht von äußeren Strukturen vorgegeben, sondern können nach Interesse gewählt werden. Für neue

Mitarbeitende ist bedeutend, herauszufinden, wer die jeweils passende Ansprechperson darstellt. *„Du musst das Projekt und wer an was arbeitet verstehen. Vorher kannst du nicht an der Software arbeiten. Also nicht im komplexen Sinne, das geht nicht.“*

Der zweite Schritt der Einarbeitung liegt darin, die Softwareumgebung aufzusetzen, eventuell benötigte Zusatzsoftware zu verstehen und erste Tests auszuführen, die bestätigen, dass alles richtig installiert wurde. *„Ich muss auch erst mal diese Entwicklungsumgebung aufsetzen. Das ist ja auch nicht so trivial, da lernt man auch schon eine Menge, wenn das nicht gut dokumentiert ist.“* Die Software RCE, die zunächst Grundlage für die zu entwickelnde Visualisierung bieten soll, hat eine graphische Benutzungsoberfläche (GUI). Wenn eine solche GUI vorliegt, besteht der nächste Schritt in der Einarbeitung in diese. Dies hängt auch direkt mit der Einarbeitung in den Anwendungsfall zusammen, da durch eigene Nutzung nochmals verdeutlicht wird, wie die Software am Ende verwendet wird.

Wenn sich die neuen Mitarbeitenden mit den Prozessen rund um die Software und ihrer Anwendung beschäftigt haben, beginnt der Einstieg in die Arbeit mit dem Code. Es ist wichtig zu betonen, dass die Prozesse der Einarbeitung hier sukzessive dargestellt werden, jedoch zu Teilen während der Einarbeitungsphase auch noch parallel passieren. Das heißt, dass zwar zunächst der Anwendungsfall betrachtet wird, jedoch auch während der Arbeit mit dem Code noch einiges über die spätere Anwendung gelernt werden kann. Die Softwarestruktur wird ohne konkrete Aufgabe nur sehr grob und nicht auf detailliertem Level betrachtet. Idealerweise liegen einzelne Softwaretests bereits vor, die ausgeführt werden und anhand derer man die Datenflüsse rekonstruieren kann. Wenn keine dieser Tests vorliegen oder nach deren Betrachtung erfolgt die Einarbeitung anhand der Bearbeitung einer ersten Aufgabe. Im Bugtracker des Teams sind dabei für Einsteiger*innen gekennzeichnete Aufgaben zu finden. Diese sind klein genug, sodass sie programmspezifisch lösbar sind, auch wenn der Anwendungsfall noch nicht in allen Details verstanden wurde. Ein häufig genanntes Beispiel für eine solche Aufgabe war die Korrektur eines Rechtschreibfehlers bei einer textuellen Ausgabe in der GUI. Beim Lösen einer solchen Aufgabe, auch Issue genannt, wird das Vorgehen wie folgt beschrieben: zunächst soll diese Issue nachvollzogen werden, das heißt im Fall des Rechtschreibfehlers wird zunächst gesucht, ob dieser Rechtschreibfehler an der vorgegebenen Stelle auftritt. Um diesen dann zu beheben, muss die richtige Stelle im Code gefunden und der Fehler behoben werden. Dabei wird über die aktuell bearbeitete Codestelle immer *„nach links und rechts geschaut“*, um den Kontext, in dem man gerade in der Software arbeitet, zu erfassen. Besonders bei den ersten Aufgaben wird beschrieben, dass das Finden der richtigen Stelle im Code zur größten Überforderung führt und oft nur mit der Unterstützung erfahrener Kolleg*innen möglich wird. Nachdem die erste Aufgabe bearbeitet wurde, folgen in der gleichen Vorgehensweise mehrere kleine Aufgaben, die gelöst werden. Somit beginnen neue Mitarbeitende ihre Arbeit an mehreren konkreten Stellen im Kleinen und entwickeln so nach und nach ein mentales Modell des Gesamtbildes. *„Ich komme eigentlich immer eher vom Quellcode her und versuche von einem bestimmten Punkt aus Abhängigkeiten zu erkennen und mich vom Quellcode aus auf die Vogelperspektive herauszubewegen.“*

**Anforderung 2:** Informationen aus den ersten Einarbeitungsprozessen liefern: Anwendungsfall, zuständige Kolleg*innen, Informationen zum Aufsetzen der Entwicklungsumgebung.

**Anforderung 3:** Beim Finden der zu bearbeitenden Codestelle Hilfestellung leisten. Für die ersten Aufgaben die passende Stelle in Visualisierung markieren.

**Anforderung 4:** Softwarestruktur darstellen, spezifizierbar auf jeweilig gelöste Aufgaben: Umgebung der bearbeiteten Codestelle anzeigen.

**c) Abgeschlossene Einarbeitung:** Die Einarbeitungszeit in ein Projekt wie RCE kann nicht genau bestimmt werden, allerdings wurde mehrfach angesprochen, dass es sich dabei um *„keine Sache von*

*Tagen oder Wochen“* handele. Eine Zeitspanne von einem halben Jahr wurde von einer befragten Person angegeben, bis diese sich sicherer im Code gefühlt habe und eigenständig erste größere Aufgaben lösen konnte. Das wiederholte Lösen der vorgegebenen Aufgaben kann auch als wichtiger Punkt gesehen werden, an dem die grundlegende Einarbeitungsphase abgeschlossen ist. Allerdings ist wichtig zu betonen, dass alle Interviewten angaben, dass eine vollumfängliche Einarbeitung in jede Stelle des Codes nie beendet sein wird. *„Gerade im konkreten Fall RCE würde ich sagen, man wird nie fertig. Ich habe letztens noch mit einem Kollegen gesprochen, der ist über sieben Jahre dabei und ich hatte ihn nach einer konkreten Codestelle gefragt und er meinte, die hätte er auch noch nie gesehen. Wüsste also auch nicht, wie das alles zusammenhängt.“* Es kann also in der ersten Einarbeitung nicht darum gehen, die Software komplett zu kennen und zu verstehen, sondern vielmehr darum zu wissen, wo die wichtigen Informationen liegen und an wen man sich mit Fragen wenden kann. Unterschiedlich gewählte Schwerpunkte führen zusätzlich dazu, dass die Expertise sich auf einen Teil der Software beschränkt und bei der Bearbeitung anderer Teile eine erneute Einarbeitung in diese Codestelle erfolgen muss. Allerdings ist diese Einarbeitung in eine neue Codestelle dann bedeutend schneller, da grundlegende Prozesse bekannt sind.

Eine einheitliche Definition des Zielzustands der Einarbeitung kann nicht gegeben werden, da eine Einarbeitung in ein großes Softwareprojekt nie komplett abgeschlossen ist. Für eine mögliche Visualisierung lässt sich die Einarbeitung allerdings in zwei verschiedene Phasen gliedern.

Dabei beinhaltet die erste Phase der Einarbeitung vor allem auch Prozesse, die um die Software herum passieren und dauert bis zur ersten Aufgabe, die vorgegeben wird. In der zweiten Phase der Einarbeitung, die immer wieder auftreten kann, soll die Visualisierung bei der Bearbeitung von Aufgaben unterstützen. Das Verständnis unbekannter Codestellen muss immer wieder neu erarbeitet werden. Informationen zur ersten Phase der Einarbeitung würden bei der späteren Einarbeitung in neue Codestellen keinen Nutzen bringen und sollten zur Reduktion der dargebotenen Informationsdichte weggelassen werden.

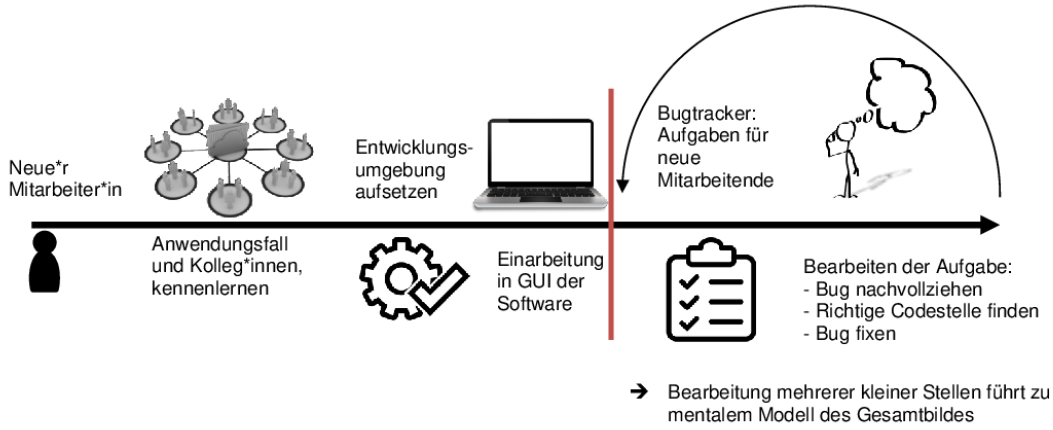



Abbildung 5.1: Vereinfachte lineare Darstellung des Einarbeitungsprozesses in zwei Phasen gegliedert (Eigene Darstellung)

**Anforderung 5:** Beide Einarbeitungsphasen unterstützen durch

**5.1.** Den Teil der Visualisierung ausblenden bzw. deaktivieren, der bei der ersten Phase zusätzlich unterstützen soll, oder

**5.2.** verschiedene Visualisierungen für die jeweilige Phase entwickeln.

In Bezug auf die Häufigkeit der Einarbeitung muss an dieser Stelle hinzugefügt werden, dass die Visualisierung der zweiten Phase mehrfach zur Anwendung kommt. Damit wird die Anforderung 1 modifiziert,

da die unkomplizierte, intuitive Bedienbarkeit in der zweiten Einarbeitungsphase an Bedeutung verliert, zugunsten zusätzlicher Funktionen und Interaktionsmöglichkeiten.

**Anforderung 1a:** Einfache Bedienbarkeit und intuitive Nutzbarkeit bieten.

**1.1** *Für erste Einarbeitungsphase:* ohne große Vorbereitung nutzbar machen.

**1.2.** *Für zweite Einarbeitungsphase: ausreichende Erklärung und Einführung bieten, zur wiederholten Nutzung mit mehr Funktionen.*

### 5.1.2 Praktische Dimension

**a) Zusätzliche Unterstützungsinstrumente:** Für die Softwareentwickler*innen gibt es neben dem Quellcode zusätzliche Unterstützungsinstrumente, die die Einarbeitung und die Arbeit mit der Software erleichtern sollen. Im DLR lassen sich zusätzliche Informationen im Wiki und im Developer Guide finden. Ersteres beschreibt hauptsächlich organisatorische Prozesse und Roadmap Planning, während der Developer Guide die Softwareumgebung und Prozesse innerhalb der Software beschreibt sowie beim Aufsetzen der Entwicklungsumgebung (IDE) unterstützt. In der alltäglichen Arbeit wird dieser nur noch selten zu Rate gezogen. Die Visualisierung kann neue Softwareentwickler*innen darin unterstützen, zu verstehen, wo jeweils notwendige Informationen zu finden sind. Die gegenseitige Ergänzung dieser drei Hilfsmittel und der Verweis an die richtigen Stellen sind grundlegend für eine Erleichterung der Einarbeitung.

**Anforderung 6:** Visualisierung mit beiden Informationsquellen (Developer Guide und Wiki) verbinden, mit jeweiligen Verweisen, wo weitere benötigte Informationen liegen.

**b) Historie:** Für die Einarbeitung in eine Software ist zunächst der Status quo relevant, da der aktuelle Zustand der Software die Grundlage für weitere Entwicklungen darstellt. Die Historie des Softwaresystems, also wie sich der Code und die Struktur der Software verändert haben, wird erst vereinzelt für konzeptionelle Aufgaben genutzt. Diese werden im Einarbeitungsprozess meist erst bearbeitet, nachdem mehrere Fehlerbehebungsaufgaben bearbeitet wurden. Dabei gaben einige Interviewte an, die Historie zur Nachvollziehbarkeit von Entscheidungen zu nutzen und um Intentionen der Kolleg*innen in einzelnen Programmiervorgehensweisen zu erkennen. Der bedeutendere Aspekt der Historie ist also viel mehr die Frage, wer eine bestimmte Stelle aus welchen Gründen verändert hat, als die Frage, wie sich diese Stelle verändert hat. Ein generelles, dauerhaftes Anzeigen der Historie ist demnach wenig sinnvoll, da die Historie nur in späteren Aufgaben von den Entwickler*innen genutzt wird.

**Anforderung 7:** Information liefern, wer welchen Teil des Codes bearbeitet hat und mit welchem Ziel dieser bearbeitet wurde.

**c) Teamarbeit:** Die gemeinsame Einarbeitung neuer Mitarbeitenden ist nur begrenzt möglich und sinnvoll, da unterschiedliche Tempi, Interessen und Herangehensweisen vorliegen. Am Code sind die Entwickler*innen *„Einzelkämpfer“*, da durch aktives *„Selbstmachen“* am meisten gelernt wird. Trotzdem spielt Teamarbeit eine enorm große Rolle im Einarbeitungsprozess, in dem Sinne, dass viel mit erfahrenen Kolleg*innen gesprochen wird und Fragen gestellt werden. Ansprechpartner*innen stehen dabei zur Verfügung und werden auch häufig befragt, wobei hierfür meist mehrere Fragen gesammelt werden. *„Sich in große Systeme einarbeiten ohne die Kollegen, das geht nicht“*. Obwohl in dieser Abteilung keine offiziellen Spezialgebiete vergeben werden, spezialisieren sich Entwickler*innen selbst. Wer sich für welches Gebiet verantwortlich fühlt, wird durch Gespräche unter den Kolleg*innen herausgefunden.

Insgesamt ist im Rahmen der fünf Interviews die Rolle der Kolleg*innen verdeutlicht worden. Das Wissen, wer einem in welchem Fall weiterhelfen kann, gehört zu den grundlegendsten und bedeutendsten

Punkten.

**Anforderung 8:** Informationen liefern, wie das Wissen über bestimmte Aspekte verteilt ist und wer die direkte Ansprechperson für eine bestimmte Frage ist.

**d) Notizen:** Das Pausieren der Einarbeitung, um Notizen anzufertigen, ist nicht bei allen Entwickler*innen Standard, sondern sehr individuell. Eine befragte Person gab an, sehr exzessiv die Möglichkeit zu nutzen, sich etwas zu notieren oder Diagramme zu zeichnen, während andere über eine kleine Übersicht zu Beginn hinaus keine weiteren Stichworte notieren. Generell ist das Ziel immer, möglichst viele Informationen in der vorhandenen Dokumentation nachzulesen.

Die Frage nach dem Pausieren und Notieren diente vor allem der Erkenntnis, welches Medium für eine Visualisierung verwendet werden soll. Die Möglichkeit des Pausierens, um sich etwas zu notieren, ist bei einer Visualisierung in VR eingeschränkt, da dies ein ständiges Auf- und Absetzen der Brille mit sich bringen würde. Es ist nicht bei allen Entwickler*innen üblich, dass jeder kleinteilige Schritt gesondert niedergeschrieben wird, weshalb eine Visualisierung in VR an dieser Stelle nicht ausgeschlossen wird.

**Anforderung 9:** (für Visualisierung mit VR): Notizzettel in Visualisierung einfügen, um Stichworte mitschreiben zu können, ohne die Visualisierung zu unterbrechen.

## 5.2 Bedarf an Unterstützung im Einarbeitungsprozess

Beim Kodieren der Aussagen zum Bedarf an Unterstützung im Einarbeitungsprozess wurde schnell deutlich, dass viele Aufgaben noch erleichtert werden können. Gleichzeitig wurde viel über den eingeschätzten Nutzen von Visualisierungen als Unterstützung gesprochen. Diese beiden Themen werden getrennt voneinander betrachtet und dargestellt. Die Aussagen zur Erleichterung von Aufgaben beziehen sich auf die Struktur, das Wording, die Benutzungsoberfläche, die Historie, den Aspekt der Teamarbeit und die Lokalisierung von Problemstellen. Bezüglich der Visualisierungsmethode werden Aussagen in die drei Teile Skepsis gegenüber Visualisierungen, positive Komponenten und Kombination mit IDE gegliedert.

### 5.2.1 Erleichterung von Aufgaben

**a) Struktur:** Wie bereits beschrieben, ist die Einarbeitung in RCE von hohem zeitlichem Aufwand und nicht in Tagen oder Wochen abgeschlossen. Eine Beschleunigung dieses Prozesses setzt ein schnelleres Verständnis voraus: *„Wenn ich jemand Neuem direkt eine etwas größere Aufgabe geben möchte, dann müsste der natürlich schneller die Zusammenhänge, die Architektur verstehen.“* Gerade bei großen Softwareprojekten liegt die Herausforderung darin, zu erkennen, wie einzelne Bestandteile miteinander zusammenhängen. Gleichzeitig wurden in den Interviews Zweifel geäußert, dass diese Zusammenhänge in sehr kurzer Zeit überhaupt erfasst werden können. Dafür ist mehr nötig, als die Architektur der Software zu kennen. Spätestens, wenn die Codezeilen gefunden wurden, die verändert werden müssen, ist es zweitrangig, die Umgebungsstruktur im Detail zu kennen. Die Entwicklungsumgebung bietet nämlich hier die Möglichkeit, sich anzeigen zu lassen, welche Funktionen aufgerufen werden. Auch zur Navigation durch den Quellcode bietet die IDE bereits Unterstützung. Das heißt, große Teile des notwendigen Strukturverständnisses werden durch die IDE bereits übernommen. Dennoch wurde in einem Interview auch die Grenze der Darstellung von Textzeilen auf einem Bildschirm betont. Es können etwa 50 Zeilen gleichzeitig dargestellt werden, was dazu führt, dass der Fokus immer auf der aktuell vorliegenden Codestelle liegt und ein mentales Modell davon, in welchem Kontext man sich gerade befindet, im Hinterkopf behalten werden muss. Zusätzliche Unterstützung hierbei bieten vereinzelt vorliegende Diagramme, die allerdings

nie die gesamte Architektur aufzeigen, meist per Hand gezeichnet wurden und nicht auf dem aktuellen Stand sind.

All diese angemerkten Punkte zeigen auf, dass über den Bedarf an Unterstützung keine einheitliche Meinung der Interviewten vorliegt. Es wurde für das Verständnis der Struktur oftmals die Qualität der IDE betont. Gleichzeitig wird deutlich, dass das Verständnis des gesamten Aufbaus der Software nirgends abgebildet wird. Der Aufbau muss von denjenigen, die diesen für die Bearbeitung der aktuellen Programmzeilen benötigen, in einem mentalen Modell im Hinterkopf behalten werden. Eine Visualisierung könnte also hier Abhilfe schaffen, indem eine Übersicht dargestellt wird, auf die bei Bedarf zurückgegriffen werden kann.

**Anforderung 4a:** *Aktuelle und automatisch generierte Übersicht der* Softwarestruktur *mit gegenseitigen Abhängigkeiten und Services* anzeigen.

**4.1.** Spezifizierbar auf jeweilig gelöste Aufgaben: Umgebung *und Abhängigkeiten* der bearbeiteten Codestelle anzeigen.

**b) Wording:** Der Quellcode der Software RCE ist dadurch besonders komplex zu erfassen, da viele Leute daran arbeiten und gearbeitet haben. Deshalb entsteht auch das Problem, dass keine einheitliche Namenskonvention vorliegt und ein Einstieg ins Wording hilfreich wäre.

**Anforderung 10:** Informationen und Hintergründe liefern zur Benennung einzelner Bestandteile der Software und dort, wo durch den Namen keinerlei Inhalt erkannt werden kann, genauere Inhaltsbeschreibungen liefern.

**c) GUI:** In einem Interview wurde verdeutlicht, dass die Benutzungsoberfläche mit ihren Funktionen und Möglichkeiten nicht ausführlich genug erläutert wird. Da die GUI allerdings das Ergebnis der bearbeiteten Software ist und für die Anwender*innen, die nicht die Softwareentwickler*innen selbst sind, einfach handhabbar sein soll, ist dieser Punkt nicht in die Entwicklung einer Visualisierung zur Einarbeitung aufzunehmen. In der Abteilung im DLR arbeiten bereits Kolleg*innen daran, Nutzer*innen der Software das Verständnis der GUI zu erleichtern, was dann auch bei der Einarbeitung in die GUI für die Entwickler*innen helfen wird.

**d) Historie:** Bei jeder Änderung des Codes wird im Rahmen einer Commit-Message auch immer die Issue-ID aus dem Bugtracker angegeben. Das heißt, die Änderungen und somit die Historie sind über die Commits angezeigt und nachvollziehbar. Eine interviewte Person stellte allerdings heraus, dass es notwendig ist, selbst ein Skript zu bauen, um die Historie zu erkennen. Beide genannten Vorgehensweisen sind mit einem hohen zeitlichen Aufwand verbunden.

**Anforderung 7a:** Information liefern, wer welchen Teil des Codes bearbeitet hat, mit welchem Ziel dieser bearbeitet wurde *und mit welcher Issue-ID diese Änderung verbunden ist.*

**e) Teamarbeit:** Spezialgebiete der Entwickler*innen sind in der Abteilung bekannt, beziehungsweise können erfragt werden. Außerdem stehen am Code jeweils die Autor*innen der jeweiligen Stellen. Allerdings herrscht, bedingt durch die wissenschaftliche Arbeit, eine hohe Fluktuation im Team, weshalb bei dem wachsenden Code eine Vielzahl an Autor*innen an einer Stelle stehen und einige davon gar nicht mehr Teil des Teams sind. Damit ist wieder fraglich, an wen man sich wenden kann.

**Anforderung 8a:** Informationen liefern, wie das Wissen über bestimmte Aspekte verteilt ist und wer die *aktuelle,* direkte Ansprechperson für eine bestimmte Frage ist.

**f) Lokalisierung Problem:** Dieser Abschnitt umfasst eine bestimmte Art der Aufgabenlösung, die im zweiten Teil der Einarbeitung vorkommen kann. Es wurde mehrfach erwähnt, dass das Finden der richtigen Stelle im Code viel Zeit bei der Problembehebung beansprucht. Zur Ausführungszeit muss beispielsweise im Code ein Workflow nachvollzogen werden, um zu verstehen, wo genau ein Problem vorliegt. Eine übersichtliche Darstellung des Datenflusses würde dabei bei der Lokalisierung helfen. Außerdem ist bei der Änderung einer bestimmten Stelle abzuschätzen, welche Bestandteile zusätzlich angepasst werden müssen. Für die Visualisierung in der zweiten Phase der Einarbeitung ist relevant, dass ein spezifischer Datenfluss nachgebildet werden kann.

**Anforderung 11:** Strukturteile hervorheben oder dynamisch darstellen, die für den aktuellen Fall relevant sind.

**Anforderung 12:** Stelle hervorheben, an der ein Datenfluss nicht korrekt ausgeführt werden kann.

### 5.2.2 Einstellung zu Visualisierungen

Da die Arbeit an einer neuen Visualisierung den Interviewten bereits bekannt war, ging es natürlich auch um die Frage, ob und inwieweit eine Visualisierung zur Einarbeitung als sinnvoll erachtet wird. Dabei wurden sowohl Überzeugung als auch Skepsis geäußert. Wichtig ist zu betonen, dass die Interviewten keine auf den Anwendungsfall der Einarbeitung bezogene Visualisierung kannten. Allerdings kannten alle Teile der aktuell vorliegenden Visualisierung, welche nicht explizit für die Unterstützung von Softwareentwickler*innen entwickelt wurde. Vor allem mit Hinblick auf bekannte Visualisierungen herrschte eine große Skepsis der Befragten vor, da diese nicht ausgereift seien, um sie in ihrer Arbeit zu unterstützen. *„Das ist etwas, dass ich Status jetzt sagen würde, das würde mich Null weiterbringen.“*, *„Bisherige Visualisierungen sind zwar ein nettes Spielzeug, bei dem man sich Demos zeigen lassen kann aber die praktische Relevanz ist noch nicht ausgereift.“* Es ist an dieser Stelle wichtig zu betonen, dass die Haltung gegenüber zukünftigen Visualisierungen von dem bisher Bekannten beeinflusst sein kann.

**a) Skepsis Visualisierung:** In Bezug auf die Software RCE, die aus sehr vielen Bundles und Funktionen besteht, wurde von einigen Interviewten Skepsis geäußert, dass eine Visualisierung in der Lage ist, dies darzustellen und gleichzeitig die Komplexität zu vereinfachen. *“Ich habe ja schon von den Bundles gesprochen. Das sind bei RCE jetzt schon relativ viele, so an die 220 oder so, da weiß ich nicht, ob eine Visualisierung auf einer Architekturebene, die Bundles und Abhängigkeiten zeigen würde, da wirklich viel helfen wird.“* Die reine Arbeit mit dem Textinterface wird von den Entwickler*innen dabei oft als schneller eingeschätzt. Sie sind es gewohnt, mit Text und Quellcode zu arbeiten und empfinden den Mehrwert gegenüber von Zahlen und textuellen Ausgaben als fraglich: *„Aber also ich persönlich, stelle mir dann die Frage, was bietet mir diese Visualisierung mehr, als eine Zahl?“* Diagramme, die zum Beispiel im Developer Guide vorliegen können, werden als sehr hilfreich angesehen, um beispielsweise Datenmodelle zu verstehen oder in der konzeptionellen Phase, Prozesse zu planen. Diese Diagramme zu einer größeren, umfassenden Visualisierung umzugestalten wird als nicht zielführend angesehen. *„Ich glaube aber auch da dann halt als Diagramm vollkommen ausreichend.“*

**b) Positive Komponenten:** Obwohl viele skeptisch gesehene Aspekte angesprochen wurden, berichteten die Interviewten auch von positiven Erfahrungen bei der Arbeit mit Visualisierungen. So half eine Laufzeitvisualisierung bereits in einem anderen Projekt bei der Suche nach einem Anhaltspunkt der Programmoptimierung. Auch Hotspotanalysen und Knowledge Maps, die aufzeigen, wie Wissen innerhalb eines Teams verteilt ist, wurden genutzt und für gut befunden. Gerade für die Kommunikation wird

Potential in einer Visualisierung gesehen, da so sichergestellt werden kann, dass alle Beteiligten das gleiche Verständnis über das Gesprochene haben. Außerdem wird vorteilhaft eingeschätzt, dass Strukturen aufgezeigt werden können, die im Quellcode nicht so einfach zu erkennen sind und eine schnellere Übersicht ermöglichen. Testausführungen in der Visualisierung im Gegensatz zu mehreren Textdateien, die sich bei einer Testdurchführung öffnen, bieten weiteres Potential der Unterstützung.

**Anforderung 8a bestätigt:** Informationen liefern, wie das Wissen über bestimmte Aspekte verteilt ist und wer die aktuelle, direkte Ansprechperson für eine bestimmte Frage ist.

**Anforderung 13:** Informationen liefern, welche Teile des Codes lange nicht verändert wurden bzw. besonders häufig verändert wurden.

**Anforderung 14:** Datenfluss bei Testdurchführungen innerhalb der Visualisierung darstellen.

**c) Kombination mit IDE:** Insgesamt wird in den Interviews betont, dass das Mittel der Bearbeitung am Ende immer die bereits sehr gut gestaltete und unterstützende IDE ist. Allerdings kann die Visualisierung eine Übersicht zur Einarbeitung bieten und für die Lokalisierung und Abschätzung von Folgen herangezogen werden. Danach würden die Entwickler*innen allerdings in die IDE wechseln, da hier schneller gearbeitet werden kann. Außerdem wurde kritisiert, dass bisherige Visualisierungen meist mit der Vogelperspektive starten und dann die Möglichkeit bieten, weiter in die Struktur zu zoomen. Allerdings ist dies oftmals nicht so weit möglich, wie es am Ende benötigt wird, um alles zu verstehen, da die unterste Ebene der Visualisierung noch nicht die erforderlichen Informationen bereitstellt. Hier wird deutlich, dass die Visualisierung vor allem die Arbeit in der IDE unterstützen soll, was einen engen Zusammenhang der beiden Instrumente voraussetzt und bestenfalls die direkte Verbindung ermöglicht, sodass von der Visualisierung ohne großen Aufwand in die IDE umgeschaltet werden kann.

**Anforderung 15:** IDE und Visualisierung direkt verbinden.

## 5.3 Konkret geäußerte Anforderungen

Es wurden durch die Beschreibung des Einarbeitungsprozesses und des Bedarfs an Unterstützung bereits einige Anforderungen abgeleitet. Im Rahmen der Interviews wurden zusätzlich einige Anforderungen von den Softwareentwickler*innen konkret formuliert. Diese bestätigen zu großen Teilen die abgeleiteten Anforderungen oder spezifizieren und erweitern diese. Einige neue Aspekte wurden angesprochen, wodurch zusätzliche Anforderungen ergänzt wurden. Insgesamt lassen sich sieben Untergruppen an Anforderungen feststellen die den Kontext, die Struktur, verfügbare Funktionen, die Verknüpfung von Code und Visualisierung, die Verwendung von Metaphern, das Medium und die Anwendung der Visualisierung betreffen.

**a) Kontext:** Im Gegensatz zu bisher bekannten Visualisierungen soll nicht nur die Struktur und Architektur angezeigt werden können. Bei der Definition der Einarbeitung wurde die Bedeutung der Domäne und der Kolleg*innen bereits betont. Deshalb ist es wichtig, sowohl Interaktionspartner*innen als auch Nutzungsgruppen in der Visualisierung zu verdeutlichen. Erklärungen über die Benennung der Klassen und die verwendeten Symbole sind notwendig.

**Anforderung 2a:** Informationen aus den ersten Einarbeitungsprozessen liefern: Anwendungsfall, *Interaktionspartner*innen*, zuständige Kolleg*innen, Informationen zum Aufsetzen der Entwicklungsumgebung.

**Anforderung 10 bestätigt:** Informationen und Hintergründe liefern zur Benennung einzelner Bestandteile der Software und dort, wo durch den Namen keinerlei Inhalt erkannt werden kann, genauere

Inhaltsbeschreibungen liefern.

**b) Struktur:** Die Visualisierung soll einen generellen Überblick über die Softwarestruktur bieten. Bezüglich der verschiedenen Ebenen wurde in den Interviews festgehalten, dass eine Komplettübersicht notwendig ist, sowie das Zoomen in untere Ebenen. Eine interviewte Person schlägt dabei vor, in zwei Ebenen unterhalb der Komplettübersicht zoomen zu können. Bei einer OSGi Software entspricht dies dem Blick innerhalb der Packages.

Es sollen gegenseitige Abhängigkeiten dargestellt werden und welche Services welche anderen benutzen. Eine Anzeige, welche Services die zentralen sind, rundet den Überblick ab. Zentrale Services sind dabei nicht nur nach der Häufigkeit der Nutzung zu bestimmen, sondern anderweitig zu priorisieren, da einige Services – beispielsweise ein Datumservice – häufig genutzt wird aber keine zentrale Rolle spielt. *„Wenn also irgendwas Dateservice heißt, das brauche ich mir nicht zu merken, wenn ich mich gerade mit der Workflowausführung beschäftige. Und teilweise ist es ja dann sogar noch so, dass wenn versucht wird etwas zu gewichten, das meistens nach der Idee abläuft, Services zu priorisieren auf die von vielen Klassen zugegriffen wird. Das ist ja an sich logisch, das heißt aber halt, dass so ein Fileservice und so ein Dateservice, die überall gebraucht werden, werden auf einmal priorisiert.“* Eine Möglichkeit, diese Priorisierung umzusetzen, wäre es, Anomalien hervorzuheben. Zusätzlich ist bei Nutzung der Visualisierung die Verbindung zur aktuell getätigten Arbeit notwendig.

Ein Suchweg durch den Code soll im Nachhinein in der Visualisierung anzeigbar gemacht werden. Außerdem soll die Visualisierung helfen, einen Überblick über den Kontext, also die Umgebung in der man sich im Code befindet, zu liefern.

**Anforderung 4b:** Aktuelle und automatisch generierte Übersicht der Softwarestruktur mit gegenseitigen Abhängigkeiten und Services anzeigen.

**4.1.** Spezifizierbar auf jeweilig gelöste Aufgaben: Umgebung und Abhängigkeiten der bearbeiteten Codestelle anzeigen.

**4.2.** *Zoomen in verschiedene Ebenen ermöglichen, mit mindestens zwei Ebenen unter Gesamtübersicht.*

**4.3.** *Zentraler Services anzeigen, die priorisiert sind und nicht nur nach Häufigkeit eingestuft werden.*

**Anforderung 11a:** Strukturteile hervorheben oder dynamisch darstellen, die für den aktuellen Fall relevant sind.

**11.1.** *Suchweg durch den Code im Nachhinein hervorheben.*

**c) Funktionen:** Es wurde eine Reihe an Funktionen genannt, die von der Softwarevisualisierung erwartet werden und hier aufgelistet sind.

**Anforderung 16:** Suchfunktion bieten.

**Anforderung 17:** Markieren von Bestandteilen ermöglichen, um zusätzliche Informationen anzuzeigen (u.a. wo Dokumentation liegt).

Um die Komplexität der Darstellung zu minimieren, ist es zudem notwendig, nur eine begrenzte Menge an Informationen pro gezeigter Schicht darzustellen. Außerdem sollten für den aktuellen Fall irrelevante Informationen ausgeblendet werden können.

**Anforderung 18:** Ausblenden irrelevanter Bestandteile ermöglichen.

Um ein besseres Verständnis der jeweiligen Codestelle zu bekommen, soll nach Auswahl eines Elements der Visualisierung, beispielsweise einer Klasse, angezeigt werden können, welche anderen Teile des Codes von einer Änderung innerhalb dieser Klasse auch betroffen wären und deshalb ebenfalls angepasst oder kontrolliert werden müssen.

**Anforderung 4b bestätigt:** Aktuelle und automatisch generierte Übersicht der Softwarestruktur mit gegenseitigen Abhängigkeiten und Services anzeigen.

**4.1.** Spezifizierbar auf jeweilig gelöste Aufgaben: Umgebung und Abhängigkeiten der bearbeiteten Codestelle anzeigen.

**4.2.** Zoomen in verschiedene Ebenen ermöglichen, mit mindestens zwei Ebenen unter Gesamtübersicht.

**4.3.** Zentrale Services anzeigen, die priorisiert sind und nicht nur nach Häufigkeit eingestuft werden.
Auch die Testausführung innerhalb der Visualisierung wird als hilfreich eingeschätzt. *„Wenn ich so eine Aktivierung machen könnte, wenn ich sage ich führe diesen Test aus, was wird da jetzt alles angesteuert? Und ich würde in einer Übersicht sehen, diese fünf sind das. Man könnte das vielleicht noch schrumpfen auf diese fünf und ins Detail gehen. Und könnte sich dann die einzelnen fünf anschauen. Das wäre interessant.“*

**Anforderung 14 bestätigt:** Datenfluss bei Testdurchführungen innerhalb der Visualisierung darstellen.
Um später nochmals auf die Erkenntnisse aus der Arbeit mit der Visualisierung zurückzukommen, ist das Speichern des aktuell gesehenen Bildes notwendig. So kann später erneut angezeigt werden, welche Teile ausgeblendet oder zusätzlich eingeblendet waren und auf welcher Ebene man sich befand.

**Anforderung 19:** Screenshotfunktion bieten und Wiederaufrufen der letzten Interaktionen mit der Visualisierung ermöglichen.

**d) Verbindung Visualisierung und IDE:** Generell ist eine entscheidende Frage, inwieweit der genaue Quellcode in die Visualisierung integriert werden sollte. Dabei halten es die meisten Interviewten für nicht zielführend, den kompletten Code im Rahmen der Visualisierung anzeigen zu lassen und würden somit den Code und die Visualisierung als getrennte Darstellungen belassen. Allerdings soll die Visualisierung eng in die Entwicklungsumgebung integriert sein. *„Gerade wenn das eng in die Entwicklung integriert ist, wäre eine graphische Übersicht gut über die Verbindungen und Abhängigkeiten, die man dann auch nutzen kann um direkt wieder zu navigieren. Ich habe die Erfahrung gemacht, wenn so etwas extern ist, ist das eine sehr große Hürde es zu benutzen. Man denkt sich, ich könnte die Visualisierung anwerfen, aber dann muss ich da erst mal nachschauen und dann wieder im anderen Editor zurück.“* Durch eine Kopplung des Codes an die Visualisierung ließe sich also die Darstellung einzelner Codestellen in der Visualisierung vermeiden. Diese wäre direkt mit dem Code verknüpft, und ist deshalb eine Einheitenebene über dem Code, also auf der Funktionsebene, anzusetzen.

**Anforderung 15a:** IDE und Visualisierung direkt verbinden.

**15.1.** *Visualisierung an der Stelle starten, die gerade in IDE bearbeitet wird.*

**e) Metapher:** Durch die bekannte Visualisierung mithilfe einer Inselmetapher ist allen Interviewten die Arbeit mit metaphorischen Darstellungen bekannt. Ob dies einer abstrakten Darstellung vorzuziehen ist, wird unterschiedlich bewertet. Eine interviewte Person betont, dass eine abstrakte Darstellung näher an der zu lösenden Aufgabe ist und das abstrakte Denken generell keine große Schwierigkeit im Bereich der Softwareentwicklung ist. Ein weiterer Interviewpartner hält abstrakte Visualisierungen für sinnvoll, wenn Dinge besonders schnell erfasst werden sollen und dabei nicht sehr komplex visualisiert werden. Allerdings halten beide eine Visualisierung mittels Metapher nicht für ausgeschlossen. Andere Interviewpartner sehen eine Visualisierung mithilfe von Metaphern positiv. *„Also tatsächlich finde ich so eine Metapher eigentlich gut. Das hilft einem schon. Also man muss natürlich einmal erklärt bekommen, wofür stehen jetzt die Inseln, die Häfen, die Städte und so weiter. Aber dann ist das besser zu verstehen“*

**Anforderung 20:** Einführung bieten, in für die metaphorische Darstellung verwendeten Regeln.

**f) Medium:** Der Umgang und die konkrete Interaktion am Medium PC sind bekannt und alltäglich. Deshalb wird generell eine Visualisierung am PC als praktikabler, zweckmäßiger und intuitiver in der Bedienung eingeschätzt als eine Visualisierung in VR oder AR. Vor allem durch die geforderte Verbindung von Visualisierung und Code hat eine Visualisierung, die mittels derselben Hardware realisiert wird, die geringsten Wechselkosten. In der Arbeit mit VR und AR werden allerdings auch Vorteile gesehen, wie der Sense of Space oder die Möglichkeit, Kontextinformationen möglicherweise noch besser darzustellen als in einer 2D-Visualisierung. Die Nachteile überwiegen allerdings. In den Interviews wird betont, dass für eine Einarbeitung auch gemeinsam an einer Visualisierung gearbeitet werden können muss, und es notwendig ist, dass dort etwas gezeigt werden kann. Dies spricht gegen die Verwendung von VR. In einem Interview wird herausgestellt, dass allerdings das gemeinsame Betrachten in AR, um einen ersten Eindruck zu bekommen, durchaus hilfreich sein könnte. Entsprechend der zweigeteilten Definition der Einarbeitung wäre es denkbar, die Visualisierung, die vor allem auf den Kontext und das gesamte Projekt eingeht, in einer AR Version zu erstellen. So kann der oder die neue Softwareentwickler*in unter der Anleitung einer erfahrenen Person eingearbeitet werden. Im zweiten Abschnitt der Einarbeitung, also der Arbeit an konkreten Aufgaben, sollte dann auf eine Visualisierung am PC zurückgegriffen werden.

**Anforderung 5a:** Beide Einarbeitungsphasen unterstützen durch

**5.1.** Den Teil der Visualisierung ausblenden bzw. deaktivieren, der bei der ersten Phase zusätzlich unterstützen soll, oder

**5.2.** verschiedene Visualisierungen für die jeweilige Phase entwickeln.

**5.3.** *Die Visualisierung, die für die zweite Phase der Einarbeitung genutzt wird, am PC darstellen.*

**g) Nutzung der Visualisierung** Die manuelle Aufrechterhaltung der Visualisierung resultiert in einem erhöhten Arbeitsaufwand. Deshalb ist die automatische Generierung der Visualisierung aus dem Quellcode notwendig. Außerdem soll die Visualisierung einfach handhabbar sein, ohne fremde Hilfe aufgesetzt und genutzt werden können und eine Einführung in verwendete Symbolik bieten. Besonders wenn die Möglichkeit zweier separater Visualisierungen für die verschiedenen Einarbeitungsphasen in Betracht gezogen wird, wäre es sinnvoll, dieselben Metaphern und Regeln zu verwenden.

**Anforderung 1a bestätigt:** Einfache Bedienbarkeit und intuitive Nutzbarkeit bieten.

**1.1.** Für erste Einarbeitungsphase: ohne große Vorbereitung nutzbar machen.

**1.2.** Für zweite Einarbeitungsphase: ausreichende Erklärung und Einführung bieten, zur wiederholten Nutzung mit mehr Funktionen.

**Anforderung 4b bestätigt:** Aktuelle und automatisch generierte Übersicht der Softwarestruktur mit gegenseitigen Abhängigkeiten und Services anzeigen.

**4.1.** Spezifizierbar auf jeweilig gelöste Aufgaben: Umgebung und Abhängigkeiten der bearbeiteten Codestelle anzeigen.

**4.2.** Zoomen in verschiedene Ebenen ermöglichen, mit mindestens zwei Ebenen unter Gesamtübersicht.

**4.3** Zentrale Services anzeigen, die priorisiert sind und nicht nur nach Häufigkeit eingestuft werden.

**Anforderung 5b:** Beide Einarbeitungsphasen unterstützen durch

**5.1.** Den Teil der Visualisierung ausblenden bzw. deaktivieren, der bei der ersten Phase zusätzlich unterstützen soll, oder

**5.2.** verschiedene Visualisierungen für die jeweilige Phase entwickeln, *mit denselben Metaphern und Regeln.*

**5.3.** Die Visualisierung, die für die zweite Phase der Einarbeitung genutzt wird, am PC darstellen.

## 5.4 Gesamtübersicht Anforderungen

Für die weitere Arbeit in der Abteilung für Simulations- und Softwaretechnik im DLR werden die in diesem Kapitel vorgestellten Anforderungen inhaltlich geordnet und zu einer Übersicht zusammengefügt, welche die Grundlage für Entwicklungsarbeiten der Visualisierung bietet. Um die Zuordnung der zuvor erläuterten Anforderungen zu erleichtern, wird in der Übersicht die Nummerierung in Klammern vor den Anforderungen beibehalten.

### 5.4.1 Unterscheidung in zwei Einarbeitungsphasen

- (5b) Beide Einarbeitungsphasen unterstützen durch
  - den Teil der Visualisierung ausblenden bzw. deaktivieren, der bei der ersten Phase zusätzlich unterstützen soll, oder
  - verschiedene Visualisierungen für die jeweilige Phase entwickeln, mit denselben Metaphern und Regeln.
  - Die Visualisierung, die für die zweite Phase der Einarbeitung genutzt wird, am PC darstellen.
- (1a) Einfache Bedienbarkeit und intuitive Nutzbarkeit bieten.
  - Für erste Einarbeitungsphase: ohne große Vorbereitung nutzbar machen.
  - Für zweite Einarbeitungsphase: ausreichende Erklärung und Einführung bieten, zur wiederholten Nutzung mit mehr Funktionen.
- (2a) Informationen aus den ersten Einarbeitungsprozessen liefern: Anwendungsfall, Interaktionspartner*innen, zuständige Kolleg*innen, Informationen zum Aufsetzen der Entwicklungsumgebung.

### 5.4.2 Strukturübersicht

- (3) Beim Finden der zu bearbeitenden Codestelle Hilfestellung leisten. Für die ersten Aufgaben die passende Stelle in Visualisierung markieren.
- (4b) Aktuelle und automatisch generierte Übersicht der Softwarestruktur mit gegenseitigen Abhängigkeiten und Services anzeigen.
  - Spezifizierbar auf jeweilig gelöste Aufgaben: Umgebung und Abhängigkeiten der bearbeiteten Codestelle anzeigen.
  - Zoomen in verschiedene Ebenen ermöglichen, mit mindestens zwei Ebenen unter Gesamtübersicht.
  - Zentrale Services anzeigen, die priorisiert sind und nicht nur nach Häufigkeit eingestuft werden.
- (11a) Strukturteile hervorheben oder dynamisch darstellen, die für den aktuellen Fall relevant sind.
  - Suchweg durch den Code im Nachhinein hervorheben.
  - (12) Stelle hervorheben, an der ein Datenfluss nicht korrekt ausgeführt werden kann.
  - (14) Datenfluss bei Testdurchführungen innerhalb der Visualisierung darstellen.

### 5.4.3 Zusätzlich aufzurufende Inhalte

- (13) Informationen liefern, welche Teile des Codes lange nicht verändert wurden bzw. besonders häufig verändert wurden.
- (7a) Information liefern, wer welchen Teil des Codes bearbeitet hat, mit welchem Ziel dieser bearbeitet wurde und mit welcher Issue-ID diese Änderung verbunden ist.
- (8a) Informationen liefern, wie das Wissen über bestimmte Aspekte verteilt ist und wer die aktuelle, direkte Ansprechperson für eine bestimmte Frage ist.
- (10) Informationen und Hintergründe liefern zur Benennung einzelner Bestandteile der Software und dort, wo durch den Namen keinerlei Inhalt erkannt werden kann, genauere Inhaltsbeschreibungen liefern.

### 5.4.4 Funktionen

- (16) Suchfunktion bieten.
- (17) Markieren von Bestandteilen ermöglichen, um zusätzliche Informationen anzuzeigen (u.a. wo Dokumentation liegt).
- (18) Ausblenden irrelevanter Bestandteile ermöglichen.
- (19) Screenshotfunktion bieten und Wiederaufrufen der letzten Interaktionen mit der Visualisierung ermöglichen.

### 5.4.5 Anwendung und Verbindung zur Arbeitstätigkeit

- (20) Einführung bieten, in für die metaphorische Darstellung verwendeten Regeln.
- (6) Visualisierung mit beiden Informationsquellen (Developer Guide und Wiki) verbinden, mit jeweiligen Verweisen, wo weitere benötigte Informationen liegen.
- (15a) IDE und Visualisierung direkt verbinden.
  - Visualisierung an der Stelle starten, die gerade in IDE bearbeitet wird.
- (9) (für Visualisierung mit VR): Notizzettel in Visualisierung einfügen, um Stichworte mitschreiben zu können, ohne die Visualisierung zu unterbrechen.

Im anschließenden Kapitel wird diskutiert, inwieweit die hier gefundenen Ergebnisse und geforderten Eigenschaften der Softwarevisualisierung mit den Anforderungen zusammenpassen, die in der Literatur zu finden sind. Außerdem werden erste Ansätze aufgezeigt, wie die zu entwickelnde Softwarevisualisierung zu klassifizieren ist.

# 6 Diskussion

## 6.1 Integration der Ergebnisse und Vergleich mit Literatur

Im Rahmen der Arbeit sollten drei Fragen beantwortet werden: Wie die Einarbeitung in ein neues Softwareprojekt aussieht, bei welchen Teilschritten und –aufgaben davon Unterstützungsbedarf besteht, also eine Visualisierung zum Einsatz kommen könnte, und welche Anforderungen an eine solche Visualisierung gestellt werden.

Die Einarbeitung ist als zweiteiliger Prozess zu beschreiben. Zunächst findet die Einarbeitung in den Anwendungsfall und die Abteilung mit zuständigen Kolleg*innen statt. Anschließend wird in wiederkehrender Abfolge eine Aufgabe aus dem Bugtracker bearbeitet. Die Softwarevisualisierung, die für den Anwendungsfall der Einarbeitung in ein neues Softwareprojekt realisiert wird, ist demnach ebenso zweizuteilen. Es sind entweder zwei unabhängige Visualisierungen zu realisieren, die in ihren verwendeten Metaphern und Regeln im Sinne der einfachen Nutzbarkeit übereinstimmen. Alternativ ist es notwendig, im Rahmen einer einzigen Visualisierung den inhaltlichen Teil der ersten Einarbeitungsphase deaktivieren zu können.

Ein Bedarf an Unterstützung beim Einarbeitungsprozess durch eine Softwarevisualisierung wurde nicht uneingeschränkt gesehen. Die negative Haltung gegenüber Softwarevisualisierungen war dabei vor allem durch die bereits in der Abteilung bestehende und bekannte Visualisierung geprägt. Diese wurde nicht spezifisch für die Anwendungsgruppe der Softwareentwickler*innen entwickelt und ist nicht für den Anwendungsfall der Einarbeitung zugeschnitten. Deshalb kann daraus nicht abgeleitet werden, dass eine neue Softwarevisualisierung für den definierten Anwendungsfall ebenfalls als wenig hilfreich wahrgenommen wird. Da einige Komponenten im Einarbeitungsprozess erwähnt wurden, bei denen externe Unterstützung hilfreich wäre, um die Einarbeitung in ein neues Softwaresystem zu vereinfachen und zu beschleunigen, ist die Entwicklung eines Hilfsmittels sinnvoll. Dass hierfür eine Visualisierung geeignet ist, lässt sich aus den auch vorhandenen positiven Einstellungen im Rahmen der Interviews schließen. Vor allem Interviewpartner*innen, die bereits mit für die Entwickler*innen zugeschnittenen Visualisierungen gearbeitet haben, sehen Potential. Dies deckt sich mit den Ergebnissen von Koschke (2003), bei dessen Befragung 82% der Forschenden im Bereich Softwarewartung, Reverse Engineering und Reengineering angaben, dass Softwarevisualisierungen notwendig oder sehr wichtig für ihre Arbeit seien. Die von Bassil und Keller (2001) benannten wichtigsten Vorteile von Softwarevisualisierungen der Zeitersparnis und des besseren Verständnisses der Softwarestruktur wurden von den Interviewten ergänzt. Die Visualisierung spart den Aufwand, ein mentales Modell über die aktuelle Situation im Hinterkopf behalten zu müssen.

In den Anforderungen an die Softwarevisualisierung stimmen die hier vorliegenden Ergebnisse mit der Literatur zu großen Teilen überein. Die Interviewten betonen vor allem die notwendige Verbindung der Visualisierung mit der Entwicklungsumgebung und dem Code. Ein einfacher Zugang vom verwendeten Symbol zum Quellcode wurde auch von Bassil und Keller (2001) als wichtigste Anforderung ermittelt und von Koschke (2003) bestätigt. Die weiteren von Bassil und Keller (2001) aufgezeigten Anforderungen finden sich ebenfalls in den Ergebnissen dieser Studie wieder. Die Möglichkeit, die Softwarestruktur zu durchsuchen und über verschiedene Hierarchien zu navigieren, wird als relevant eingeschätzt. Zusätzlich bestätigte sich die gering gesehene Nützlichkeit von VR-Techniken, die auch in der Befragung von Bassil und Keller (2001) angegeben wurde. VR-Techniken wurden auch von den Interviewten, vor allem für die zweite Phase der Einarbeitung, zum aktuellen Zeitpunkt als nicht zielführend eingestuft.

Entgegen des sogenannten Mantras der Informationssuche von Shneiderman (1996) „Overview

first, zoom and filter, and then details-on-demand“, wird bei der Einarbeitung der Weg von innen nach außen zum Überblick über die Gesamtstruktur verfolgt. Dieser Grundsatz wird für die Visualisierung dahingehend realisiert, dass Zoomen und Ausblenden irrelevanter Informationen genauso wie der mögliche Abruf von Zusatzinformationen ermöglicht wird. Der Start der Visualisierung ist allerdings nicht der Überblick über die Struktur, sondern das Lösen konkreter Aufgaben auf einer tieferen Ebene der Struktur, über die dann durch Herauszoomen die Gesamtstruktur betrachtet werden kann. Die in dieser Arbeit aufgestellten Anforderungen geben einen Ausblick darauf, dass die drei charakteristischen Bestandteile von Informationsvisualisierungen nach Schumann und Müller (2004) auch im Rahmen der zu entwickelten Softwarevisualisierung wiederzufinden sind: Unterschiedlich erzeugte Bilder für verschiedene visuelle Zugänge, Interaktionstechniken zum Wechsel zwischen diesen Bildern (hier beispielsweise Ebenen der Struktur) und die Ermöglichung des gezielten Zu- und Wegschaltens von Informationen.

## 6.2 Klassifizierung der Softwarevisualisierung

In der Unterscheidung nach Ball und Eick (1996) in Quellcode, Softwarestruktur und Laufzeitverhalten lässt sich die zu entwickelnde Visualisierung der Softwarestrukturvisualisierung zuordnen. Der Quellcode soll zwar eng mit der Visualisierung verknüpft und demnach sehr schnell aufrufbar sein, wird aber nicht an sich visualisiert. Diese Strukturvisualisierung wird allerdings insoweit individualisierbar gestaltet, als dass über Interaktionstechniken und Zu- bzw. Ausblenden von Inhalten ermöglicht wird, die Visualisierung auf ein konkretes Problem zuzuschneiden. Bei der Klassifizierung nach Panas et al. (2007) in Quellcodelevel, Architekturlevel und problemspezifisches mittleres Level, lässt sich die zu entwickelnde Visualisierung als eine Kombination der beiden letzten Arten ansehen. Nach Price et al. (1993) lassen sich Softwarevisualisierungen darüber hinaus nach Fragen zu Umfang, Inhalt, Form, Methode, Interaktion und Effektivität klassifizieren. Nach dem Erstellen der Anforderungsanalyse können diese Fragen über die zu entwickelnde Softwarevisualisierung beantwortet und damit ein erster zusammenfassender Überblick über diese gegeben werden.

1 **Umfang:** Die Visualisierung ist für die Einarbeitung in Softwareprojekte der Einrichtung Simulations- und Softwaretechnik des DLRs konzipiert. Demnach umfasst der Umfang die in den jeweiligen Projekten verwendeten Programmiersprachen.
2 **Inhalt:** Es wird die Softwarestruktur mit allen Modulen und gegenseitigen Abhängigkeiten visualisiert. Außerdem werden die Historie, die Nomenklatur, Zuständigkeiten von Kolleg*innen und Kontextinformationen dargestellt.
3 **Form:** Die Visualisierung der zweiten Einarbeitungsphase ist eine Visualisierung am PC. Für die erste Phase ist darüber hinaus die Visualisierung in AR eine Möglichkeit.
4 **Methode:** Es gibt eine direkte Verbindung von Visualisierung und Entwicklungsumgebung. Außerdem wird eine Metapher genutzt, deren Feinheiten für die zweite Einarbeitungsphase ausführlich erklärt werden.
5 **Interaktion:** Es gibt Interaktionsmöglichkeiten, wie das Zoomen, Ein-, Ausblenden und Hervorheben bestimmter Informationen, Suchen nach bestimmten Stellen und Erstellen eines Screenshots.
6 **Effektivität:** Die Effektivität muss im Rahmen einer Evaluation nach der Entwicklung einer ersten Version überprüft werden. Die festgestellten Anforderungen sollen eine hohe Effektivität gewährleisten.

## 6.3 Kritik und Forschungsimplikationen

Zum Entstehungszeitpunkt dieser Arbeit gab es keine neuen Mitarbeitenden in der Einrichtung, die sich gerade in ein Softwareprojekt einarbeiteten und beobachtet hätten werden können. Es konnten Interviewpartner*innen gewonnen werden, die seit Ende des letzten Jahres in der Einrichtung arbeiten und demnach noch gut vor Augen haben, wie die erste Einarbeitungsphase aussah. Bei anderen Interviewpartner*innen lag der Zeitpunkt der letzten Einarbeitung in ein Softwareprojekt bereits einige Jahre in der Vergangenheit, was die Erinnerung des genauen Ablaufes möglicherweise gefährdet. Dennoch konnte durch den detaillierten Interviewleitfaden und gezieltes Nachfragen nach den einzelnen Schritten der Einarbeitungsprozess genau beschrieben werden.

Die Einschätzungen bezüglich der Effektivität von Softwarevisualisierungen waren möglicherweise stark von dem geprägt, was in der Einrichtung bereits bekannt war. Die zu entwickelnde Softwarevisualisierung ist genau auf den Anwendungsfall zugeschnitten, weshalb eine abschließende Bewertung des Nutzens der Softwarevisualisierung für den Einarbeitungsprozess anhand dieser neuen Visualisierung erfolgen muss.

Die hier erarbeitete Anforderungsanalyse bildet die Grundlage für die Entwicklung einer zielgerichteten und effektiven Unterstützung durch die Visualisierung. Um die Qualität der aufgestellten Anforderungen zu sichern, wurden die Qualitätsmerkmale Adäquatheit, Vollständigkeit, Widerspruchsfreiheit, Verständlichkeit, Eindeutigkeit und Prüfbarkeit in der Erarbeitung der Analyse berücksichtigt. Auf Adäquatheit der Anforderungen wurde während des Ableitungsprozesses geprüft, weshalb beispielsweise die in einem Interview gewünschte Hilfestellung beim Verständnis der Nutzungsoberfläche nicht als Anforderung aufgenommen wurde. Diese wird einem anderen Aufgabenbereich der Abteilung zugeordnet. Widerspruchsfreiheit wurde durch Abwägung und Kombination einzelner Interviewaussagen zu den jeweiligen Unterthemen erreicht. Verständlichkeit soll dadurch sichergestellt werden, dass die Anforderungen im Rahmen der Bachelorarbeit umfassender erklärt und teilweise mit Zitaten verdeutlicht werden. Eindeutigkeit und Prüfbarkeit wurden bei der Formulierung der Anforderungen berücksichtigt und bieten damit die Ausgangslage späterer Evaluationen der Visualisierung.

Wenn die hier erarbeiteten Anforderungen erfüllt sind, kann die Softwarevisualisierung dazu beitragen, dass neuen Mitarbeitenden in der Abteilung Simulations- und Softwaretechnik die Einarbeitung in das für sie neue Softwareprojekt einfacher gemacht wird. Der nächste Schritt dafür ist der Entwurf eines Modells der Visualisierung, welches spezifiziert und realisiert wird. Durch Evaluation von Zwischenschritten, wie beispielsweise ersten Designvorschlägen, kann sichergestellt werden, dass eine Umsetzung der Softwarevisualisierung den Anforderungen entspricht. Besonders in einem so langfristigen Projekt, wie der Arbeit an den Softwaresystemen in der Einrichtung, können sich Anforderungen an eine Visualisierung gegebenenfalls projektbedingt verändern. Um dies festzustellen sind der regelmäßige Kontakt und die Rücksprache mit den Softwareentwickler*innen im weiteren Entwicklungsprozess der Visualisierung essentiell. In einer abschließenden Evaluation der fertiggestellten Softwarevisualisierung ist zu bestimmen, ob die aufgestellten Anforderungen umgesetzt wurden und inwieweit dies zu einem hohen Ausmaß der Usability führt.

# Literaturverzeichnis


Avci, O. (2008). Warum entstehen in der Anforderungsanalyse Fehler? Eine Synthese empirischer Befunde der letzten 15 Jahre. In G. Herzwurm & M. Mikusz (Hrsg.), *Industrialisierung des Software-Managements* (S. 89-103). Bonn: Gesellschaft für Informatik e.V.

Ball, T. & Eick, S. G. (1996). Software visualization in the large. *IEEE Computer Society Press*, *29* (4), 33–43.

Balzert, H. (2000). *Lehrbuch der Software-Technik: Software-Entwicklung* (2. Aufl.). Berlin, Heidelberg: Spektrum, Akademischer Verlag.

Banker, R. D., Davis, G. B. & Slaughter, S. A. (1998). Software development practices, software complexity, and software maintenance performance: A field study. *Management science*, *44* (4), 433–450.

Bassil, S. & Keller, R. K. (2001). Software visualization tools: Survey and analysis. In *Proceedings 9th international workshop on program comprehension. iwpc 2001* (S. 7–17). IEEE Computer Society Press.

Brooks, F. P. (1974). The mythical man-month. *Datamation*, *20* (12), 44–52.

Card, M. (1999). *Readings in information visualization: using vision to think.* San Francisco: Morgan Kaufmann.

Diehl, S. (2007). *Software Visualization: Visualizing the Structure, Behaviour, and Evolution of Software.* Berlin, Heidelberg: Springer-Verlag.

Dijkstra, E. W. (1972). The humble programmer. *Communications of the ACM*, *15* (10), 859–866.

Dresing, T. & Pehl, T. (2018). *Praxisbuch Interview, Transkription & Analyse. Anleitungen und Regelsysteme für qualitativ Forschende.* (8. Aufl.). Marburg: Eigenverlag.

Döhring, N. & Bortz, J. (2016). *Forschungsmethoden und Evaluation in den Sozial-und Humanwissenschaften* (5. Aufl.). Berlin, Heidelberg: Springer-Verlag.

Gershon, N. & Eick, S. G. (1997). Information visualization. *IEEE Computer Graphics and Applications*, *17* (4), 29–31.

Gerstl, S. (2018). *Raus aus der Software-Krise: 50 Jahre Software-Engineering.* Zugriff am 05.08.2019 unter https://www.embedded-software-engineering.de/raus-aus-der-software-krise-50-jahre-software-engineering-a-765527.

Glinz, M. (2002). Software Engineering [Vorlesungsskript]. *Universität Zürich*. Zugriff auf `https://homepages.fhv.at/hv/Semester4/OOAD/glinz.pdf`

Haber, R. N. & Wilkinson, L. (1982). Perceptual components of computer displays. *IEEE Computer Graphics and Applications*, *2* (3), 23–35.

Hagemann, J. & Henle, J. (2014). *Transkribieren nach GAT 2 (Minimal-und Basistranskript)–Schritt für Schritt.* Zugriff auf `https://www.ph-freiburg.de/fileadmin/dateien/mitarbeiter/hagemannfr/Transkribieren_nach_GAT_2.pdf`

Kaiser, R. (2014). *Qualitative Experteninterviews: Konzeptionelle Grundlagen und praktische Durchführung.* Wiesbaden: Springer-Verlag.

Koschke, R. (2003). Software Visualization in Software Maintenance, Reverse Engineering, and Reengineering: A Research Survey. *Journal of Software Maintenance and Evolution: Research and Practice*, *15* (2), 87–109.

Mayring, P. (2010). *Qualitative Inhaltsanalyse: Grundlagen und Techniken* (11. Aufl.). Weinheim: Beltz.

McCormick, B. H. (1988). Visualization in scientific computing. *ACM SIGBIO Newsletter*, *10* (1), 15–21.

Mieg, H. A. & Brunner, B. (2001). *Experteninterviews.* (MUB-Working Paper 6). Professur für Mensch-Umwelt-Beziehungen, ETH Zürich.

Newman, W. M., Lamming, M. G. & Lamming, M. (1995). *Interactive system design.* Wokingham: Addison-Wesley.

Nielsen, J. (1993). *Usability Engineering.* San Diego: Academic Press.

Panas, T., Epperly, T., Quinlan, D., Saebjornsen, A. & Vuduc, R. (2007). Communicating Software Architecture using a unified Single-View Visualization. In *12th IEEE International Conference on Engineering Complex Computer Systems (ICECCS 2007)* (S. 217–228). IEEE.

Pfadenhauer, M. (2009). Das Experteninterview. In *Qualitative Marktforschung* (S. 449–461). Berlin, Heidelberg: Springer-Verlag.

Preim, B. & Dachselt, R. (2010). *Interaktive Systeme: Band 1: Grundlagen, Graphical User Interfaces, Informationsvisualisierung.* Berlin, Heidelberg: Springer-Verlag.

Preim, B. & Dachselt, R. (2015). *Interaktive Systeme. Band 2. User Interface Engineering, 3D-Interaktion, Natural User Interfaces.* Berlin, Heidelberg: Springer-Verlag.

Price, B. A., Baecker, R. M. & Small, I. S. (1993). A principled taxonomy of software visualization. *Journal of Visual Languages & Computing*, *4* (3), 211–266.

Richter, M. & Flückiger, M. D. (2013). *Usability Engineering kompakt: Benutzbare Produkte gezielt entwickeln.* Berlin, Heidelberg: Springer-Verlag.

Robertson, P. K. (1990). A methodology for scientific data visualisation: choosing representations based on a natural scene paradigm. In *Proceedings of the 1st conference on visualization'90* (S. 114–123). IEEE Computer Society Press.

Schumann, H. & Müller, W. (2004). Informationsvisualisierung: Methoden und perspektiven. *it-Information Technology*, *46* (3), 135–141.

Schumann, H. & Müller, W. (2013). *Visualisierung: Grundlagen und allgemeine Methoden.* Berlin, Heidelberg: Springer-Verlag.

Shneiderman, B. (1996). The eyes have it: A task by data type taxonomy for information visualizations. In *Proceedings of the 1996 IEEE Symposium on Visual Languages* (S. 336). IEEE Computer Society.

Sommerville, I. (2011). *Software Engineering* (9. Aufl.). Boston: Pearson Education.

The Standish Group. (1994). *The CHAOS Report.* Zugriff auf `https://www.standishgroup.com/sample_research_files/chaos_report_1994.pdf`

Zimmermann, J., Konrad, S. & Nerdinger, F. W. (2009). *Bedarfs- und Anforderungsanalyse zur Entwicklung einer internetbasierten Kommunikationsplattform zur Unterstützung des Forschungstransfers.* Rostock: Lehrstuhl für ABWL: Wirtschafts- und Organisationspsychologie der Universität Rostock.

# Anhang

## Anhang A: Interviewleitfaden

**Interviewleitfaden Anforderungsanalyse Softwarevisualisierung**

Datum/Uhrzeit: _____

Interviewpartner*in: _____

**Einleitung (ohne Aufnahme)**

a. Warum Interview?
   i. Ziel: neue Visualisierung zur Einarbeitung neuer Mitarbeitenden
   ii. Bachelorarbeit – über mehrere Interviews Anforderungsanalyse erstellen

b. Ablauf
   i. Dauer: 1h bis 1,5h
   ii. Aufgenommen – Anonym ausgewertet und in Zusammenhang mit anderen Interviews gebracht
   iii. Erst Fragen zur Person (hauptsächlich Hintergrund) – dann: Inhaltlich zu Softwarevisualisierung

| **1) Person und Rolle** | | | |
|---|---|---|---|
| Kernfragen | *Mögliche Nachfragen* | *Erwartete Informationen* | |
| Sie Sind Softwareentwickler*in?<br><br>Beschreiben Sie kurz Ihren beruflichen Werdegang.<br><br>Was ist aktuell Ihre genaue Tätigkeit? | | *Ja/Nein, seit wann, Studium, bisherige Jobs, aktuell Entwicklung von Software?* | |
| Haben Sie bereits mit Softwarevisualisierungen gearbeitet?<br><br>*Falls nein* – Kennen Sie sich mit Softwarevisualisierung aus?<br><br>➔ *Ansonsten kurzer Überblick* | *Welche, wofür, wie lange?*<br><br>*Erfahrung eher positiv oder negativ einschätzen?*<br><br>*Schon mal zur Einarbeitung in neues Softwareprojekt genutzt?* | *Visualisierung genutzt oder erstellt; Zweck; Dauer; Gerne oder ungerne genutzt;* | |

| **2) Was unterscheidet Visualisierung von Quellcode?** | | | |
|---|---|---|---|
| Kernfragen | *Mögliche Nachfragen* | *Erwartete Informationen* | |
| *Eröffnungsfrage*: Kam es schon einmal vor, dass Sie sich eine Visualisierung gewünscht hätten anstatt mit Quellcode zu arbeiten?<br><br>Bei welchen (weiteren) Aufgaben könnte die Visualisierung geeigneter sein als der Quellcode?<br><br>Gibt es Aufgaben, für die der Quellcode geeigneter ist? | *Ja → Bei welchen Aufgaben? Was wäre einfacher gewesen?*<br><br>*Warum? Wie genau würde Visualisierung unterstützen? Was genau ist Vorteil ggü. Code?*<br><br>*Warum? Wie genau würde Code unterstützen? Was genau ist Vorteil ggü. Visualisierung?* | *Unterschiedliche Nutzung,*<br><br>*Quellcode Detail, Visualisierung Überblick, Auswirkung von Veränderung auf andere Module*, Verständnis für Ganzes, | |
| **3) Einarbeitung in neue Software** | | | |
| Kernfragen | *Mögliche Nachfragen* | *Erwartete Informationen* | |
| *Eröffnungsfrage*: Wie oft haben Sie sich schon in eine neue Software eingearbeitet?<br><br>Haben Sie zur Einarbeitung bisher mit dem Quellcode gearbeitet oder standen weitere Dokumente/ Medien oder andere Hilfsmittel zur Verfügung?<br><br>*Stellen Sie sich vor Sie sind ein*e neue*r Mitarbeiter*in für ein Softwareprojekt und bevor Sie beginnen an einem bestimmten Modul zu arbeiten, sollen Sie sich zunächst in die Software einarbeiten.*<br><br>Gibt es Informationen, die schon vor der eigentlichen Einarbeitung (z.B. mit dem Quellcode) bekannt bzw. relevant sind? | *Gab es eine Dokumentation?*<br><br>*Ziel der Software? Wer arbeitet daran? Programmiersprache?* | *Häufigkeit*<br><br>*Mit Quellcode, kaum Dokumentation*<br><br>*Evtl. Spreadsheet vorher mit Main facts* | |

| | | | |
|---|---|---|---|
| Beschreiben Sie so kleinschrittig wie möglich den Prozess des Einarbeitens. Welche Informationen sind wann wichtig und warum?<br>1. Welche Frage stellt sich als erstes?<br>2. Wie geht man dann Schritt für Schritt vor? | *Was ist wichtigste Information am Anfang?*<br>*Detaillierter, was genau machen Sie als nächstes?* | *Erste Information, die gesucht wird: Hauptkomponente. Von dieser aus: Abhängigkeiten zu anderen Komponenten*<br><br>*evtl. wenn Aufgabe etwas bestimmtes anzupassen → diesen Bereich finden* | |
| Sie haben jetzt das schrittweise Vorgehen beschrieben. Wie genau, sieht ihrer Meinung nach der Zielzustand aus, der nach der Einarbeitung erreicht sein sollte? | *Welche Informationen über die Software müssen Sie wissen?*<br><br>*Wie genau ist der Detailgrad? (Namen von Komponenten?)*<br><br>*Ist für die Einarbeitung die Historie der Software von Relevanz* | *Hauptkomponenten, Abhängigkeiten, Wie alles verbunden, Größe und Umfang*<br><br>*keine Namen, eher oberflächlicher Überblick,*<br><br>*Historie nicht relevant, da aktueller Stand zunächst wichtig* | |
| *Wenn vorher bei Schritten nicht genannt:* Gibt es Zwischenschritte, nach denen Sie pausieren, etwas notieren, auch etwas ausprobieren oder versuchen Sie sich zunächst einen kompletten Überblick zu verschaffen? | *Abhängig von Software?*<br>*Wie oft pausieren?*<br>*Notizen oder Rücksprache?* | *Notieren, Pausieren und v.a. Abhängigkeiten anzeigen lassen (eigene Veränderungen vornehmen und Auswirkungen anschauen)* | |
| Wenn Sie sich bisher in neue Softwareprojekte eingearbeitet haben, arbeiteten Sie in Einzelarbeit oder Teamarbeit? Welche Arbeitsweise würden Sie bevorzugen und warum? | *Selbst entschieden?*<br>*Mit wem?*<br>*Alleine, mit Möglichkeit nach Hilfe zu fragen?* | *Teamarbeit bei Nachfragen mit eingearbeiteten Personen* | |

| | | | |
|---|---|---|---|
| Wenn Sie sich konkret daran erinnern, wann Sie sich das letzte Mal in ein Softwareprojekt eingearbeitet haben. Können Sie einen Zeitrahmen für die Einarbeitung angeben? Stunden, Tage, Wochen etc.? | *Erfolgt Einarbeitung am Stück und ist dann abgeschlossen oder sukzessiv? Abhängig von Software?* | *Tage bis Wochen → abhängig von Softwarekomplexität; generell nie beendet* | |
| **4 ) Kombination Visualisierung und Einarbeitung (aufgreifen, was vorher gesagt)** | | | |
| Kernfragen | *Mögliche Nachfragen* | *Erwartete Informationen* | |
| *Eröffnungsfrage*: Beim konkreten Anwendungsfall – Sie sind Softwareentwickler*in und fangen neu in einem Team an. Ihre erste Aufgabe ist es, sich in ein bestehendes Softwareprojekt, an dem sie arbeiten sollen, einzuarbeiten. Glauben Sie, eine Softwarevisualisierung ist dabei nützlich zur Einarbeitung? | *An welchem Schritt nützlich ?*<br><br>*Kombination mit anderen Medien? Quellcode und Visualisierung trennen? Oder Quellcode jeweils anzeigen lassen können?*<br><br>*Wie lange mit Visualisierung arbeiten? Einmalig Visualisierung nutzen und danach gar nicht mehr? Vorstellen auch später noch mit Visualisierung zu arbeiten?*<br><br>*Teamarbeit? Gleichzeitig mit Visualisierung arbeiten?* | *Zunächst Spreadsheet, dann Visualisierung, dann Quellcode; Quellcode in Visualisierunganzeigen lassen*<br><br>*Visualisierung einmalig zur Einführung – später mit höherem Detailgrad für andere Zwecke (Abschätzung, womit bearbeitende Komponente zusammenhängt?)*<br><br>*Teamarbeit hilfreich (AR?)* | |
| Glauben Sie, dass Interaktionsmöglichkeiten unterstützen bei Einarbeitung oder eher hinderlich sind? *(Bspw. bestimmte Dinge auswählen und Infos anzeigen lassen)*<br><br>Fallen Ihnen bestimmte Interaktionsmöglichkeiten und Funktionen der Visualisierung ein, die Sie bei der Einarbeitung hilfreich fänden? | *Wo liegt Benefit gegenüber „Film“, der nacheinander bestimmte Abschnitte zeigt; Individualität des Prozesses der Einarbeitung?*<br><br>*Erinnern Sie sich nochmal an das schrittweise Vorgehen bei der Einarbeitung: Was könnte Sie bei den einzelnen Schritten unterstützen?*<br>*Welche Informationen sollten flexibel angezeigt werden können?* | *Persönliche Vorgehensweise sinnvoll!*<br><br>*Bestimmte Dinge anzeigen lassen, Wechsel der Ebenen durch Zoom, bestimmte Dinge ausblenden Suchfunktion, Abhängigkeiten anzeigen* | |

| | | | |
|---|---|---|---|
| Wenn Sie wählen könnten, ob eine Abstrakte Darstellung (Kugeln und Verbindungen) oder eine metaphorische Darstellung für die Visualisierung gewählt wird (bspw. Städte) – was würde bei der Einarbeitung in eine neue Software am ehesten nützen?<br>*(ggfs. Beispiele zeigen)* | *Warum? Metapher ablenkend oder vereinfachend? Generell das eine dem anderen überlegen oder nur im konkreten Anwendungsfall der Einarbeitung?* | *Vor allem bei Einarbeitung und Überblicksgewinn ist Metapher hilfreich* | |
| Wenn Ihnen dabei zur freien Wahl stünde, ob Sie sich die Visualisierung zur Einarbeitung in VR, AR oder am PC anschauen, was würden Sie wählen? | *Warum? Schon mal mit VR/ AR gearbeitet? Was sind Vor-/Nachteile von VR/AR/Computer?* | *AR – Team?, VR – Konzentration?, Computer – Notizen und weniger Aufwand der Vorbereitung?* | |
| **5) Abschluss** | | | |
| Was sind kurz gefasst Ihrer Meinung nach die wichtigsten Anforderungen an eine Softwarevisualisierung zur Einarbeitung in ein neues Softwareprojekt?<br><br>Haben Sie noch weitere Anmerkungen zu einer der gestellten Fragen?<br><br>Haben Sie noch irgendwelche Fragen zum Interview, zur Auswertung oder zur Anforderungsanalyse generell?<br><br>Herzlichen Dank für Ihre Teilnahme. | | | |

# Anhang B: Übersicht Softwarevisualisierungen

Um über die bisherige Softwarevisualisierung der Abteilung hinaus einen Einblick über Darstellungsmöglichkeiten von Softwarevisualisierungen zu erhalten, wurde diese Übersicht erstellt. In zwei Interviews gehen die Interviewten auf die einzelnen Möglichkeiten ein. In diesen Transkripten wird in Klammern angegeben über welche der Darstellungen gesprochen wird. Die Originalpaper, aus denen die Visualisierungsbeispiele entnommen wurden, sind hier aufgelistet:

**Abbildung a:** Marcus, A., Feng, L. & Maletic, J. I. (2003, June). 3D representations for software visualization. In *Proceedings of the 2003 ACM symposium on Software visualization* (S. 27-37). ACM.

**Abbildung b:** Caserta, P. & Zendra, O. (2010). Visualization of the static aspects of software: A survey. *IEEE transactions on visualization and computer graphics, 17*(7), 913-933.

**Abbildung c:** Balzer, M. & Deussen, O. (2007). Level-of-detail visualization of clustered graph layouts. In *6th International Asia-Pacific Symposium on Visualization* (S. 133-140). IEEE.

**Abbildung d:** Holten, D. (2006). Hierarchical edge bundles: Visualization of adjacency relations in hierarchical data. *IEEE Transactions on visualization and computer graphics, 12*(5), 741-748.

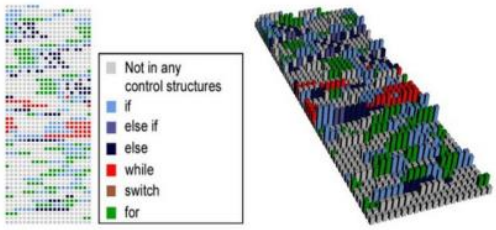
**Abbildung e:** Telea, A. & Voinea, L. (2004, September). A framework for interactive visualization of component-based software. In *Proceedings. 30th Euromicro Conference, 2004.* (S. 567-574). IEEE.

**Abbildung f:** Yang, H. & Graham, H. (2003). Software Metrics and Visualisation. *University of Auckland, Tech. Rep.*

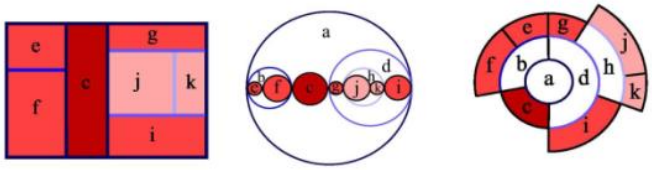
**Abbildung g:** Panas, T., Berrigan, R. & Grundy, J. (2003, July). A 3d metaphor for software production visualization. In *Proceedings on Seventh International Conference on Information Visualization, 2003.* (S. 314-319). IEEE.

# Beispiele Softwarevisualisierungen

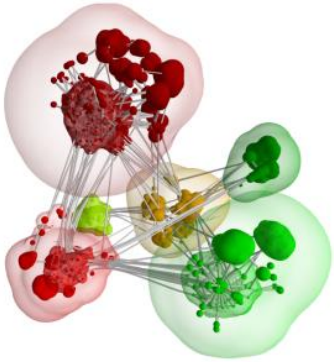
a)



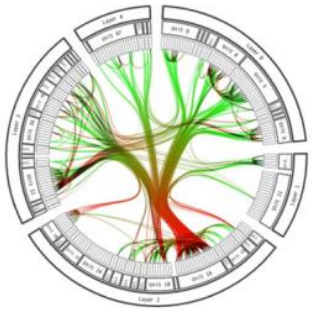

b)




c)

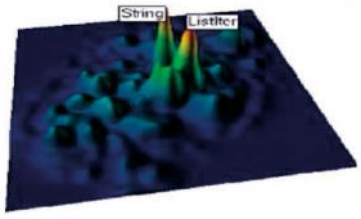


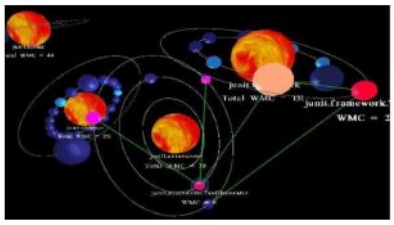
d)



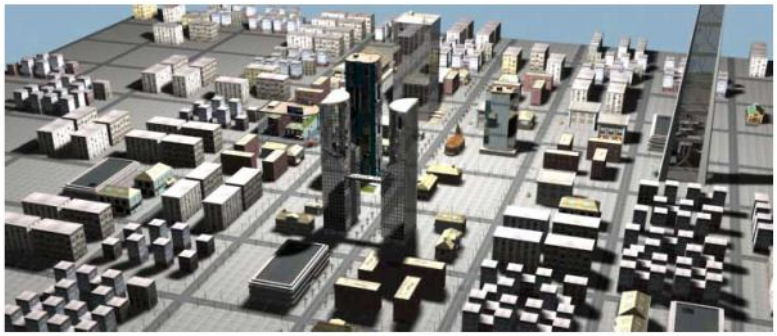
f)

e)



g)



## Anhang C: Transkribierte Interviews

Um die Anonymität der Interviewten zu gewährleisten, wurden bei den hier veröffentlichten Transkripten, die ersten Fragen zur Person und zum beruflichen Werdegang nicht abgedruckt. Auch im weiteren Verlauf der Interviews wurden die Stellen ausgelassen, die eine genaue Zuordnung auf die interviewte Person erlauben. Diese Stellen sind durch folgendes Zeichen gekennzeichnet: [...]

## Interview 1

[…]

I: Und hast du schon mal mit Softwarevisualisierungen gearbeitet?

B: Gearbeitet nicht wirklich, ich hab mir mal die Islandviz zeigen lassen, als HoloLens Variante. Und sonst hatte ich letztens bei einer Bachelorarbeit zu Visualisierung von Neuronalen Netzen mitgemacht. Ansonsten zu Visualisierungen nicht wirklich.

I: Auch nicht am PC irgendwann mal Visualisierungen von Software genutzt? Weil das ist ja nicht nur in VR oder AR möglich.

B: Ich überlege gerade, aber ansonsten zur Visualisierung eigentlich wenig, das ist alles rein textbasiert was ich da gemacht habe.

I: Okay alles klar, aber habt ihr darüber mal irgendwas gelernt in der Uni oder hast du darüber gelesen? Kannst du dir vorstellen was der Nutzen davon sein könnte? Oder ist das für dich etwas ganz neues?

B: Was ich kenne sind so UML-Diagramme. Oder generell Diagramme die die Softwarearchitektur beschreiben. Und da habe ich beim Einarbeiten in RCE auch gemerkt, dass aus der reinen textuellen Beschreibung, die man dann als Code vor sich hat, nicht die komplette Struktur hervor geht. Die muss man sich dann mental selbst zusammenbauen.

I: Aber gab es dafür ein Diagramm bei RCE?

B: Nein. Gibt es nicht. Also es kann sein, dass es Diagramme von Teilen gibt. Also ist es auch nicht für die gesamte Architektur. Und wenn, dann sind die Diagramme auch per Hand gemacht und wahrscheinlich auch outdatet. Also haben nicht mehr so viel damit zu tun, was tatsächlich Stand der Dinge ist.

I: Du hast quasi bei RCE nach solchen Diagrammen gesucht? Hast du bei anderen nachgefragt?

B: Ich hatte mich im Wiki umgeschaut. Wir haben ein Wiki für die Entwicklung und ein Entwicklerguide und in den beiden war jetzt nichts, was wirklich geholfen hätte. Also es gab immer Ansätze. Aber entweder outdated oder nicht das, was man jetzt gerade braucht.

I: Okay das passt jetzt nämlich ganz gut zu meiner nächsten Frage. Das wäre, ob du dir mal gewünscht hättest, dass es eine Visualisierung gibt, anstatt nur mit Quellcode zu arbeiten.

B: Ich denke gerade für den allerersten Einstieg wäre es vielleicht ganz praktisch halt auch wirklich auf einem relativ hohen Architekturniveau. Dass man sieht, welche Teile es gibt, welche Bundles und vor allem auch (... ) also die Bundles hängen ja voneinander ab und die verschiedenen Services, die auch in RCE implementiert sind. Und da einfach mal so ein erster Überblick welcher Service benutzt welchen anderen Service. Und daraus sollte man dann wahrscheinlich direkt schon sehen können, was sind so die zentralen Services die von sehr vielen Sachen genutzt werden.

I: Dann hast du ja jetzt gerade gesagt die Visualisierungen seien eher für so einen groben Überblick. Das heißt du findest auch Aufgaben, für die die Visualisierung gar nicht geeignet wäre, sondern einfach der reine Quellcode? Also wenn du dich eingearbeitet hast würdest du sagen, hast du dann die Architektur in deinem Kopf?

B: Das wahrscheinlich schon, ja. Ist halt auch die Sache, relativ viel Arbeit an RCE ist Bugfixing. Und da geht es dann wirklich um den einzelnen Programmcode. Das sind teilweise ganz diffizile Unterschiede in den einzelnen Programmzeilen, die auf falschen Annahmen beruhen und deshalb irgendwelche Bugs verursachen.

I: Und da musst du nicht vorher wissen, wie der Zusammenhang ist, wenn du dort etwas veränderst, mit der gesamten anderen Software?

B: In einzelnen Programmzeilen, wenn man die erst mal gefunden hat, dann wahrscheinlich nicht. Weil wenn man die erst mal gefunden hat, weiß man ja auch, wie man da hingekommen ist und welche anderen Zeilen das beeinflusst. Wenn man bestimmte kleine Funktionen im Programm hat, kann man sich auch in der Entwicklungsumgebung anzeigen lassen, welche Funktionen diese Funktion aufruft und umgekehrt welche Funktionen aufgerufen werden. Gerade bei so zentralen Funktionen sind das teilweise Listen von 20, 30, 40 Funktionen, die damit verbunden sind. Und da denke ich dann auch wieder, dass da eine Visualisierung vielleicht schon überfordert wäre. Wenn man da ein kleines Quadrat hat auf welches 40 andere dann eingehen, wüsste ich jetzt nicht wie das dargestellt werden könnte.

I: Aber du denkst, dass die Visualisierung überfordert wäre oder, dass dir das quasi durch die Visualisierung nicht schöner dargestellt werden könnte, also dass du auch nichts mit der Visualisierung anfangen könntest?

B: Sagen wir mal so, ich denke ich wäre mit einem textuellen Überblick in diesem Fall schneller.

I: Alles klar. Das klang jetzt alles so, als ob RCE auch die erste große Software, in die du dich einarbeitest?

B: Ja auf jeden Fall in dieser Größenordnung.

I: Und davor? Wie häufig kommt so etwas vor? Und was denkst du wie häufig kommt das noch vor, dass du dich beispielsweise im Rahmen deiner Arbeit hier nochmal in ein neues Softwareprojekt einarbeiten musst?

B: Im Rahmen der Arbeit hier im DLR wahrscheinlich sehr, sehr selten bis gar nicht mehr. Einfach weil RCE auch ein sehr lang laufendes Projekt ist und es auch nicht absehbar ist, dass das irgendwie in Zukunft durch etwas anderes ersetzt werden würde.

I: Und im Laufe deines Studiums hast du ja gesagt, hättest du auch schon ab und zu ein bisschen Softwareentwicklung gemacht. Was würdest du sagen, wie oft hast du dich da schon in etwas Neues eingearbeitet?

B: Ich schätze so im Schnitt etwa einmal im Semester ungefähr. Allerdings war es da auch so, dass es nie so richtig ernsthafte Softwareprojekte waren, in die man sich eingearbeitet hat. Das war immer im Rahmen von irgendwelchen Übungen und dann waren die Projekte gerade so gestrickt, dass man ein Gerüst hat, in das man dann seine eigene Arbeit einbinden kann. Das war schon stark darauf hingearbeitet.

I: Die Projekte waren also so übersichtlich dass du da nicht das Gefühl hattest, so eine Visualisierung könnte dich da groß bei der Einarbeitung unterstützen?

B:Ich denke nicht. Gerade auch, weil der Fokus bei so Aufgaben im Studium liegt dann ja doch immer darauf etwas selbst noch einzuarbeiten. Und diese ganzen Softwareprojekte die man dort bekommt sind eigentlich nur dafür da, dass man sich auf das konzentrieren kann, was man gerade lernen soll. Und sind extra so gemacht und so erklärt, dass man das mehr oder weniger gut dokumentiert bekommt. Je nach Lehrstuhl und Veranstaltung sieht das natürlich immer etwas anders aus. Aber da liegt halt der Fokus wirklich darauf, dass man selbst etwas machen soll und das Einarbeiten wird einem dort so einfach wie möglich gemacht.

I: Bei RCE, kannst du mir da nochmal erzählen, was du genau genutzt hast, zur Einarbeitung?

B: Ich würde da gerne nochmal einhaken. Mir ist gerade eingefallen, ich hatte ein Projekt im Studium, das war ein Programmierpraktikum, wo wir tatsächlich bestehende Software nehmen und parallelisieren sollten. Also so verändern, dass sie auf mehreren Kernen laufen kann. Und da habe ich doch tatsächlich schon mal Visualisierung genutzt. Da ging es allerdings tatsächlich um das Laufzeitverhalten. Also das waren zigtausend Funktionen, die nacheinander aufgerufen werden. Und die Frage war, welche von den Funktionen verrichtet die meiste Arbeit. Da haben wir uns im Nachhinein anzeigen lassen: Okay diese Funktion ruft diese Funktion auf, die ruft die auf und die ruft die auf. Und achtzig Prozent der Laufzeit werden in der einen Funktion hier verbracht. Also wäre das hier der erste Ansprechpunkt, wo man ansetzen könnte um das zu parallelisieren.

I: Also die Visualisierung lag bereits vor und ihr habt diese am Anfang genutzt?

B: Ja das war ein Tool von Intel, glaube ich, das diese Laufzeitmessungen macht. Da habe ich mich auch das eine mal im Studium in echte Software eingearbeitet. Und da hat halt diese Visualisierung auch sehr, sehr gut geholfen. Die war halt auch echt gut zugeschnitten auf diese eine Aufgabe. Das war schon sehr hilfreich. Sorry für die Unterbrechung.

I: Nein, nein, das ist perfekt. Immer wenn dir noch etwas einfällt zu einem Punkt, kannst du das einfach direkt sagen. Wenn du nochmal daran denkst, wie du dich bei RCE eingearbeitet hast, da hast du dann hauptsächlich mit dem Quellcode gearbeitet? Oder hattest du noch andere Dokumente? Vorher hattest du was vom Wiki erzählt.

B: Ja, es gibt Dokumentation in Form eines Developer Guide, der allerdings auch die ganze Umgebung drum herum beschreibt. Das ist vielleicht auch ganz interessant, weil gerade bei RCE gibt es einmal den Quellcode selbst. Und es gibt die ganzen Prozesse drum herum. So nach dem Motto wenn wir hier den Quellcode haben, wie bauen wir daraus tatsächlich die ausführbare Software? Wenn wir diese gebaut haben, wie veröffentlichen wir die? Wie testen wir das Ganze? Also wie bekomme ich es tatsächlich hin, dass wenn ich etwas geändert habe, dass ich meine Änderungen auch wirklich überprüfen und lokal sehen kann. Dieses ganze Drumherum ist im Developer Guide am ehesten beschrieben.

I: Also wie du dann deine Sachen, an denen du arbeitest insgesamt einbauen kannst?

B: Ja genau. Und im Wiki geht es größtenteils um organisatorische Fragen. Also ganz viel Roadmap Planning ist da mit drin. Nach dem Motto, wo sind wir im Moment? Was sind unsere aktuellen Projekte? Wo geht es hin? Und vor allem, wie bekommen wir das hin unsere ganzen Deadlines auf die Leute zu verteilen? Und was so der Ansatz ans Einarbeiten war - also was wir haben ist ein Bugtracker. Also eine große Liste an irgendwelchen Bugs, die halt irgendwann einmal aufgefallen sind. Und da gibt ein paar die als Issues für Einsteiger gekennzeichnet sind, wo man mal irgendwann hingegangen ist und gesagt hat, das Problem liegt hier und hier. Entweder ist es nicht schwerwiegend genug, dass man es jetzt direkt lösen muss, oder man hat gerade keine Zeit das zu machen. Das denkt sich ein Mitarbeiter der schon länger dabei ist, macht eine Kennzeichnung dran - Das ist für neue Mitarbeiter - und dann hat man als neuer Einsteiger so einen Anhaltspunkt. Nach dem Motto - das und das hier läuft schief, gehe mal in den Sourcecode, schaue nach woran das liegt und versuche das zu fixen.

I: Das sind also vorgeschlagene Aufgaben schon mal?

B: Ja genau.

I: Wie viele Leute arbeiten an RCE?

B: Fünf ein halb. Drei Leute die länger dabei sind. […]

I: Und gerade über solche Sachen, wer daran mit arbeitet zum Beispiel, sind das die ersten Sachen über die man sich informiert? Kannst du sehen wer welchen Teil bearbeitet hat oder so etwas?

B: Das ist bei RCE nicht so strikt getrennt, dass irgendwelche Leute bestimmte Teile zugewiesen bekommen. Das ist eher thematisch nach Interesse aufgeteilt. Nach dem Motto, wenn ich jetzt Interesse hätte an der unterliegenden Netzwerkschicht zu arbeiten, dann würde ich das einfach machen und mich bei Fragen mit denjenigen absprechen, die daran noch am ehesten interessiert sind. Einfach auch, weil das Projekt schon fünfzehn Jahre oder so alt ist. Dementsprechend sind schon viele Entwickler, die sich ursprünglich damit beschäftigt hatten, gar nicht mehr hier. Was es gibt ist auch noch eine Historie. Also bei jeder Datei kann man sich anzeigen lassen, wer hat wann welche Zeile bearbeitet. Das trackt allerdings auch nur, wer das Ganze in die Versionsverwaltung eingecheckt hat. Das heißt auch das muss man immer noch ein bisschen interpretieren.

I: Das muss also nicht heißen, dass diese Personen das auch bearbeitet haben?

B: Genau. Also man kann sich das einmal pro Datei anzeigen lassen, wer die zuletzt angepackt hat. Letztens war beispielsweise Jahreswechsel, das heißt wir mussten in allen Dateien so einen Header anpassen. Copyright, ehemals bis 2018 jetzt bis 2019. Das heißt, bei jeder Datei steht jetzt als letzter

Bearbeiter der Kollege, der das gemacht hat. Und da muss man dann einmal drauf gehen. Aha, okay, ne, das ist dieser eine Commit, da steht jetzt Header angepasst, dann schreiben wir den an, der das davor bearbeitet hat, oder den davor. Oder andere Sachen: Wenn wir so verschiedene Entwicklungsstränge haben, die hinterher zusammengeführt werden, dann ist das halt eine Person, die diese Zusammenführung macht. Die steht dann als letzter Bearbeiter bei allem möglichen dabei. Da muss man halt schauen, wer das davor bearbeitet hat.

I: Aber generell, würde dir das bei der Einarbeitung auch helfen, wenn man erkennen könnte, wer daran gearbeitet hat? Oder sind das dann spezifische Fragen, wo du erst später darauf kommst, dass du das mit jemandem sprechen musst?

B: Ich denke das würde so beim zweiten oder dritten Schritt der Einarbeitung helfen. Also wenn man jetzt die Einarbeitung nimmt, wie ich sie gemacht habe. Dass man sich also so ein Issue nimmt und diesen versucht zu lösen. Da ist der erste Schritt wirklich so ein allgemeiner Überblick über das ganze Programm zu haben. Den hat denke ich jeder, der schon länger damit arbeitet. Und dann, wenn man ungefähr herausgefunden hat, woran es jetzt liegen könnte, wäre es schon praktisch zu sehen, wer sich jetzt für diesen Teil verantwortlich fühlt. Wie gesagt formell verantwortlich ist halt keiner.

I: RCE hat ja eine Benutzeroberfläche am Ende. Hast du dir das als erstes angeschaut oder hast du dir das überhaupt angeschaut?

B: Das hatte ich mir tatsächlich schon zur Vorbereitung auf die Bewerbungsgespräche angeschaut. Da auch mal ein bisschen rumgespielt.

I:Das wäre also schon auch der erste Schritt zur Betrachtung einer Software, wenn es eine Oberfläche gibt, dass man sich anschaut, wie das überhaupt aussieht und wie man am Ende damit umgeht?

B: Genau, alleine auch schon weil der erste Schritt alleine ist, das ganze zum Laufen zu bringen, damit man auch Änderungen machen kann, und da ist der erste Test: Wenn es erst mal startet und nicht sofort wieder abstürzt hat man schon mal irgendetwas richtig gemacht. Also das kommt dann auch automatisch beim ersten Einarbeiten, beim ganzen Aufsetzen der Software.

I: Danach schaust du dir als allererstes den Quellcode an? Was versuchst du als erstes herauszufinden? Die Hauptkomponenten oder Module? Oder du hast ja vorhin erzählt, dass Issues schon gekennzeichnet wurden, sind diese dann das Erste was du dir anschaust?

B: Ja genau, weil ich RCE eben schon gekannt habe und mehr oder weniger wusste, was man damit anfangen kann. Also das Erste, was ich dann gemacht habe, ist zu versuchen die Issue nachzuvollziehen. Da steht dann halt, wenn man auf den und den Button klickt und das und das Fenster aufgeht, dann ist da ein Tippfehler in dem und dem Text. Um mal etwas ganz einfaches zu nehmen. Und der allererste Schritt ist dann wirklich nachzuvollziehen, okay, ich habe das jetzt zum Starten bekommen. Dann drücke ich mal auf den Button und schaue den Text an und aha da ist wirklich der Tippfehler. Dann würde ich wirklich daran arbeiten den einen konkreten Bug dann zu fixen. Wenn das wirklich nur ein Tippfehler ist, kann man den ganzen Programmcode durchsuchen nach dem einen Wort. Das wird ja dann hoffentlich nur einmal vorkommen. Das sollte man dann relativ schnell finden. Aber dabei schaue ich auch immer bisschen links und rechts. Wo sind wir hier gerade? Was macht die Programmzeile, die ich gerade sehe? In welchem Kontext steht das Ganze? Und dabei Erfahrungen einbeziehen: beispielsweise die graphische Oberfläche von RCE ist nicht komplett selbstgeschrieben, sondern wir benutzen Frameworks, die von Eclipse kommen. Ich habe durchs Studium bisschen Erfahrung bekommen, wie so ein Framework logischerweise aufgebaut sein sollte. Ich habe dann eine Erwartung wie der Text entstehen könnte, also was die Zeilen sind, die sie dann tatsächlich auf dem Bildschirm anzeigen. Und dann versuche ich abzugleichen mit meiner Erwartung, was ich jetzt im Quellcode sehe. Ob das so ist, wie ich mir das vorgestellt habe, oder ob da etwas ganz merkwürdiges ist. Und dann in den ersten Tagen habe ich mich sowas dann mal irgendwann gefragt: "Ich habe das und das gesehen, warum machen wir das so und so? Das ist nicht so, wie ich das erwartet habe."

I: Wenn du dir vorstellst, dass du dich jetzt nochmal in ein neues Softwareprojekt einarbeiten müsstest oder du würdest nochmal neu hier anfangen. Würdest du dann sagen, dass es die beste Art der Einarbeitung ist, dass man sich direkt mit einem konkreten Problem beschäftigt? Oder würdest du als ersten Schritt -weil du vorhin erwähnt hast, dass die Softwarevisualisierung zum Überblick geben gut

ist - dir nur einen Überblick verschaffen ohne konkret ein Problem zu haben und nach einer Lösung zu suchen?

B: Ich denke das kommt auf das Projekt an. Gerade bei RCE. Ich habe ja schon von den Bundles gesprochen. Das sind bei RCE jetzt schon relativ viele, so an die 220 oder so, da weiß ich nicht, ob eine Visualisierung auf einer Architekturebene, die Bundles und Abhängigkeiten zeigen würde, da wirklich viel helfen wird. Gerade weil die Abhängigkeiten dazwischen über die Jahre auch ein bisschen verschwommen sind. Ich denke da gibt es kein klares Konzept mehr dahinter, welche Bundles welche anderen Bundles benutzen. So eine Architekturvisualisierung würde im Moment zeigen, dass die Architektur ziemlich Kraut und Rüben ist. Wenn man eine Software hat, die kleiner ist, oder wo funktionale Komponenten besser und strikter voneinander getrennt sind, dann denke ich würde das schon eher helfen, dass man eine grobe Übersicht bekommt.

I: Aber um zu verstehen, dass die Architektur und die Zusammenhänge Kraut und Rüben sind, würde da nicht auch schon helfen können? Dass man sieht wie groß das ist und, dass alles irgendwie mit allem zusammenhängt?

B: Das wäre auf jeden Fall interessant. Also ich vermute, dass es so ist. Es kann natürlich auch sein, dass sich irgendwie zwei, drei, vier große Blöcke herausbilden. Wo man dann direkt sagen kann, okay, dann können wir die an dieser Stelle schon einmal voneinander trennen. Dann hätten wir wenigstens schon mal nur zwei große Felder voller Kraut und Rüben.

I: Also du würdest dir als erstes anschauen, wie die einzelnen Module zusammenhängen. Mal angenommen, es wäre eine Software bei der das nicht alles Kraut und Rüben wäre, dann würdest du dir die Hauptkomponenten als erstes anschauen oder wie würdest du da vorgehen?

B: Ich würde wahrscheinlich am ehesten vom Programmeinstiegspunkt ausgehen. Bei RCE hat man ja die Graphische Benutzeroberfläche. Da würde ich zuerst mal verstehen wollen, wo wird diese Benutzeroberfläche definiert? Und wie wird sie gestartet? Nach dem Motto, das ist das Erste, das ich sehe und ich möchte herausfinden, wie komme ich wirklich dahin, dass ich das jetzt sehe. Wie gesagt, wenn ich kein konkretes Problem habe, das ich in dem Moment sehen sollte. Und sonst, die hauptalgorithmischen Komponenten, was halt jetzt bei RCE die Verteilung von Daten übers Netzwerk ausmacht oder was bestimmt, welcher Code wo aufgerufen wird, würde ich eher im zweiten Schritt mir anschauen. Ich vermute mal, so ziemlich jede Software hat einen Softwarekern. Der wird allerdings relativ viele verschiedene Einstiegspunkte haben, weil wenn man einen guten Kern hat, möchte man diesen von verschiedenen Stellen aus benutzen. Das heißt, es gibt wahrscheinlich 18 verschiedene Einstiegspunkte für verschiedene Nutzungsszenarien. Und da wäre es dann sehr, sehr praktisch zu haben, okay ich sehe bei RCE ich kann auf den grünen Pfeil drücken und dann auf okay und dann läuft mein Workflow los. Das dann im Code nachvollziehen zu können und das ab dem Punkt: ich starte RCE und sehe das Feld vor mir.

I: Du würdest quasi von der Nutzerperspektive der Software am Ende gehen und daran versuchen den Code nachzuvollziehen.

B: Ja genau, einfach aus der Idee heraus, wenn ich etwas ändern möchte, sollte das am Ende einen sichtbaren Effekt für den Nutzer haben. Oder auch nur für den Entwickler aber irgendwie soll man sehen können, dass sich etwas verändert hat. Wenn sich nichts ändert, hätte ich die Änderung nicht machen müssen. Deshalb ist es denke ich sehr wertvoll am Anfang nachvollziehen zu können, wie spiegeln sich Änderungen in der Nutzererfahrung wieder? Wenn man also keinen konkreten Einstiegspunkt hat, wäre der Einstiegspunkt des Nutzers denke ich der sinnvollste.

I: Aber ansonsten würdest du anhand deiner Aufgabe vorgehen. Also du schaust nach den Issues und versuchst diese zu lösen.

B: Ist natürlich auch eine andere Sache, wenn man einen Internetservice hat, der keine graphische Benutzeroberfläche hat, sondern einfach eine API *(Anwendungsprogrammierschnittstelle)* für andere Nutzer zur Verfügung stellt. Da gibt es dann wahrscheinlich nicht den einen konkreten Einstiegspunkt. Aber da würde ich wahrscheinlich ein Hello-World Beispiel nehmen, das zeigt wie man diese API benutzt und da dann durchgehen wie diese Anfragen bearbeitet werden.

I: Dann haben wir jetzt eine Weile darüber gesprochen, wie du anfängst dich in eine Software einzuarbeiten. Gibt es eine Art Zielzustand, den du vor Augen hast, wenn du sagst du arbeitest dich ein?

B: Gerade im konkreten Fall RCE würde ich sagen, man wird nie fertig. Ich habe letztens noch mit einem Kollegen gesprochen, der ist über sieben Jahre dabei und ich hatte ihn nach einer konkreten Codestelle gefragt und er meinte, die hätte er auch noch nie gesehen. Wüsste also auch nicht wie das alles zusammenhängt. Daraus entwickelt sich also auch die de facto Spezialisierung, die wir haben. Dass sich manche Leute um bestimmte Teile vom Code kümmern. Das heißt aber auch, dass nie einer so richtig den kompletten Überblick über die Gesamtarchitektur hat. Ich denke am ehesten hat man noch damit zu tun, wenn man sich wirklich mit dem Bauprozess beschäftigt. Also wie bekomme ich aus meinem Code wirklich das ausführbare Programm raus? Dann muss man ja so eine grobe Übersicht haben, was sind die verschiedenen Komponenten, die jetzt zusammengebaut werden. Das geht dann natürlich zu Lasten der Detailkenntnisse in den einzelnen Komponenten. Zu einzelnen Codestellen kann man dann Codearchitekten auch weniger befragen.

I: Wenn du jetzt sagst, dass du auf eine Stelle spezialisiert bist und es niemanden gäbe, der den Gesamtüberblick hat, würde dann eine Visualisierung der Architektur auch nach dem ersten Kennenlernen der Software helfen können, damit alle den gleichen Stand der Architektur haben? Kann man sich damit möglicherweise besser über die Software unterhalten?

B: Ich denke schon, dass es helfen würde. Nicht nur zur Kommunikation, sondern dann auch in der weiteren Arbeit. Ich habe hier jetzt schon mal ungefähr die Stelle herausgefunden, an der ich arbeiten möchte. Auf welche anderen Komponenten greift die jetzt tatsächlich zu? Sowas visuell dargestellt zu haben - ich meine es sind ja nicht immer nur einzelne Komponenten an denen man arbeiten muss - es ist ja teilweise auch ein Zusammenspiel von drei, vier, fünf Komponenten. Da den Kontext der Komponenten dargestellt zu bekommen, das denke ich würde schon helfen.

I: Vorhin hast du ja an einem anderen Punkt bereits die Historie der Software erwähnt. Ist das für die Einarbeitung wichtig oder schaust du dir zunächst nur den Status Quo an?

B: Ich würde mir als erstes nur den Status Quo anschauen. Die Historie wäre dann eher in einem zweiten Schritt interessant. Wenn ich irgendwo tatsächlich nicht weiterkomme, wäre es praktisch zu sehen, wer hat sich damit beschäftigt.

I: Aber auch wirklich wer sich damit beschäftigt hat oder wie sich das entwickelt hat?

B: Größtenteils eher wer sich damit beschäftigt hat, weil das dann derjenige ist, den ich direkt ansprechen kann. Dann habe ich einen ersten Anhaltspunkt. Die Frage wie hat sich das entwickelt ist teilweise ganz interessant, wenn man schon ein bisschen mehr Ahnung von der Architektur hat, dann durch den Code geht und nicht nachvollziehen kann warum einzelne Codezeilen jetzt da sind. Dann kann man ab und zu sich auch anschauen wie sich eine Datei jetzt entwickelt hat und dann auch nachvollziehen, warum Sachen mit reingekommen sind. Und in den allermeisten Fällen sind sie einfach noch da, weil sie niemand gelöscht hat.

I: Wie kannst du das im Moment nachvollziehen? Über den Code?

B: Ja genau, man kann sich die Historie anzeigen lassen. Dann wird gesagt die Programmzeile ist an dem Datum von dem und dem reingekommen mit dieser Commit-Message. Und hoffentlich ist dann in dieser Commit-Message ein Hinweis drin, warum das da ist. Was bei uns immer noch ganz praktisch ist, in den Commit-Messages haben wir es halt zwingend mit drin, dass eine Issue-ID drin sein muss. Das heißt, jede Änderung muss mit einer Issue in unserem Bugtracker verknüpft sein. Dadurch kann man sagen, das und das war das Problem und diese Codezeile soll was mit diesem Problem zu tun haben. Da kann man dann überlegen, wenn man die Zeile raus nimmt, ob das Problem dann wieder zurückkommt.

I: Kann man Dinge, die gelöscht wurden dann auch noch über den Code nachvollziehen?

B: Wenn ich konkret danach suche, dann schon. Diese Codehistorie zeigt dann auch an: Vor zwei Jahren sah diese Datei so und so aus und im Vergleich zur aktuellen Version fehlen die und die Zeilen. Oder auch einfach diese Datei wurde mit jenen Änderungen angepackt und dabei sind folgende Zeilen rausgeflogen.

I: Das wird aber alles erst relevant, wenn du dich spezifisch an eine Aufgabe setzt? Davor würdest du die Historie dir erst mal noch nicht anschauen oder?

B: Wahrscheinlich noch nicht. Es sei denn - was ich eben auch gemeint hatte - wenn ich mir ein bisschen den Kontext anschaue. Wenn ich irgendwas mache, und mir da dann irgendetwas auffällt, was nicht so ist, wie ich mir das vorgestellt hätte. Dann könnte man da halt auch über die Historie gehen. Einmal um denjenigen herauszufinden, den man ansprechen kann. Und andererseits kann man vielleicht so schon rausfinden, warum die Sachen so sind, wie sie sind.

I: Wenn wir nochmal einen Schritt zurück machen von diesen Details. Wenn man sich jetzt erst mal einen Überblick über die Architektur verschafft, hast du eine grobe Einschätzung, wie lange du dich damit beschäftigen würdest? Also geht das dann um so fünf Minuten, in denen du dir anschaust, wie das zusammenhängt oder würdest du dich vier Wochen damit beschäftigen?

B: Also für mich persönlich würde ich sagen, wenn ich mir so eine Architekturvisualisierung anschaue, dann habe ich ja irgendein Ziel. Beispielsweise als ich mir die IslandViz angeschaut habe, wollte ich eigentlich nur sehen, wie wird das über AR in die Umgebung eingebunden. Das war dann eine Sache von fünf bis sechs Minuten. Das war halt auch auf einer Messe und ich konnte mir das gerade zeigen lassen, als niemand an unserem Stand war. Ansonsten, wenn ich ein bestimmtes Ziel habe, dann würde ich schon etwa 20 - 30 Minuten damit verbringen, mir das anzuschauen, um auch die nächsten Schritte herauszufinden. Im Moment bei RCE würde ich so eine Visualisierung am ehesten nutzen um meine aktuelle Vermutung, dass die Architektur Kraut und Rüben ist zu unterfüttern. Allerdings bringt das ja auch nichts wenn man einfach nur sagt, dass alles chaotisch ist und besser sein müsste. Man müsste einfach da irgendwie hinkommen, wie bekommen wir das besser gemacht?

I: Man kann ja die Visualisierung auch so gestalten, dass nicht nur die Architektur zu sehen ist. Beispielsweise könnte man sich dort auch den Code anzeigen lassen.

B: Ich würde das wahrscheinlich getrennt halten. [..] Ich merke auch, dass ich bei so einem Textinterface auch einfach schneller im Bearbeiten bin. Zum wirklichen Arbeiten würde ich also immer aufs Textinterface zugreifen.

I: Du hattest ja eben schon von der AR-Visualisierung gesprochen und einen Zeitrahmen von etwa 20 Minuten bis 30 Minuten angegeben. Würdest du bei einer solchen Einarbeitung dann auch Pausen machen um gegebenenfalls auch Dinge zu notieren? Also ist es wichtig, die Einarbeitung zwischendurch dahingehend zu unterbrechen?

B: Ja zum Beispiel ganz konkret: Ich habe zwei Whiteboards in meinem Büro und die nutze ich auch ziemlich ausführlich wenn es um Einarbeitung geht. Um mir beispielsweise ad hoc kleine Diagramme zu basteln. Die Klasse greift auf die Klasse und so weiter. Und da merke ich ja schon, wenn nicht am Whiteboard dann mach ich mir Notizen auf Papier.

I: Das ist spannend. Du versuchst dir also selbst die Visualisierung zu machen?

B: Ja genau. Aber was ich da an Erfahrung mit möglichen Tools gemacht habe, die eine Visualisierung machen, ist sehr, sehr häufig, dass ich mir anzeigen lassen kann, welche Klasse Links zu welcher anderen hat. Allerdings ist das nicht priorisiert. Das heißt es zeigt an: die Klasse greift auf einen Fileservice zu, der Dateien öffnet und schließt, auf einen Datumsservice, der das aktuelle Datum zurück gibt und auf irgendetwas wichtiges, wie ein Workflow Execution Service. Wenn ich mir das jetzt automatisiert anzeigen lassen würde, würde ich einfach eine Liste bekommen, das sind die drei Klassen, auf die das zugreift. Es müsste also priorisieren können, was die relevanten Klassen sind, auf die zugegriffen wird. Das mache ich dann halt in einer manuellen Visualisierung schon intuitiv. Wenn also irgendwas Dateservice heißt, das brauche ich mir nicht zu merken, wenn ich mich gerade mit der Workflowausführung beschäftige. Und teilweise ist es ja dann sogar noch so, dass wenn versucht wird etwas zu gewichten, das meistens nach der Idee abläuft, Services zu priorisieren auf die von vielen Klassen zugegriffen wird. Das ist ja an sich logisch, das heißt aber halt, dass so ein Fileservice und so ein Dateservice, die überall gebraucht werden, werden auf einmal priorisiert. Ich meine es gibt ja Visualisierungen auch von großen Architekturen aber da ist es dann häufig so, dass es ein, zwei, drei zentrale Services gibt, auf die dann alle zugreifen. Dann würde man sagen, okay, die müssen unglaublich wichtig sein. Dabei ist es dann meistens nur irgendwas wie das Datum.

I:Wäre es dann sinnvoller die Services vorher raten zu lassen, beispielsweise von den Entwickler*innen selbst?

B:Ich sage mal so, es wäre schön dieses Rating zu haben, wie wichtig sowas ist. Das Problem, dass ich dabei sehe, ist, dass Entwickler eher faul sind. Schon Sachen ordentlich zu dokumentieren ist ein Krampf. Wenn man sie dann noch bitten würde zu raten wie wichtig so etwas ist, das würde vermutlich nicht passieren. Aber es wäre auf jeden Fall interessant. Was ich mir vorstellen könnte wäre es, Anomalien zu highlighten. Beispielsweise: Deine Klasse greift auf diese drei anderen Klassen zu. Die ersten beiden davon werden häufig genutzt, das ist vermutlich nichts für dein spezifisches Problem. Die dritte Klasse, auf die greifen nur diese drei bis vier anderen Klassen zu. Das heißt, das ist dann irgendwie etwas Besonderes.

I: Das könnte dann auf das spezifische Problem hindeuten, klar. [..] [Die Einarbeitung,] ist das eher eine Sache, die man in Einzelarbeit macht, weil man eben auch unterschiedlich vorgeht?

[..]

B: Das denke ich schon. Auch wenn man vom selben Background ist, hat man unterschiedliche Tempi, wie man sich einarbeitet. Und auch unterschiedliche Interessen und Herangehensweisen. Vor allem, wenn man zwei Leute, die sich noch nicht damit auskennen vor dasselbe Problem setzt, würde sich die Ungewissheit vermutlich eher gegenseitig aufschaukeln, weil noch keiner die konkrete Ahnung hat. Was da vielleicht sinnvoller wäre, wäre sich dann zu dritt mit einem dahin zu setzen, der das schon kennt und einem das erklärt.

I: Und wenn der jetzt - in einer idealisierten Welt können wir uns das ja mal vorstellen - 24/7 für dich da wäre. Dann würdest du dich trotzdem am liebsten selbst einarbeiten oder eher zu dieser Person gehen und sagen "erkläre mir alles"?

B: Ich würde mich trotzdem lieber alleine einarbeiten. Ich hätte dann auch wirklich das Gefühl dass mir immer jemand über die Schulter schaut. Aber ich muss auch sagen, dass wir eigentlich bei der Einarbeitung immer einen Ansprechpartner hatten, den man fragen konnte. Man wusste aber eben auch, wenn man wieder weg geht, dass der sich wieder mit seinen eigenen Sachen beschäftigt und das ist mir persönlich lieber.

I: Und hast du das oft genutzt, dass du einen Ansprechpartner hattest, an den du dich immer wenden konntest?

B: Ja, schon. Gerade am Anfang sehr, sehr häufig.

I: Wir haben das gerade im Gespräch immer mal wieder schon mit einfließen lassen aber ich würde gerne nochmal konkreter fragen: Kannst du dir vorstellen, dass gerade bei der Einarbeitung eine Visualisierung nützlich sein könnte, wenn sie gut gemacht ist?

B: Das auf jeden Fall. Wenn die Visualisierung auf das Problem maßgeschneidert ist, das man gerade zu lösen versucht, dann würde das einem auf jeden Fall helfen. Das Ziel ist am Anfang am ehesten Bugfixing. Architekturverstehen so als Hauptbereich der Visualisierung, ist nicht so richtig das Ziel es wäre also eher eine Methode oder Werkzeug um ein bestimmtes Ziel zu erreichen.

I: Dann ist es aber auch immer eine Kombination, wenn ich das richtig verstanden habe? Einfach vorgestellt du könntest beides gleichzeitig haben, Quellcode und Visualisierung, beispielsweise an verschiedenen Bildschirmen. Würdest du beides gleichzeitig nutzen? Oder zuerst mit der Visualisierung arbeiten und dann mit dem Quellcode?

B: Also ich würde es auf zwei Bildschirmen offen lassen, einfach, weil es nicht stört. Gerade zur Navigation durch relativ komplexe Klassenstrukturen würde die Visualisierung helfen, vor allem wenn sie mit dem Code auch verknüpft ist aber sonst, wenn es in dieser Visualisierung auf Codezeilenebenen wäre, denke ich sie wäre zu detailliert. Das würde nicht wirklich einen Mehrwert über den Code hinaus bringen. Und ansonsten hat man dann auch das Problem, die nächste Ebene über den Codezeilen sind dann eigentlich Funktionen. Einzelne Funktionen können in RCE schon mal so 300 bis 500, 1000 Zeilen haben. Das heißt, da ist es dann wieder zu ungenau um nur mit der Visualisierung zu arbeiten.

I: Wenn du dir vorstellen könntest, dass man switchen kann zwischen der Visualisierung, fändest du das gut? Und darüber hinaus, gibt es andere Interaktionsmöglichkeiten, die du dir hilfreich vorstellst, wie sich etwas zusätzlich anzeigen zu lassen oder etwas auszublenden?

B: Gerade wenn das eng in die Entwicklung integriert ist, wäre eine graphische Übersicht gut über die Verbindungen und Abhängigkeiten, die man dann auch nutzen kann um direkt wieder zu navigieren. Ich habe die Erfahrung gemacht, wenn so etwas extern ist, ist das eine sehr große Hürde es zu benutzen. Man denkt sich, ich könnte die Visualisierung anwerfen, aber dann muss ich da erst mal nachschauen und dann wieder im anderen Editor zurück. Dort dann wiederum das finden, was ich gerade in der Visualisierung gesehen habe. Das ist wieder ein bisschen wie das was ich zu Beginn meinte. Ich versuche immer die direkte Verbindung zwischen dem was ich als Nutzer sehe und dem was ich als Programmierer sehe zu finden. Sowas würde ich mir dann auch für eine Visualisierung wünschen. Wenn da die direkte Verbindung da ist, denke ich das könnte gerade für die Navigation auch sehr, sehr hilfreich sein. Einfach auch, das ist vermutlich ein Merkmal der Programmiersprache, in Java hat man es sehr häufig, dass man sehr viele Klassen hat. Wenn man also einen Codestrang nachverfolgt, hat man sehr schnell an die 15, 20, 25 Klassen auf und verliert sich dann ein bisschen in dem Zusammenhang. Wenn so etwas nachverfolgt würde, zum Beispiel warum bin ich gerade in der Klasse hier? Wo bin ich abgebrancht? Was war mein Suchweg bis hierhin? Das müsste natürlich auch in den Editor irgendwie integriert sein. Denn auch das bringt wenig, wenn es in einem anderen Fenster ist und man damit nicht interagieren kann, aber ich denke das würde sehr stark helfen.

I: Dann wäre es auch so, dass wenn du den Quellcode vor dir hast und dann die Visualisierung aufmachst, würdest du dort gerne starten, wo du gerade in deinem Code auch bist?

B: Ja, ich denke, das wäre das sinnvollste. Also ob das jetzt wirklich einzelne Funktionen oder Klassen sind, sei mal dahingestellt aber prinzipiell ja.

I: Wie eine Art Suchfunktion in der Visualisierung? Falls es beispielsweise nicht auf einem Editor läuft, könnte es ausreichen, dass man die Codezeile eingeben kann und dann an der Stelle bei der Visualisierung landet?

B: Ja, das würde sehr stark helfen. Das habe ich auch bei der IslandViz gemerkt. Man hatte dieses tolle Meer voller Inseln und dann kann man im Endeffekt sich random durch die Inseln durchklicken und feststellen: ja das ist ein großes Bundle.

I: Du hast ja jetzt die Visualisierung mit der Inselmetapher mehrmals erwähnt. Generell gibt es bei den Visualisierungen verschiedene Arten. Es gibt mehrere Metaphern, wie die Inseln oder Stadtmetaphern. Oder aber es ist so, wie du dir das selbst vermutlich an die Whiteboards malst. Das wäre dann eher abstrakt dargestellt. Fändest du eine Realweltmetapher hilfreich für das Verständnis?

B: Ich denke eher dass das Abstrakte hilfreicher wäre, weil das wahrscheinlich auch näher an der Aufgabe wäre, die es dann wahrscheinlich gerade zu lösen gilt. Und, gut das ist dann jetzt eine Verallgemeinerung die gar nicht durch Daten unterstützt ist, aber, ich habe so ein bisschen die Erfahrung gemacht, dass gerade Programmierer und Informatiker sich wohlführen im abstrakten Denken. Wenn man mit Software arbeitet hat man halt nie etwas, das man tatsächlich anfassen kann, deswegen ist das für viele Leute, mit denen ich gesprochen habe nichts ungewöhnliches relativ abstrakt zu denken.

I: Eine letzte Frage noch zur technischen Umsetzung, einfach weil du ja auch schon erwähnt hast, dass das hier jetzt schon gemacht wurde mit den Visualisierungen in VR und AR. Wenn du an den konkreten Fall denkst, du müsstest dich in eine neue Software einarbeiten und hättest alle technische Möglichkeiten zur Verfügung. Also VR, AR und auch einfach am PC. Was würdest du wählen?

B: Wahrscheinlich am PC. Also VR und AR haben eben diesen Sense of Space, dass man halt sich irgendwie um ein 3D-Objekt bewegen kann und es besser manipulieren kann. Mir fällt jetzt keine Information ein, die von dieser 3D Einstellung so stark profitieren würde, dass ich sagen würde, dafür rolle ich von meinem PC zurück, setze ein Headset auf und schaue mir alles an. Das ist ein ziemlich krasser Kontextwechsel. Das wäre eine andere Sache, wenn man direkt in VR entwickeln würde, aber das ist vermutlich eine ganz andere Baustelle. Deshalb denke ich, dass diese relativ geringe Hemmschwelle das am PC zu benutzen der größte Vorteil von einer Visualisierung am PC wäre.

I: Damit haben wir auch schon die Hauptfragen, die ich hatte, durchgesprochen. Am Ende würde ich dich einfach nochmal kurz bitten, ob du mir zusammenfassend sagen kannst, was deiner Meinung nach die wichtigste Anforderung an eine Softwarevisualisierung? Wenn ich dir deine Traumsoftwarevisualisierung entwickeln könnte zur Einarbeitung in eine neue Software, was ist das wichtigste?

B: Eine kontextsensitive Visualisierung Ein Überblick über den Kontext, in dem man sich gerade befindet. Gerade weil irgendwelche Codeeditoren, dadurch dass es halt Text ist, immer nur so fünfzig Zeilen auf dem Bildschirm zeigen können, ist man sehr fokussiert auf den einzelnen bestimmten Punkt. Und man muss sich selbst ein mentales Modell bauen, in welchem Kontext bewege ich mich gerade. Wenn man so etwas sinnvoll visualisieren könnte, um das einmal dann aus dem Kopf raus zu bekommen, dass man es nicht im Hinterkopf behalten muss und auf der anderen Seite um auch zu bemerken, was man gerade in seinem mentalen Modell übersehen hat. Das wäre glaube ich so die ideale Visualisierung. Und dabei natürlich auch, deswegen auch kontextsensitiv, darf die Visualisierung einen dabei nicht mit Infos überladen. Das ist das, was ich meinte mit, wenn die Funktion 33 Mal eine Funktion aufruft, was eigentlich nur eine Zeile auf den Bildschirm ausgibt. Gut, dann wird die Zeile oft aufgerufen, aber die ist mir persönlich egal, weil so Bildschirmausgaben sind wahrscheinlich nur zum Debuggen da. Da müsste man wahrscheinlich schon sehr intelligent vorgehen um das zu priorisieren. Je nach Kontext ist die Priorisierung dann auch wieder unterschiedlich. Zusammengefasst wäre also die ideale Visualisierung die, die genau das macht, was ich gerade brauche und nichts anderes.

I: Alles klar, das wäre es von meiner Seite, außer du hast noch irgendwelche Anmerkungen zu einem der Themen oder sonst noch Fragen zum Interview, zur Auswertung oder ähnlichem?

B: Nein, sonst alles gut.

I: Super, dann vielen Dank, dass du dir so viel Zeit genommen hast.

## Interview 2

[…]

I: Alles klar. Hast du selbst auch schon mal mit Softwarevisualisierung gearbeitet?

B: Ein bisschen. Ich hab ein bisschen mit Software Analytics, jetzt mal vor einem Jahr ein bisschen, im 2D-Bereich gearbeitet und mir das angeguckt. Ich mag mehr diese Analysen, die auch diese zeitlichen Komponenten mit reinnehmen. Weil du dann eben auch so eine Interaktion zwischen den Entwickelnden da drin sehen kannst. Also was ich schon mal gemacht habe, das hab ich zwar auch mehr nachempfunden, aber dann auch bisschen verbessert, dass es auch so Renamings und Version Control abkann, ist so eine Knowledge Map. Also so eine 3D Map, mit den Bubbles und so weiter, und dann eingefärbt, ist das entweder gleichverteilt das Wissen oder ist das eher eine Person, die dann gehighlighted wird. Da kann man dann ganz interessante Sachen machen, wie, was passiert denn, wenn derjenige weg ist? Wo sind die Übergabepunkte und so? Da steckt halt viel Zeug drin, das man sehen kann. Weil so dieser organisatorische Einfluss auf Softwarequalität doch sehr hoch ist. Da passiert halt relativ viel, wenn die Organisation drum herum nicht passt. Also wenn du Teams anders schneidest als es inhaltlich sinnvoll ist, dann hast du natürlich viel mehr Abstimmungen und so einen Kram. Das macht es dann nicht einfacher. Da gibt es aber auch ganz gute Bücher dazu. „Code as a Crime Scene“ und so. Das ist auch von einem, der zwar Softwareentwickler ist aber auch so Psychologie macht. Diese Schnittstelle finde ich übrigens auch sehr interessant. Das ist halt nicht mehr irgendwie: Ja da hast du einen Prozess und ein paar Tools. So statische Codeanalyse zeigt dir ganz viele Findings aber welche von denen sind denn interessant und relevant?

I: Gerade hast du ja schon über die Knowledge Map geredet und auch über die eher organisatorischen Informationen gesagt, dass du Visualisierungen ganz hilfreich findest. Gibt es noch weitere Fälle, wo du beispielsweise bisher nur mit Quellcode gearbeitet hast und dir dachtest, da würde eine Visualisierung jetzt helfen?

B: Nur Hotspotanalyse. Dass da verschiedene Parameter (…) und die sind halt spannend eigentlich passend für das Projekt zusammenzustöpseln. Ich glaube dass es da keinen statischen one size fits all Ansatz gibt. Da gibt es ja diverse Tools, SonarQube und hast du nicht gesehen, und das ist halt so 0815. Sowas muss man anpassen können. Dafür braucht es passende Dashboards. Und diese Parameter hängen halt wirklich vom konkreten Softwareteam ab. Was ist für die gerade wichtig und nicht. Damit es halt auch im Rahmen bleibt. Und so Hotspotanalysen finde ich halt spannend. Also vereinfacht gesagt: Länge und Breite, dann wird es unübersichtlich und dann stellst du noch fest, du hast keine Tests. Dann weißt du auch noch, dass sich das sehr häufig ändert. Du weißt, das ist ein Ding, das du dir angucken musst, die Gegend, wo du am Ende was findest. Aber es ist schon seit zehn Jahren nicht mehr angefasst worden. Dann ist das vielleicht ein Refactoring Kandidat, der für dich uninteressant ist.

I: Also eher so, wer wann an welchem Teil gearbeitet hat?

B:Natürlich, aber plus die Information aus dem Code an anderen Sachen. Der Code selbst hat ja vereinfacht gesagt eine Breite und eine Länge, also Lines of Code, so was ähnliches wie zyklomatische Komplexität. Dann wird es ja nochmal interessant, was sind da für andere Sachen drin. Zum Beispiel könntest du sagen, der Checkstyle hat eine Million Anmerkungen dazu. Dann weißt du die Tendenz ist, du hast einen Indikator, dass das nochmal zusätzlich schwer verständlich sein könnte. Und wir haben keine Tests. Das heißt die werden nicht automatisch über das Stück Code beim Bauen und Testen ausgeführt, dann wird es interessant. Wenn dann die ganze Software aus sowas besteht, dann fange vielleicht doch von vorne nochmal an. Das könnte dann die Schlussfolgerung sein, muss es aber nicht. Das ist halt immer der Punkt. Solche Sachen sind nett, kann man in einer Visualisierung schön hin plotten, aber du brauchst halt immer noch das Kontextwissen. Es gibt keine Schwarzweißaussagen glaube ich, das ist dann immer eine Tendenz. Spannend ist dann eben auch, das ist dann interessant, wenn man schon so einen Hotspot hat, das kommt auch aus diesen anderen Analysen, die ich mir mal so grob angeguckt habe, die ich natürlich nicht alle umgesetzt habe. Aber was halt eben spannend ist: Seit wann hat die Komplexität denn zugenommen? Das hat halt alles nichts mit Einarbeitung zu tun.

I: Darum geht es ja im Moment auch noch gar nicht spezifisch.

B: Ok. Dass du dann halt A siehst, seit wann ist das Ganze aus dem Ruder gelaufen. Und das dann idealerweise begleitend nutzen könntest um das nächste Mal diese Kipppunkte von einem Modul ins "Schlechte" vorher festzustellen, das finde ich ganz interessant.

I: Eine Hotspotanalyse ist dann eher gedacht um sich einen ersten Überblick zu verschaffen, also, wo muss ich ansetzen. Gibt es dann Aufgaben, für die einfach die reine Arbeit mit dem Quellcode geeigneter ist?

B: Danach reicht es eigentlich. Hotspot wäre jetzt hier der Fall wir wollen halt Code verbessern. Wir wollen wissen wo wir anfangen sollen. Zum Reinkommen ist natürlich so eine Übersicht, also halt die Grobmodule und wie hängen die zusammen, also eher ein Architekturblick sozusagen, spannend. Und dann weiß ich halt, okay, das ist das Package, das kümmert sich um das Wegschreiben einer Datenbank oder so. Und da ist genau der Punkt dann, wo ich in eine IDE wechseln würde. Nicht tiefer. Insbesondere nicht manuell irgendwie aufgedingst. Weil dann ist man in der IDE einfach schneller, weil das dein übliches Werkzeug ist als Softwareentwickler. Du brauchst dann nicht mehr eine Modellsicht von irgendeiner Klasse, die dasselbe Level ist. Was dir hilft ist Struktur. Denk dir RCE mit was weiß ich wie vielen Klassen - alles in einer Liste. So Hierarchien, dieses Chunking wird dann halt besser durch so eine Struktur unterstützt. Und die würde ich dann halt gerne, wenn es Richtung Einarbeitung geht als erstes sehen. Und dann habe ich vielleicht eine kleine Aufgabe. Ich soll irgendwie in der Persistenzschicht fixen, dass ich dann halt schnell weiß, was ist Persistenzschicht, wo kommt das hin und dass ich dann halt relativ schnell, meinen Einsprung finde, wo muss ich denn meine IDE aufmachen und suchen. Dann musst du halt aber immer wissen, Einarbeitung für wen? Für einen, der irgendwie Java entwickeln kann und bisschen über die Software gehört hat, die er entwickeln soll, oder für einen kompletten Java-Neuling. Da nützt dir das auch nicht. Das wäre auch nochmal für den von dir genannten Use Case interessant. Du hast ja Einarbeitung gesagt. Welches Level? Also für welches Skill Level soll das passieren? Also ohne Vorwissen der Programmiersprache, das sollte man auf jeden Fall festhalten.

I:Ja das ist natürlich ein guter Punkt. Wenn das jetzt primär die Softwarevisualisierung für diese Abteilung wird um sich in RCE einzuarbeiten, ist natürlich schon eine gewisse Programmierkenntnis vorhanden. Generell soll die Softwarevisualisierung neuen Mitarbeitenden in dieser Abteilung bei der Einarbeitung helfen. Du hast ja gerade auch schon viel erzählt, wie da eine Visualisierung helfen könnte. Ich würde gerne nochmal einen Schritt zurückgehen und die Einarbeitung unabhängig von der Visualisierung betrachten. Wie oft ist das so, dass du dich in eine neue Software einarbeiten musst?

B: Öfters. Das habe ich bestimmt schon 10, 20 Mal gemacht. Und dann weißt du halt eben auch, was dir eben fehlt. Der Worstcase ist, du kriegst das zwar irgendwie vielleicht nochmal gebaut und dann machst du irgendwas. Dann läuft dir ein Fehler ein, musst einen Debugger anwerfen und guckst dir dann wirklich an wo springt er lang und wo ist es. Das ist so das Worstcase-Szenario für eine Einarbeitung, weil du ja sonst nichts hast. Normalerweise heutzutage würde ich erwarten, ich habe ausreichend Informationen um loszulegen. Also ich weiß welche Umgebung muss ich aufsetzen, ich muss erst mal mir den Code holen, dann muss ich schauen, dass ich Java 11 installiert habe und danach muss ich das Maven Package aufrufen. In zwei drei Schritten würde ich das Ding eben bauen können. Dann baut es das ganze Ding, das verifiziert, dass ich das grundlegend in meiner Umgebung dann auch nutzen kann. Es gibt Tests, die laufen durch und idealerweise sind sie auch grün und wenn sie nicht grün sind, helfen sie dir auch herauszufinden, was du vielleicht doch übersehen hast. Das wäre so mal der Start. Wenn du das schon mal nicht hast wird es schwierig.

I: Würdest du auch sagen es kommen auch noch mehr Sachen vorher hinzu bevor du an der Software konkret sitzt?

B: Ja, ich will eine ordentliche Readme haben. Da soll drin stehen, in kurzen, einfachen Sätzen, was macht die Software. Das ist noch unabhängig von einer Visualisierung. Und dann natürlich, insbesondere, wenn ich irgendwelche Dinge erledigen möchte, möchte ich sowas wie eine Softwarearchitekturdokumentation oder ein technisches Konzept oder wie auch immer man das dann genau nennt. Das ist halt nicht nur bei der Visualisierung vom Code sondern auch eine textuelle Beschreibung wichtiger Entwurfsentscheidungen zum Beispiel.

I: Du hattest ja vorher die Knowledge Map erwähnt. Würde dir das bei der Einarbeitung auch helfen?

B: Ja das wäre interessant, genau. Gerade wenn es doch ein bisschen größeres Team ist würde es vor allem aber interessant werden herauszufinden, wen kann ich denn zu was fragen.

I: Wenn du dich bis jetzt eingearbeitet hast in eine neue Software, hast du da schon mit Visualisierungen gearbeitet?

B: Meistens statische 2D-Visualisierungen und da musst du auch Glück haben, dass die aktuell sind. Wenn eine Visualisierung, dann sollte sie aktuell sein und insofern nutzbar, dass ich nicht zu viel tun muss um einen Überblick zu bekommen. Das geht bei diesem 3D Kram einfach noch schwierig weil man natürlich selten eine 3D-Brille irgendwo liegen hat. Eine 2D-Repräsentation würde in vielen Sachen also schon ausreichen.

I: Und hast du dir Informationen auch mal selbst visualisiert, also Zusammenhänge aufgezeichnet?

B: Natürlich, sowas machst du dann halt auf Papier.

I: Wenn du dir jetzt vorstellst, du würdest dich in eine neue Software einarbeiten - zum Beispiel hier in RCE, was wäre denn das erste, das du dir anschauen würdest? Wie würdest du genau vorgehen?

B: Wie gesagt, ich würde zunächst schauen, dass ich es zum Laufen bekomme. Ich brauche einen Grundstock an sinnvoller Dokumentation. Ich würde erst nochmal wissen wollen, was macht das Ding, wofür ist es überhaupt da. Dann würde ich erst mal gucken, dass ich das bei mir lokal zum Laufen bekomme und dann würde ich mir ein paar Beispielworkflows anschauen. Also auch Beispiele sind immer ganz, ganz wichtig. Dann hängt es einfach so davon ab, was für Aufgaben man so am Anfang macht. Wenn du die Aufgabe hast, was zu optimieren oder ein Feature zu entwickeln, dann ist die erste Aufgabe immer, dass du herausfinden musst, wo zum Himmel muss ich das denn ran tun und was hängt denn alles dran? Und dann wird es langsam interessant. Zum Beispiel dann könnte dir eine Visualisierung eben helfen: Wenn ich diese Klasse ändere, die hat noch so einen Rattenschwanz an anderen Sachen dran, die ich auch noch anpassen muss. Das ist schon mal eine Aussage. Also ich habe eine Aufgabe, zum Beispiel eine neue Funktion einzubauen. Und dann möchte ich möglichst schnell herausfinden, mit Hilfsmitteln, Leute fragen oder was auch immer, wo muss ich denn da hin gucken? Du hast insgesamt immer eine grundlegende Setup Phase und dann hängt es viel von den Aufgaben letztendlich ab. Das hast du ja die ganze Zeit übrigens. Ich würde auch behaupten, nicht jeder weiß auch im RCE wo alles ist, wenn die jetzt was Neues machen müssen. Auch den Leuten hilft das dann in dieser Phase.

I: Und du würdest sagen, dass du dir die Software an sich nicht vorher anschauen würdest, ohne mit einer konkreten Aufgabe im Hinterkopf?

B: Nur auf einem sehr groben Level, nicht im detaillierten Sinne. Das detaillierte Level kommt dann aufgabenbezogen aus meiner Sicht. Dann musst du die Stelle finden, dann machst du die Änderung, dann musst du die Änderung verifizieren, dafür brauchst du wieder so Tests und so einen anderen Kram. Aber ich denke mal schon für die Lokalisierung und um zu erahnen, was da alles so dran hängt, ist das, wofür man so eine Softwarevisualisierung gebrauchen kann.

I: Wenn der Use Case ist, dass du dich in diese Software einarbeiten sollst. Würdest du sagen, es gibt einen Zielzustand? Wann bist du in eine Software eingearbeitet?

B: Ja, wenn ich in der Lage bin erste Aufgaben selbstständig zu lösen. Sagen wir mal das erste Mal - es gibt ja sowas wie merge oder pull requests. Wenn [jemand] sagt, hey, dein merge request ist accepted, dann wäre so eine erste Phase abgeschlossen. Also du bringst was in die Software rein und hast eine quasi produktive Änderung und das passt dann auch. Und wenn du das wiederholt hinbekommst, dann hast du eine erste Phase der Einarbeitung auch geschafft.

I: Und was würdest du sagen, welche Information über die Software müsstest du dann wissen? Wenn ich mir die OSGi Software vorstelle - welche Packages oder ähnliches müsstest du kennen? Oder wie viel davon - ist es nur die grobe Struktur?

B: Das ist sehr unterschiedlich. Gehen wir mal weg von der Visualisierung. Du hast halt unterschiedliche Ebenen. Erstmal was macht die Software? Was sind die Kernfeatures? Dann schaust du dir die Beispiele an, damit du dich im Groben damit vertraut machst. User Interface immer starten und ein paar Workflows mal machen. Da brauchst du ja auch immer schon ein paar Informationen, würde ich sagen. Dann eher um solche Aufgaben sag ich mal zu lösen, brauchst du diesen groben Überblick. Wenn du dir sowas wie RCE zum Beispiel anschaust, haben die ja nicht alle Klassen da

liegen aber es sind ja trotzdem 1000 Packages, die auf einer Liste sind, da brauchst du erstmal eine logische Sortierung davon, das ist so der erste Schritt. Also, wo sind irgendwelche Schichten, wie hängen die zusammen. Damit du irgendwie halbwegs effizient in dem ganzen Kuddelmuddel noch navigieren kannst. Und dann eher so über diese Schnittstellen. Hängt von der Aufgabe ab - ist es etwas, das ich lokal nutzen kann, dann müsste ich mir eher so die Klassen im Package anschauen, wie hängen die voneinander ab, sozusagen. Das könnte hilfreich sein. Und wenn die Aufgabe dann über mehrere Packages hinweg zu lösen ist, dann auf einer höheren Granularität schauen, wie hängen die denn eigentlich zusammen. Und dann hofft man, dass nicht alles mit allem irgendwie vermischt ist, weil dann ist die Visualisierung auch wieder nicht schön.

I: Und nochmal inhaltlich zur Einarbeitung, ist die Softwarehistorie dafür auch relevant?

B: Das hängt von der Aufgabe ab. Also wenn du jetzt eine Fehlersuche machst ist das schon hilfreich. Du kannst entweder Bugfixen, neue Funktionen einbauen, die Struktur besser machen oder einen anderen Aspekt wie die Performance besser machen. Das sind so die vier Hauptpunkte warum du Code änderst. Und da runter gebrochen brauchst du unterschiedliche Unterstützung. Beim Bugfixen hilft dir auch die Historie, weil der erste Punkt könnte sein, seit wann ist das denn so? Dann kommst du automatisch in die Historie rein. Aber dann guckst du dir das Change Set der Änderungen an und hoffst dass da nicht irgendwelche Leute dran gearbeitet haben, die dachten es sei cool, wenn alle drei Trilliarden Änderungen in ein Commit gepackt werden. Dann hilft es dir. Aber das wäre ein Klassiker für die Fehlersuche. Ansonsten wenn ich ein neues Feature einbaue, ähnliche Features oder so – nein, das macht seltener Sinn. Hängt dann wieder davon ab. Es gibt ja auch immer so Spezialisten, die bauen schon in weiser Voraussicht vor fünf Jahren das Feature halb ein, das ist noch schlimmer, wenn es nur halb funktioniert. Aber da passt es nicht. Ich glaube die Historie ist auch spannend, wenn du einen Code optimierst und dann auch feststellst, dass er langsamer geworden ist. Da musst du auch den Punkt finden, wann das so geworden ist. Da musst du aber die Performance sichtbar oder messbar machen sozusagen und dir dann die Commits dagegen anzuschauen. Damit kommst du wieder auf die Änderungen und siehst, okay, da hat einer versucht in Java 3 Millionen Strings in einer Vorschleife mit dem Plus Operator zusammenzubauen. Das bedeutet, dass jedes Mal ein neues Objekt erzeugt wird, das ist nicht so toll. Für sowas halt eben. Wenn du siehst, ich hab einen Bug, dann musst du erst mal diesen Root Cause finden und bei dieser Root Cause Analysis hilft glaube ich ganz gut auch die Historie.

I: Aber das findest du jetzt auch über den Code? Kannst du dir das auch anzeigen lassen?

B: Ja, dann muss ich mir halt selbst irgendwie ein Skript bauen. Oder müsste dann wirlkcih die Änderungen im Code anschauen. Was macht denn die Instanz zu diesem Zeitpunkt von deiner Software in Bezug auf die Performance oder so. Genau, und die Historie. Wir haben ja auch noch den anderen Punkt, wir müssen refactorn, Code besser machen. Wenn wir Code besser machen wollen, wir haben da ja nur ein gewisses Budget. Sagen wir mal so, weil du wirst nie den Code 100% perfekt bekommen. Deswegen interessiert es dich da, wo nutzt es dir am meisten. Da kommt wieder die Analyse der Historie ein Stück weit mit rein. Insbesondere wenn du dann halt gucken möchtest, zum Beispiel wie hoch ist die Änderung von diesem Codestück. Das bekommst du aus der Änderungshistorie.

I: Was meinst du mit wie hoch ist die Änderung?

B: Wie häufig hat sich das in einem Zeitfenster X geändert. Weil umso höher die Codeänderung umso höher ist auch die Möglichkeit, dass sich da irgendwelche Fehler eingeschlichen haben. Aber sowas willst du halt wissen. Wenn wir sagen, wir haben dann zunächst mal nur eine Hotspotanalyse von groß, breit, ohne Tests und dann kommt noch die Information des Code Change. Das bekommst du aus der Historie raus, dann hat man eine besser priorisierte Liste, wo man ansetzen kann.

I: Vorher hast du ja auch gesagt, dass du selbst dir zwischendurch irgendwelche Zeichnungen machst. Wie ist das so mit Zwischenschritten bei der Einarbeitung? Der erste Schritt war bei dir, dass du alles installierst und zum Laufen bringst. Wie sieht das aus, ist das eher so ein Morgen und du machst das komplett durch oder zwischendurch mit Pausen und notierst dir Sachen?

B: Das hängt von der Aufmerksamkeitsspanne ab. Wenn da irgendwelche Sachen zwischendurch sind, dann würde ich mir schon nochmal etwas notieren. Wenn du feststellst, dass das hilfreich ist und irgendeine Anleitung um etwas in Gang zu bekommen noch nicht existiert, dann ist das der erste Commit.

I: Hast du dich bis jetzt eher selbst, also alleine in Software eingearbeitet?

B: Häufiger alleine, würde ich sagen. Und dann hast du ja hoffentlich einen Ansprechpartner, mit dem besprichst du dich dann. Der wird ja auch inhaltlich zu den Sachen passen, die du als erste machst. Mit dem würde man sich dann eher absprechen.

I: Aber du würdest eher eine Weile für dich alleine arbeiten und dann Fragen sammeln?

B: Ja natürlich. Du willst die andere Person ja nicht die ganze Zeit nerven. Du versuchst dir erst mal selbst ein mentales Modell von der Software zu bauen. Da hat ja jeder ein anderes, weil jeder in unterschiedliche Bereiche rein schaut. Das brauchst du ja, du musst dir ja eine interne Konzeptmap aufstellen, von Detailkonzepten, die das RCE Team beispielsweise benutzt. Erst dann kannst du ja Fragen darüber stellen. Das ist zum Beispiel auch spannend. Zum Beispiel bei Leuten die noch nie programmiert haben, da sagt man zuerst, wir müssen denen erst mal beibringen, wie man richtig googelt. Was dahinter steckt ist eigentlich, die richtigen Begriffe zu kennen und zu kennen, was das bedeutet. Damit man dann überhaupt bei einem Problem X das selbst lösen zu können. Sei es nur über Stack Overflow suchen oder sowas. Das hast du halt hier bei einer spezifischen Software nicht. Das sind dann die Fragen die du deinem Einarbeitungsbegleiter stellen würdest.

I: Gut, dann würde ich jetzt in einem letzten Schritt nochmal versuchen das Ganze zu kombinieren. Du hast selbst schon ganz viel über Visualisierung zur Einarbeitung erzählt. Ich versuche jetzt nochmal herauszufinden, wie geeignet du Visualisierungen dafür findest. Wenn du dich wieder in eine neue Software einarbeitest und es gäbe eine Visualisierung, würdest du diese nützlich finden?

B: Die Frage ist sehr allgemein. Aber ich denke schon, unter der Voraussetzung, dass sie nicht manuell aufrechterhalten werden muss. Es gibt halt schon auch relativ abstrakte Visualisierungen. Ich sag mal die obersten Level von so einer Architekturschicht, die bekommst du auch manuell hin. Auch manuell aktuell gehalten, weil sich da nicht so viel ändert. Also wenn du da nicht jede Klasse mit rein bringst, dann geht das. Aber auf dieser Art von Visualisierung, wenn man jetzt die statische Struktur betrachtet, dann fände ich die Highlevelsicht, vielleicht noch in ein Package rein und eine Ebene tiefer sinnvoll. Hängt davon ab wo dann der Einsprungspunkt sich in die einzelnen Klassen und Module ergibt. Genau diese Schnittstelle zwischen (…) ich habe davon was, weil es mir Dinge anzeigt, die ich im Quelltext nicht so einfach erkennen kann, also Strukturen, die ich gerne Visualisiert haben würde. Dann mit dem Ziel, an die richtige Stelle zu kommen.

I: Und dann aber in den Code gehen und die Visualisierung muss deshalb noch gar nicht so detailliert sein?

B: Ja genau. Spannend ist dann aber trotzdem ggfs. wenn ich dann mal so Module oder eine Klasse oder direkte Abhängigkeiten anzeigen lasse, also zum Beispiel so eine Dependency Grafik. Also das Package hängt von den Sachen ab, damit du einschätzen kannst: Ist das jetzt so ein riesen Monster, dass du anfasst und dann fliegt dir alles andere um?

I: Und das wäre für eine spätere Frage dann also nochmal sinnvoll, wenn du gezielt ein Package hättest, mit dem du dich beschäftigst?

B: Ja genau, wenn ich dann eine Aufgabe hätte, wenn ich ein neues Feature entwickle, um das dann abschätzen zu können. Komplementär dazu wären natürlich auch in manchen Bereichen, wenn sie automatisiert bereitgestellt werden, so gewisse Ablaufdiagramme gut. Aber auch auf abstrakterem Level. Zum Beispiel wie sieht so ein Ablauf bei RCE aus, um einen Workflow durchzuführen. Da ist immer nur die Frage - das sind immer nur so punktuell wichtige Sachen. Du hast ja die statische aber auch die dynamische Schicht. Das ist halt so ein Punkt. Spannend ist auch, weiß aber nicht, ob man das aus einer Softwarevisualisierung auch raus bekommt, die Deployment Sicht. Ja, wie läuft das beispielsweise ab. So eine RCE Instanz kann ja auf zehn Knoten laufen. Wie sieht überhaupt das Installationssetup aus. Da sind immer so die drei wichtigen Sichten. Bei der Struktursicht würde ich von der System View - das ist das System da steht RCE drauf (…) da klickst du dann einmal drauf und das ist dann die erste Phase, die ich meine mit den Packages und Persistence und sowas. Das wären so fast die wichtigsten Sachen. Wenn man das automatisiert zusammenbekommt, ist das nett. Ab diesem Bereich kommst du dann in die Details und so rein. Das ist jetzt Fokus Einarbeitung mit dem Ziel eine Aufgabe ein neues Feature oder vielleicht auf Bugfixen. Das ist dann gut um dich zu erinnern, wo ist das denn? Hauptsächlich initial, auch diesen ersten beiden: System View und drum

herum mit den Packages und so, das ist denke ich mal initial am Anfang wichtig, damit du überhaupt eine Vorstellung hast. Danach wird es dann sehr spezifisch, an welchen Teilen du überhaupt arbeitest.

I: Ich habe noch eine kurze Rückfrage. Wenn diese Ebenen sich mehr oder weniger trennen. Du hast ja jetzt auch wieder von der Struktur geredet, die man sich anschauen sollte zur Einarbeitungsphase. Würdest du die Visualisierung und den Code auch verknüpfen wollen? Wäre es gut den Code auch in der Visualisierung anzeigen lassen zu können?

B: Ich würde meinen Link zu meiner IDE haben, wenn cih den Code irgendwie brauche. Plus, was man vielleicht noch sagen sollte: Ein Bild ist nett, aber ich möchte auch Erklärungen haben, also Kontext. Selbst auf der obersten Ebene. Was heißt denn, Datenserver, sozusagen. Selbst wenn du das halbwegs standardisiert beschreibst. Da sind dann immer irgendwelche Symbole drin, die muss man erklären. Ich möchte wissen, was ist die Notation. Wozu ist zum Beispiel der Datenbankserver da? So ein bisschen Zusatzinfo letztendlich. Wo auch immer man die herbekommt. Aber das brauchst du mehr als irgendwelche reinen Strukturen, dieses Kontextwissen, sozusagen.

I: Nur nochmal, ob ich das richtig verstanden habe. Du würdest die Visualisierung zu Beginn auf diesem gröberen Strukturlevel nutzen. Und dann später nochmal spezifischer, aber dann mit einem detaillierteren Level?

B: Ja ganz genau.

I: Okay, und würde dir dann helfen, wenn es die gleiche Art der Visualisierung ist? Oder könnte das auch eine andere Visualisierung sein?

B: Ich denke zwei Punkte kannst du von der Struktur her kombinieren. Diese initialen Highlevelsichten und dann mit einem Klick weiter reingehen und eher so dependencyartige Sicht, wo sind die Abhängigkeiten? Also das kann man schon kombinieren. Auch bei so einer Klasse, die ich eben meinte, wenn ich eine Aufgabe lösen will wie Code besser zu machen. So eine Hotspotanalyse wäre dann natürlich nett. Aber auch diese Strukturview, wenn du das wirklich bis zur Klasse runter machst. Da ist dann Komplexität visualisiert im zeitlichen Zusammenhang, das ist dann ganz spannend. Aber das ist nicht das wesentliche am Anfang. Das Big Picture muss dir ja irgendwie klar werden. In Form von einem Bild plus Erklärungen. Das wäre glaube ich gerade für die Einarbeitung am wichtigsten. Der Rest ist dann sehr aufgabenspezifisch und gerade für die Entwicklung später auch interessant.

I: Gerade hast du schon sehr selbstverständlich von Interaktionsmöglichkeiten gesprochen, wie dem Klicken und damit Wechseln in der Ebene. Also hältst du Interaktionsmöglichkeiten auf jeden Fall bei der Visualisierung für relevant?

B: Ja. Ja klar. Übergehen, etwas rausfinden, was ist der Kontext aber dann auch reingehen, in die Struktur reinzoomen, das wirst du schon brauchen. Weil sonst hast du wieder nur so eine Tapete, auf die du etwas kleben kannst, aber das nützt dir halt dann nichts. Eine überschaubare Menge an Information pro Schicht sozusagen, ist wichtig, ansonsten wird es zu viel und ist zu überladen und bringt dir wieder nichts.

I: Und ist es dann auch wichtig, dass du deine eigene Vorgehensweise wählen kannst? Man könnte ja auch beispielsweise eine Art Film ablaufen lassen, der automatisch die verschiedenen Ebenen nacheinander anzeigt?

B: Ich würde schon gerne selbst hin und her springen. Das hängt immer davon ab. Du lernst ja die ganze Zeit etwas, dann willst du vielleicht nochmal zurück. Ein Intro-video ganz am Anfang für die Basics würde eventuell noch gehen, aber irgendwann willst du es ja dann auch bisschen interaktiver gestalten, wenn du schon so eine Visualisierung hast.

I: Jetzt haben wir ja schon über das Zoomen gesprochen, kannst du dir auch noch andere Interaktionsmöglichkeiten vorstellen, die von Nutzen wären?

B: Ja, einen Knoten markieren und die Abhängigkeiten zu anderen anzeigen lassen, beispielsweise. Das ist so das was mir spontan einfällt. Und dann von dort aus relativ einfach in die richtige Stelle in meine Entwicklungsumgebung kommen.

I: Es gibt ja unterschiedliche Möglichkeiten. Du hast vorher schon von 3D und 2D gesprochen. Es gibt darüber hinaus auch noch Unterschiede in der Darstellung, beispielsweise ob diese abstrakt ist oder metaphorisch. Es gibt Darstellungen über Städte oder irgendwelche Planeten. (Beispiele auf Papier gezeigt)

B: (über d) - kenne ich aber das ist dann schon wieder etwas anstrengend, das ist für die Abhängigkeiten. Also ich fand so eine Stadt immer ganz nett. Ob das jetzt Inseln sind oder Städte sind oder ob die nun 3D oder 2D ist.

I: Wenn du sagst, du findest das ganz nett. Heißt das auch, dass du findest, die Metapher unterstützt das mehr, als wenn es abstrakt dargestellt werden würde?

B: (über b) auch die drei Sachen sind sehr zweckmäßig, für jemanden der Software entwickelt ist das glaube ich machbar. Weil die ja eh mit solchen Konzepten wie Graphen gefüttert sind. Für jemanden, der vielleicht ein bisschen von einer anderen Seite ran kommt, da würde ich mir die Metapher dann hilfreicher vorstellen.

I: Die Zielgruppe von der neuen Visualisierung sind ja Softwareentwickler, und bei denen würdest du dann sagen...

B: ... würde das abstraktere auch ausreichen, weil die eben schon bekannt sind von der Art. Also für mich würde das durchaus reichen, aber da ist ja jeder anders. Finde das aber auch durchaus auch relativ schön, diese Städte etc. Es ist schon ansprechender aber auch kein absolutes Muss.

I: Wenn dir das jetzt von der technischen Umsetzung frei stehen würde, ob du das ganze in VR, AR oder am PC haben könntest, wie würdest du dich entscheiden?

B: Im Moment würde ich mir das in 2D anschauen, bis das wirklich praktikabel ist in 3D augmented, wie auch immer. Das ist eher so, dass die Hardware auch einfacher zugänglich sein müsste. Dann würde ich das auch nochmal ausprobieren, aber so lange das nicht so ist, reicht mir das in einer 2D Darstellung, weil die zweckmäßiger ist mit den aktuellen Interaktionsmöglichkeiten, die du typischerweise hast.

I: Also wenn das alles perfekt laufen würde in 3D - weil du eben meintest, du würdest das dann auch nochmal ausprobieren - wäre die Visualisierung so hilfreich, dass du dir auch eine VR Brille aufsetzen würdest?

B: Im Moment denke ich nicht, weil mir die Visualisierung ja helfen soll, die Zeitspanne klein zu halten an die richtige Stelle im Quellcode zu kommen. Dadurch habe ich höchstwahrscheinlich durch diese 3D Variante, wenn ich das nicht in meine Alltagsbürobrille integriert habe, schon Nachteile. Es sei denn, dieser Augmented Teil ist wirklich so hilfreich, dass dieses Kontextwissen, das da rüberkommt hilft. Es geht nicht um die Visualisierung per se sondern um die Kontextinformationen. Also das Gesamtbild aus Visualisierung und Kontextinformationen, wenn das wesentlich einfacher zu konsumieren ist, als wenn ich das in 2D mache, dann würde ich das machen. Aber dann würde es mir ja wieder beim Ziel helfen möglichst schnell zur richtigen Stelle zu gelangen. Ich schau mir das ja nicht aus Spaß an, sondern ich möchte ja meine Aufgabe lösen. Also für so einen Überblick, okay, dann so ein Flug über eine Insel, das ist alles schick. Aber wenn du wirklcih Dinge erledigen sollst, darauf sollte eine Visualisierung eher getrimmt sein.

I: Das waren auch inhaltlich die drei Blöcke, die ich hatte. Meine letzte Frage wäre einfach nochmal, ob du kurz deine wichtigsten Anforderungen auf den Punkt bringen kannst.

B: Naja ich möchte halt nicht alle 1000 Klassen auf einmal sehen, sondern verschiedene Abstraktionsebenen haben. Von dort will ich dann von oben nach unten in eine Software reinschauen können. Ansonsten hilft mir das nicht so viel mehr, als das, was eine gute IDE oder ein Texteditor mit guten Entwicklungsplugins dir sonst leisten kann. Das ist so ein Punkt. Die Dependency View ist ganz interessant. Und nicht nur Struktur. Auch dynamische Aspekte - da gibt es ja unterschiedliche Varianten. Bisher ist es ja viel auf Struktur gemünzt und wenn du etwas weiterentwickeln willst, ist das nicht das einzige. Wenn man das zusammen bringen würde, es gibt ja auch gute Modelle um Softwarearchitektur aus verschiedenen Sichten darzustellen. Momentan sind wir da aus meiner Sicht zu stark auf die Struktur eingefahren. Das ist so die Message. Zu einem Gesamtbild gehört noch mehr dazu. Und ein letzter Punkt noch. In einer Visualisierung sollte dann eben gut dieses

Kontextwissen, was irgendwo steht, sei es in Kommentaren oder sowas, ersichtlich sein und hier bekannt werden. Einfache Sachen, wie Erklärungen zu irgendwelchen Boxen. Wenn du dir ein System Level View anguckst, mit dem System in der Mitte und den direkten Interaktionspartnern, was hat Interaktionspartner A, B, oder C zu tun. Da gibt es dann immer auch fachlich technisch/ genau das was wir hier vergessen haben, sind zum Beispiel die Nutzergruppen. Damit könnte man ganz gut einsteigen, in diese Systemkontextview und dann sich weiter in die Tiefen begeben. Das wäre ja gerade spannend für den Anfang, weil man ja am Anfang nicht weiß für wen das da ist. Dieses Nichttechnische drum herum ist da ja erst mal viel spannender. Damit du das Verständnis erstmal hast, damit du wirklich reingeführt wirst, sozusagen. So vom Ablauf. Also wenn du wirklich diesen Fokus Einarbeitung hast, dann sind glaube ich die Konzepte drum herum als Einstieg erst mal wichtiger als eine detaillierte Darstellung. Wie gesagt, erst, wenn du eine Aufgabe als Teil deiner Einarbeitung machst. Für mich wäre bei Einarbeitung auch immer wichtig, dass irgendwelche Aufgaben erledigt werden. Du brauchst dann immer jemanden, der den Neuen begleitet und dann erkennt, okay, der ist jetzt drin in der Software. Aber die erste Meile, das sollte erst mal sein, sich dieses Kontextwissen anzueignen, damit man seine Aufgaben erledigen kann. Wenn das die Visualisierung aufgreift und diesen Prozess in Teilen unterstützt, dann wäre das eine schöne Sache. Dann kann man das auch dahingegen weiter entwickeln, weil dann wird es auch - das ist mir immer wichtig - praktisch relevant. Bisherige Visualisierungen sind zwar ein nettes Spielzeug, bei dem man sich Demos zeigen lassen kann aber die praktische Relevanz ist noch nicht ausgereift.

I: Das ist ja auch genau das Ziel einer neuen Visualisierung, dass diese praktische Relevanz sichergestellt werden kann. Das wären dann auch alle meine Fragen gewesen. Hast du denn noch irgendwelche Fragen oder noch zu einem Thema Anmerkungen?

B: Nein, das wär eigentlich alles.

I: Dann, vielen Dank, dass du dir die Zeit genommen hast, meine Fragen zu beantworten.

## Interview 3

[…]

I: Und hast du schon mal mit Softwarevisualisierung gearbeitet?

B: Die IslandViz kenne ich. Ich habe letztendlich von Anfang an irgendwo immer mitbekommen was hier passiert. Habe mich auch dafür interessiert. Und habe dann auch immer mal, wenn sich die Möglichkeit ergeben hat, mir angeschaut, wie gerade der Stand ist. Kenne also sowohl diese VR-Geschichte als auch die AR-Geschichte.

I: Und abgesehen von der IslandViz, hast du da schon mal mit Softwarevisualisierung gearbeitet?

B: Nein. Also vielleicht noch ergänzend, es gab ja vorher schon so 2D-Visualisierungsansätze, am Computer, wo es ja auch verschiedene Ideen gab. Diese Comicvisualisierung und Städtemetaphern und so. Das habe ich auch alles so am Rande mitbekommen. Immer mal wieder was gesehen, aber mich jetzt nicht so intensiv damit auseinandergesetzt, dass ich gesagt hätte okay, ich habe vollständig das Mapping von Softwarearchitektur auf diese Visualisierung jetzt verstanden.

I: Und bei der Entwicklung von Software hast du auch noch nie mit Visualisierungen gearbeitet?

B: Da muss man bisschen abgrenzen, was ist Softwarevisualisierung letztendlich? Ich verstehe das in diesem Kontext so, dass man versucht Software oder Code zu visualisieren. Also was die Software an sich ist. Wir visualisieren in RCE zum Teil auch, aber mehr dann Daten. Das sind auch Daten die aus irgendeiner Software raus kommen und dann gibt es da irgendwelche Graphen die vielleicht angezeigt werden. Aber das ist ja glaube ich nicht das Verständnis von Softwarevisualisierung.

I: Das wäre ja dann eher der Outcome von der Software?

B: Ja richtig, genau. Und wenn man das so betrachtet, dann habe ich mit Softwarevisualisierung noch nichts gemacht.

I: Aber gab es schon mal eine Situation in der du dachtest es wäre praktisch eine Visualisierung zu haben? Oder, dass du dir selbst eine kleine Visualisierung erstellt oder aufgezeichnet hast?

B: In Form von Diagrammen, ja klar. Also wo ich gerade sagte, dass ich viel in der Datenbankecke unterwegs bin, da macht man natürlich unglaublich viel mit Diagrammen. Zum Beispiel UML-Diagramme oder simple Abhängigkeitsgraphen. Das kommt dann ja aber eher aus der anderen Ecke. Man plant vielleicht anhand eines Diagramms und setzt das dann in Software um. Ansonsten, was es bei uns natürlich gibt, sind so Netzwerkdiagramme oder so. Das kommt aber eher aus der konzeptionellen Phase, da werden solche Diagramme erstellt und später implementiert und nicht anders herum.

I: Könntest du dir aber vorstellen, dass dir das zum Verständnis helfen könnte?

B: Ja, definitiv. In bestimmten Bereichen zumindest, gerade wieder auch im Datenbankbereich. Um ein Datenmodell zu verstehen ist so ein Diagramm, wenn man das als Visualisierung versteht, natürlich hilfreich. Ich glaube aber auch da dann halt als Diagramm vollkommen ausreichend.

I: Und für die restliche Arbeit als Entwickler empfindest du die reine Arbeit mit dem Quellcode als geeigneter.

B: Stand jetzt würde ich sagen, ja. Alles was ich bis jetzt kennengelernt habe, da sehe ich noch nicht, dass mich das weiter bringt. Aber aus folgendem Grund: Wir nutzen ja sogenannte IDEs, also Entwicklungsumgebungen um zu programmieren. Und diese Entwicklungsumgebungen haben mittlerweile eine wahnsinnig gute Unterstützung um in einem Quellcode navigieren zu können. Das heißt irgendwelche Abhängigkeiten innerhalb von Methoden kann ich sozusagen, per Tastendruck nachverfolgen, kann sehr schnell sehen, wie hängen Dinge zusammen. Also da bietet diese Entwicklungsumgebung sehr, sehr viel. Und gleichzeitig hatte ich immer das Problem, bei den Visualisierungen die ich kenne, dass ich nicht den Schritt zum Quellcode machen konnte. Weil der Quellcode ist ja in den meisten Fällen, zumindest in den Sprachen in denen wir uns hier bewegen, das sind ja so Hochsprachen, und da ist Quellcode gut lesbar. Und wenn ich den Quellcode lese, verstehe

ich. Das ist immer das, was mir dann gefehlt hat. Ich konnte bis jetzt vielleicht Abhängigkeiten zwischen Klassen in so einer Visualisierung sehen. Ein Klassenname kann dann vielleicht auch noch sprechend sein, sodass ich vielleicht eine grobe Vorstellung davon bekommen könnte, was damit gemeint ist, aber ich sehe nicht, was darin passiert. Und dann bringt mir auch das Wissen über eine Abhängigkeit nichts mehr.

I: Alles klar. Dann würde es im nächsten Teil generell um die Einarbeitung in eine neue Software gehen. Und die erste Frage die ich mir dabei gestellt habe ist, wie oft das überhaupt vorkommt, dass man sich in eine neue Software einarbeitet? Wie häufig hast du dich in eine neue Software eingearbeitet?

B: Also super oft kommt das nicht vor. Bei einem Softwareentwickler kommt das genau dann vor, wenn man irgendwohin wechselt. Also in eine neue Einrichtung oder in einer Einrichtung oder einer Abteilung in ein Team dazu stößt. Ansonsten zeichnet sich die Softwareentwicklung ja gerade dadurch aus, dass man gerade bei Null anfängt und etwas entwickelt. Da ist ja keine Einarbeitung erforderlich, man fängt ja bei Null an. Ich hab das bei RCE gemacht […] Ich habe das auch schon mal im kleineren Rahmen, weil RCE ist eine recht große Software mit viel Quellcode und vielen Abhängigkeiten, vielen Besonderheiten auch. In kleineren Projekten auch schon mal vorher gemacht. Das sind aber dann so Größenordnungen, wo man sich innerhalb von wenigen Tagen zurechtfindet, also schnell eingearbeitet hat, bei den kleineren Projekten. Bei RCE ist das mit Sicherheit nicht so gewesen. Also das ist eine größere Aufgabe und das sehe ich jetzt aktuell auch […] dass das keine Sache ist von Tagen oder Wochen, dass man ein Verständnis entwickelt hat. Das dauert durchaus länger. Das hat aber nicht nur, meiner Meinung nach, mit der Software an sich zu tun, mit der Architektur und dem Quellcode und allem was dazu gehört. Sondern man muss ja auch erst mal den Anwendungsfall der Software verstehen. Man braucht gegebenenfalls auch Domänenwissen über die Domäne, wo die Software eingesetzt wird. Das kommt ja dazu. Je komplexer schon der Anwendungsfall ist, den man verstehen muss, desto schwieriger ist letztendlich auch die Einarbeitung in die Software, weil die Software ist ja "nur" eine Umsetzung der Anforderungen an die Software.

I: Du hast gerade gesagt, dass es bei RCE länger dauert als Wochen. Hast du da einen Endzeitpunkt vor Augen, wann man eingearbeitet ist und alles gesehen hat?

B: Also ich hab noch nicht alles gesehen. Das lässt sich glaube ich kaum quantifizieren. Das ist halt so, ich glaube es ist ein bisschen ein Spezialfall, weil wir hier in der Wissenschaft unterwegs sind und diese Software Wissenschaft unterstützt und wir als Projektpartner in wissenschaftlichen Projekten unterwegs sind. Da kommen oft Anforderungen aus Projekten, die wir auch in der Software umsetzen, wo die Funktionalität selbst auch noch erprobt wird, ob das überhaupt Sinn macht sozusagen. Gleichzeitig ist aber RCE eine Open Source Software, das heißt die kann sich jeder, auch außerhalb des DLR, herunterladen und nutzen. Und wenn man dann einmal eine Funktion implementiert hat, ist es schwierig diese wieder herauszunehmen, weil wir wissen ja nicht, wer nutzt sie und wer nutzt sie nicht. Und so sind auch vielleicht einfach seit vor meiner Zeit, also RCE wird ja schon seit 2005 entwickelt, Teile hereingekommen, die sind in Vergessenheit geraten. Die nutzt wissentlich kaum einer den wir kennen oder damit kommt man sehr selten mit in Berührung. Es passiert mir schon, dass man immer mal wieder neue Ecken findet. Nicht unbedingt nur Funktionen sondern auch Ecken eben im Code über die man noch nicht gestolpert ist.

I: Vorhin hast du auch gesagt, dass zunächst wichtig ist, den Anwendungsfall zu verstehen. Gibt es da irgendwelche Dokumente oder Hilfsmittel oder hast du mit Leuten gesprochen oder wie kann man sich das vorstellen?

B: Das ist ein Mix aus allem. Klar mit Leuten sprechen, also mit den Entwicklerkollegen letztendlich. Was mir auch unbedingt geholfen hat, ist, dass ich dann echten Anwendern über die Schulter schauen konnte. Wir entwickeln ja eine Software, bei der wir selbst nicht Anwender sind. Wir testen zwar und wissen auch, wie etwas funktionieren soll, aber den echten Anwendungsfall haben wir ja nicht. Den haben wir in den Projekten halt. Wenn wir dann sehen, was dort passiert. Wie da konkret die Software eingesetzt wird, unabhängig von den kleinen Beispielaufgaben, die wir hier als Tests benutzen, dann versteht man auch nochmal eine Ecke mehr. Und das ist halt erstmal um das Verständnis der Software von der Anwenderseite her zu bekommen. Das Wissen muss man haben, also was soll die Software überhaupt machen, bevor man wirklich tief in den Code einsteigt. Was dann die weitere Einarbeitung in den Quellcode angeht, da machen wir das auch heute noch so, und das habe ich damals letztendlich auch so gemacht, wir haben ja so ein System, in dem wir unsere Entwicklungsarbeiten drüber Koordinieren. So einen Bugtracker. Da gibt es dann sozusagen Bugs

oder Issues, wie wir es nennen, die zu bearbeiten sind. Am Anfang hat dann […] Teamleitung mir Issues zugewiesen, das aber unter einem speziellen Fokus. Sprich Aufgaben, die klein genug waren, dass ich sie schon lösen konnte, weil sie vielleicht ein Stück weit unabhängig von dem Anwendungsfall der Software sind sondern vielleicht eher programmierspezifisch sind. Also man muss sozusagen nicht den großen Zusammenhang erkennen um kleinere Bugs zu fixen. Irgendeine Textausgabe auf einer Kommandozeile oder so, das kann jeder, da brauch ich nicht wissen, warum kommt die. Das heißt es waren von Anfang an lösbare Aufgaben, aber die Aufgaben dann weit verteilt über die ganze Codebasis, weil man liest sich ja in den Code ein. Und wenn man dann in viele Stellen reinguckt, fängt man an das System zu verstehen. […] Ich glaube das ist auch was, da lernt man unglaublich viel drüber. Das ist ja auch so, am Anfang fragt man auch viel, hat viele Rückfragen. Und das ist auch was, das immens wichtig ist, dass man da Kollegen hat, die die Erfahrung und das Wissen haben und mit denen man dann rücksprechen kann. Viele Sachen lassen sich über den Code ableiten, viele Sachen lassen sich vielleicht auch über eine Architektur oder eine Form von Visualisierung ableiten. Aber das Wissen warum denn vielleicht eine Funktion in die Software reingekommen ist oder was das Ziel dieser Funktion ist oder warum vielleicht auch eine Funktion in der Form, in der sie implementiert ist, implementiert ist. Man kann ja Sachen auf unterschiedlichen Wegen implementieren, aber es gibt ja immer einen Grund, warum man das so gemacht hat. Und dieses Wissen, bestenfalls ist es dokumentiert, in der Realität ist das meistens nicht dokumentiert, und dieses Wissen dann von den Kollegen, die länger da sind abzugreifen, ist immens wichtig, weil dann fängt man auch an zu verstehen.

I: Und wie geht man da vor? Woher weißt du wer das bearbeitet hat und das wissen könnte?

B: Wie gesagt, wir haben ja im Team unsere Spezialgebiete und so groß ist das Team nicht, das bekommt man ja schnell raus. Dann weiß man, wen man fragen kann. Oder man geht halt zu jemandem der einem dann sagt, das kann ich nicht beantworten frage mal den oder den.

I: Aber das kann man nicht über den Code erkennen?

B: Ja doch, in manchen Fällen schon, nicht unbedingt in allen. Am Code steht schon immer der Autor dran. Aber so Code wächst, das heißt es stehen auch an verschiedenen Dateien, verschiedenste Autoren dran. Und zusätzlich, auch wieder wissenschaftstypisch, wir haben natürlich auch eine vergleichsweise hohe Fluktuation. Das heißt, es stehen auch mitunter Namen dran, die bekommt man nicht mehr im Nachbarbüro gegriffen.

I: Und du hast ja jetzt auch gesagt, man versucht dann an verschiedenen Stellen in den Code zu gehen. Wie entwickelt sich dann das Bild für das große Ganze? Einfach, weil ich viele Sachen dann sehe?

B: Ich glaube, das ist einfach ein Prozess. Wie gesagt, man fängt sehr kleinteilig an, oder mit sehr einfachen oder kleinen Aufgaben, und dann ist das natürlich immer auch wieder eine Rücksprache mit dem Team oder Teamleitung. Ich glaube da hilft dann auch zusätzlich dann noch, dass wir so fixe Termine und Runden haben, in denen wir uns austauschen. Ich glaube das ist auch etwas, das viel hilft, dass man über die Dinge redet um zu verstehen. Ich glaube dann wächst man da rein. Dann bekommt auch irgendwann eine Teamleitung mit, ob man schon etwas größere Aufgaben übernehmen kann oder dann auch mal konzeptionelle Arbeiten übernimmt. Und nicht nur die Teamleitung bekommt das mit, man selbst merkt das ja auch und dann redet man da drüber und dann merkt man, dass man mehr übernehmen kann. […] Das ist halt auch was, dass halt so eine Software nicht fix ist, die entwickelt sich weiter. Gerade bei uns, wir sind da ständig am Weiterentwickeln mit neuen Features aber auch am Dinge verbessern, neue Technologien versuchen wir mitreinzukriegen. Wir merken auch, dass wir an manchen Stellen mit der jetzigen Implementierung an Grenzen stoßen, einfach, weil sie schon alt ist. Also dreizehn Jahre, da ändert sich viel. Nicht nur an den Anforderungen an die Software sondern halt auch einfach an den Anwendungsfällen. Die werden halt auch größer und dann war das vielleicht mal falsch skaliert oder wie auch immer, und dann muss man da wieder ansetzen. Und da muss man einfach drüber sprechen. Weil ich glaube alleine wäre das eine Mammutaufgabe einfach.

I: Wenn du gerade gesagt hast, dass man irgendwann bereit ist für diese größeren Aufgaben. Würdest du da sagen, dass an diesem Punkt der erste Einarbeitungsschritt abgeschlossen ist?

B: Das ist schwierig zu sagen. Ich sag jetzt einfach mal, um das ein bisschen zu quantifizieren, die simpelsten Aufgaben, also so ein Issue oder ein Bugfix oder so, das kann sein, dass das ein

Arbeitsaufwand ist, von einer halben Stunde. Oder noch geringer, dann ist man damit durch. Also wenn man sich dann schon ein bisschen auskennt, ich sage mal das simpelste ist dann beispielsweise in der GUI oder wo auch immer in der Ausgabe einen Rechtschreibfehler zu korrigieren. Das ist ja was, das kann so ein Issue sein, ein Bugfix, und das ist sofort erledigt. Eine große konzeptionelle Aufgabe inklusive Implementierung kann eine Sache sein, von einem halben, dreiviertel Jahr. So und dazwischen gibt es die komplette Bandbreite von Aufgabengrößen. Das ist dann glaube ich sehr individuell auch abhängig vom Entwickler. Zum Einen, wie schnell man sozusagen das Verständnis für die gesamte Software entwickelt. Aber auch wie man persönlich gestrickt ist, was man gerne macht. Es gibt ja Leute, die gerne konzeptionell arbeiten, sich dann auch gerne in diese gesamten Zusammenhänge einarbeiten. Es gibt aber auch Leute die gerne einfach nur Code schreiben. Die dann sozusagen die Anforderungen vorgesetzt bekommen und dann gerne das einfach implementieren. Das spielt glaube ich auch eine Rolle, wenn man sagt, welche Aufgaben übernehme ich überhaupt. Das ist ganz vielfältig und deshalb kann ich jetzt nicht sagen, grundsätzlich ist eine Einarbeitungsphase dann abgeschlossen, wenn eine bestimmte Aufgabengröße überschritten wird.

I: Vorher hast du ja wegen der Visualisierung gesagt, dass man auf der Ebene der Klassen war, also dass man sich dort noch Namen anzeigen lassen konnte. Wie genau weiß man denn darüber Bescheid? Wie groß ist der Detailgrad, den du dir anschaust, wenn du dir einen Überblick verschaffen möchtest.

B: Ich muss zugeben, ich bin ein bisschen geprägt von dem, was ich hier schon gesehen habe. Das ist eventuell etwas problematisch. Das ist etwas, dass ich Status jetzt sagen würde, das würde mich Null weiterbringen. Weil, selbst wenn ich sozusagen die Software an sich schon mal gesehen habe und ich weiß, was die Software tut. Das ist das, was ich sehe wenn ich die Software benutze. Das hat ja erst mal überhaupt nichts mit der Implementierung zu tun. Also da brauche ich kein Wissen über die Architektur der Software oder die Programmiersprache oder irgendwelche Datenbanken oder was weiß ich, das interessiert mich ja erst mal nicht. Da muss auch nicht ein erkennbarer Zusammenhang sein. Ich sag mal, ich kann ein und dieselbe Software auf unterschiedliche Arten konzeptionieren und auch implementieren. Das heißt, wenn ich das Wissen habe, wie das funktioniert. Und ich setze mich dann hin und sage ich kann mir jetzt visuell irgendwas anzeigen lassen, wie Zusammenhänge sind, zum Beispiel auf Klassenebene, dann kann ich halt mit diesen Klassennamen nichts anfangen, weil ich diese Zuordnung zur Funktionalität vielleicht in Grenzen hinbekomme, weil Klassen entsprechend benannt sind. Wenn ich jetzt aber als Externer ins RCE Team kommen würde... Wir haben das jetzt zufällig mal gemacht, weil wir einige Dateiheader ändern mussten, wir haben ungefähr 2,5 Tausend Quellcodedateien in RCE. Und in jeder Quellcodedatei, oft ist das eine Klasse, aber es gibt auch Quellcodedateien, in der dann mehrere Klassen drin stehen. Und die können nicht alle so benannt sein, dass man das sofort erkennt, was das ist. Wenn ich mir dann da so was angucke, dann kann ich nur sagen, das sieht nett aus. Den Mehrwert erkenne ich da ja aber noch nicht. Weil ich nicht auf die Ebene des Quellcodes komme, wo ich durch Lesen verstehen kann, was da passiert. Ich weiß auch nicht, ob nicht sogar, also ich habe das noch nie probiert, mich über die Architektur oder die Zusammenhänge in die Software einzuarbeiten. Für mich war das eigentlich immer der Weg aus dem Quellcode, also von einem bestimmten Punkt heraus. Das kann eine kleine Methode sein, die vielleicht nur eine Ausgabe macht, wo ich dann aber gucke, okay wo kommen denn die Daten her, die ich damit ausgebe. Und dadurch springe ich dann vielleicht in eine andere Methode rein, vielleicht sogar in eine andere Klasse. Und dann erkenne ich, okay es gibt Abhängigkeiten zwischen den Klassen. Aber ich komme eigentlich immer eher vom Quellcode her und versuche von einem bestimmten Punkt aus Abhängigkeiten zu erkennen und mich vom Quellcode aus auf die Vogelperspektive herauszubewegen. Die Visualisierungen, die ich bis jetzt gesehen habe, hatten immer mehr erst diese Vogelperspektive, wo man dann versucht hat näher ranzukommen. Und an einem bestimmten Punkt war Ende. Die Information, die ich dann vielleicht nötig habe um das zu verstehen, an die komme ich dann nicht mehr dran.

I: Das Ziel ist es ja jetzt eine neue Visualisierung zu erstellen. Wenn du dir vorstellen würdest, dass diese dir dann von innen nach außen helfen würde. Also du kannst dir anzeigen lassen, wo du im Quellcode bist und du könntest dann rauszoomen. Würde das helfen?

B: Also es gibt Punkte, wo man diese Information braucht. Das ist auch wieder sehr abhängig von der Aufgabe, sage ich jetzt mal, und da ist halt/ da kann ich mir noch nicht so richtig vorstellen, aber es könnte vielleicht funktionieren. Ich habe es so noch nicht erlebt. Wenn ich jemand Neuem direkt eine etwas größere Aufgabe geben möchte, dann müsste der natürlich schneller die Zusammenhänge, die Architektur verstehen. Das ist halt nicht das, was wir so leben. Wir leben im Moment kleine Nadelstiche sozusagen um dann im Verlauf und durch Gespräche und durch implizites Lernen

letztendlich die Zusammenhänge zu verstehen. Das ist auch etwas, wo ich glaube, dass es auch auf die Größe der Software ankommt. Wir haben mit RCE schon etwas, das ist ein riesen Ding. Im Vergleich zu anderer Software. Es gibt natürlich noch Dinge, die sind viel, viel größer. Die sind dann aber auch mit anderen Teams aufgestellt und sind dann oft in andere Bereiche untergliedert und dann hat man Zuständigkeiten für kleinere Bereiche letztendlich. Dann ist es vielleicht doch wieder vergleichbar. Aber ist halt etwas, wo ich glaube, dass man nicht, weil man eine große Visualisierung hat, in unglaublich kurzer Zeit erfassen kann, wie ist das aufgebaut. Ich finde, es gehört mehr dazu als nur Verständnis über eine Architektur der Abhängigkeiten zu haben. Ich muss ganz viel lernen um dieses gesamte Bild zu bekommen. Ich muss den Quellcode kennen, ich muss die ganze Architektur kennen, ich muss vielleicht ein Datenmodell kennen, ich muss den Anwendungsfall kennen. Und dann kann ich mir das Gesamtbild aufbauen. Ich glaube wenn jetzt jemand sagt okay, ich kann von einer Stelle Quellcode mir visualisieren, wie sind jetzt Abhängigkeiten zu anderen Softwarekomponenten, dann ist das vielleicht etwas, das ich mir im Moment auf einem anderen Weg erarbeite. Klar, die Information brauche ich irgendwann, aber die alleinige Information reicht nicht aus, um das gesamte Verständnis zu entwickeln.

I: Ich habe noch eine Frage, bevor wir weiter über Visualisierungen sprechen können, zur Einarbeitung generell. Wie relevant ist denn die Historie der Software am Anfang?

B: Also ich glaube zu Beginn beschäftigt man sich nicht damit. Normalerweise würde ich sagen möchte man den Ist-Status erfassen, weil an diesem Status wird weiterentwickelt. Was ich aber eben schon mal sagte, es kommen Fragen auf, warum ist bei einer Implementierung die Entscheidung getroffen worden, das in dieser Form oder der Form zu machen. Oder warum ist die Entscheidung getroffen worden, diese oder jene Technologie auszuwählen. Das sind vielleicht Fragen, die irgendwann einmal relevant werden. Dann ist man aber meistens auch in einer Phase, in der man schon konzeptionell arbeitet, also wo man schon ein Stück weiter ist. Das ist glaube ich nicht vom Beginn der Einarbeitung aus wichtig. Und auch, wenn ich jetzt gerade überlege, welche Fragestellung ich damit beantworten könnte, dann ist das immer was, das sehr speziell auf eine Frage. Wo dann vielleicht in einem Punkt mal wichtig ist, warum ist das so oder so gelaufen. Aber da wäre die Information über die gesamte Historie viel zu viel, das würde mir auch nicht weiterhelfen.

I: Dann können wir jetzt zurück zu den Visualisierungen. Es gibt generell auch die Möglichkeit Visualisierungen noch ganz anders zu gestalten. Dass es nicht nur die Architektur ist, man kann auch zum Beispiel den Quellcode über eine Hotspotanalyse visualisieren. Das ist ein ganz anderer Ansatz. Könntest du dir sowas eher vorstellen?

B: Ja... also... kann ich mir vorstellen. Es gibt ja viele Sachen die man sich vorstellen kann. Es gibt auch viele Ideen, die man visualisieren kann. Die Nutzung von Features zum Beispiel, vielleicht die Nutzung von Code, vielleicht die Testabdeckung von Software, also was ist getestet und was nicht, was weiß ich. Aber wie soll ich sagen, als Entwickler bin ich gewohnt mit Text und mit Quellcode zu arbeiten. Und für all diese Fragestellung gibt es Tools und Softwareunterstützung, die mir Zahlen ausspucken, sage ich jetzt mal, oder Pointer auf Code stellen, direkt. Also was du gerade sagtest, ich könnte ja mal visualisieren über unterschiedlich große Darstellungen, was wird viel genutzt in der Software und was wird wenig genutzt, das kann man alles sich ausspucken lassen. Also, wenn es darum geht, was wirklich viel auch an Funktion genutzt wird, dann müsste man etwas einbauen um das auch mitzutracken und so. Aber also ich persönlich, stelle mir dann die Frage, was bietet mir diese Visualisierung mehr, als eine Zahl. Oder zum Beispiel die Testabdeckung, da gibt es Tools für, das machen wir sogar auch, wo geguckt wird, welche Codestellen werden durch sogenannte Unit Tests, also Tests, die wir auch im Code schreiben und die permanent automatisch auch ausgeführt werden, welche Codestellen werden davon abgedeckt und welche nicht? Und dann kann man sich das ausgeben lassen und dann weiß man Bescheid eigentlich. Oder, was man ja auch visualisieren könnte, beispielsweise über Distanzen zu irgendwelchen Kernen, welche Codestellen also auf welchen Pfaden wird was erreicht. Was sind so die längsten Pfade und wohin. Klar, kann ich mir alles visualisieren, kann ich mir aber auch einfach ausgeben lassen. Ich lese immer Quellcode, ich lese immer Text und wie gesagt, diese Entwicklungsumgebung bieten immer noch Unterstützung dass sie einem sagen, da ist jetzt das wo du dich gerade darauf fokussierst, geben dir das textuell aus. Meistens kannst du einfach drauf klicken und dann bist du an der Stelle. Ich finde dass dort die Unterstützung für einen Entwickler schon echt gut ist. Auf der anderen Seite, so etwas an einem Bildschirm zu visualisieren, da könnte ich mich noch eher drauf einzulassen. Ich könnte mir aber zum Beispiel nicht vorstellen, mich hier an den Schreibtisch zu setzen und eine Brille aufzusetzen um irgendwo einen Hotspot zu finden, den ich mir jetzt mal genauer angucken müsste, was ich Stand heute, aber dann wieder am Bildschirm machen würde. Dann müsste ich die Brille wieder absetzen,

dann müsste ich wieder hin und her switchen, da ist mir der Aufwand auch wieder viel zu groß. Und es ist vor allem auch nicht bequem.

I: Wenn du jetzt dir vorstellen könntest, du hättest am Bildschirm eine Visualisierung. Du hast von ganz vielen Fällen gesprochen, an denen du keine Visualisierung brauchst. Gibt es irgendetwas wo du denkst, da könntest du generell noch Unterstützung gebrauchen?

B: Wir haben ja in der Softwareentwicklung immer mit Debugging zu tun. Alles was wir entwickeln ist ja nicht von Anfang an fehlerfrei oder lauffähig, deshalb testen wir ja auch. Wir schreiben bestenfalls sogar den Test für eine bestimmte Methode, bevor wie die Methode schreiben und entwickeln dann dahin, dass dieser Test funktioniert. Das funktioniert aber nicht immer, gerade im GUI-Bereich funktioniert sowas nicht unbedingt gut. Das heißt man muss sozusagen oft zur Ausführungszeit sich den Quellcode angucken um zu sehen, was passiert jetzt genau da. Also man hat eine Logik hingeschrieben, aber wenn jetzt diese Logik mit Daten abgearbeitet wird, dann kann das ja auch sein, dass man im Kopf einen Denkfehler hatte und anhand eines Datums oder so eine Entscheidung getroffen hat, die eigentlich falsch ist. Das heißt man muss zu Ausführungszeit reingucken, was passiert wo. Auch da bieten diese Entwicklungsumgebungen schon eine super Unterstützung. Das heißt ich kann sozusagen meine Software ausführen, kann sogenannte Breakpoints setzen in der Software, wenn die Codestelle erreicht werde, wo ich diesen Punkt gesetzt habe, dann stoppt das Programm und ich kann sozusagen im Quellcode sehen wie ist der Status meiner Variablen, die ich da habe. Ich kann sozusagen codezeilenmäßig weiterspringen und sehen, wo läuft das Programm jetzt hin, entscheidet es sich nach links oder rechts und so weiter. Das ist was, das macht man schon häufig beim Entwickeln, wirklich dieses Debugging. Und das ist manchmal was, wo ich mir denke, das könnte man vielleicht übersichtlicher oder schicker gestalten. Ist aber halt auch was, wo ich auch dazu sagen muss, da kenn ich halt jetzt diese Umgebung die wir nutzen, das ist Eclipse. Da weiß ich jetzt halt nicht wie andere das lösen. Das ist mit Sicherheit auch etwas, wo es auch sehr abhängig ist vom Anwendungsfall, was man visualisieren müsste oder könnte. Oft geht es ja darum in der Programmierung, um irgendwelche logischen Entscheidungen, die man nachvollziehen möchte oder um irgendwelche Bedingungen letztendlich. Wann wird eine Schleife abgebrochen, bei welchem Eingangsdatum mache ich was, bei welchem Button Klick soll was passieren. Und das, keine Ahnung in welcher Form, etwas schicker zu machen, da vielleicht sich selber sozusagen Elemente zusammenziehen zu können in einer Visualisierung, die dann irgendwie vielleicht einen Datenfluss abbilden oder so, keine Ahnung, das könnte ich mir schon irgendwie vorstellen, dass man sowas machen kann. Aber auch da letztendlich muss ich wieder auf meinen Quellcode zurückkommen. Das ist immer das, was ich so sehe. Sonst hilft mir das nichts. Das ist ja zum Beispiel, wenn ich zum Beispiel Debugging mache, dann weil irgendwie vielleicht mein Quellcode nicht richtig funktioniert an der Stelle und ich will die Stelle finden, wo der nicht richtig funktioniert. Und das ist halt genau der Quellcode.

I: Ich hab noch eine generelle Frage zur Visualisierung, weil du vorher schon die Städtemetapher erwähnt hast und gerade gibt es ja diese Inseln. Und diese 2D Visualisierung die es davor gab war eher auf einem abstrakteren Niveau mit Knoten und Verbindungen. Hast du die mal gesehen?

B: Ja die hatte ich auch mal gesehen, ja.

I: Was würdest du denn so spontan sagen, fändest du unterstützender?

B: Das ist eine gute Frage. Es kommt ja ein bisschen durch, dass ich den Anwendungsfall noch gar nicht sehe, weil mir beides noch nicht wirklich gezeigt hat, dass es mich weiter bringt. Das ist aber ja meine Sichtweise als Entwickler. Ich habe immer ein bisschen den Eindruck dass diese Metaphern (...) Man bildet auf diese Metaphern ja letztendlich irgendeine Quantifizierung ab von Lines of Code oder über vielleicht Distanzen irgendwie die Distanz von Code zueinander. Was hat wie viel miteinander zu tun und so weiter. Das kann vielleicht auf irgendeiner anderen Ebene interessant werden. Ich bin kein Softwarearchitekt. Das haben wir auch nicht in Reinform in unserem Team. […] Aber das gibt es vielleicht in riesigen Softwareprojekten in großen Firmen, dass es da den Softwarearchitekten gibt. Der hat vielleicht da Interesse an so einer Sichtweise, sage ich jetzt mal. Ich als Entwickler, der quasi zwar auch in so Architekturentscheidungen eingezogen ist und auch konzeptionell arbeitet, aber dann auch wirklich das ganze implementiert und umsetzt, ich denke irgendwie halt mehr aus der Implementierung heraus, aus dem Quellcode heraus. Und ich habe auch sozusagen da glaube ich einen anderen Entwicklungszyklus dadurch. Also ich kriege nicht vom Architekt vorgesetzt so muss es aussehen und das ist dann schon gesetzt. Sondern ich kann jederzeit steuernd wieder eingreifen. Wir versuchen natürlich auch so gut wie möglich zu planen aber wir haben

halt einfach diese Rollen nicht klar abgegrenzt. Ich glaube deshalb ist das auch ein Punkt, warum ich das auch so ein bisschen anders sehe. Diese Sicht eines Softwarearchitekten, der vielleicht gar nicht richtig in den Quellcode guckt, die habe ich irgendwie nicht. Und deshalb hat mir bis jetzt beides nicht so richtig weiter geholfen. Und ich glaube dass letztendlich, wenn es dann einen sinnvollen Anwendungsfall gibt, dann muss man anhand des Anwendungsfalls entscheiden, was macht da Sinn. Also ich kann mir vorstellen, dass es Anwendungsfälle gibt, wo so eine Städtemetapher Sinn macht, weil genau das was ich halt mit unterschiedlich hohen Gebäuden oder Gebäudeformen oder -abständen Visualisieren kann, halt genau das ist, was ich auch sehen möchte. Ich kann mir aber auch vorstellen, dass halt so eine abstrakte Visualisierung, wo ich vielleicht auch nur zwei Aspekte zum Beispiel Größe von Kreisen und Abstände zueinander oder so halt sehr schnell erfassen möchte, dass es da dann einfach mehr Sinn macht sowas zu haben. Da könnte ich jetzt nicht sagen, grundsätzlich finde ich das oder das besser.

I: Das war dann eigentlich auch schon meine letzte Frage. Zusammenfassend kann man dann eher sagen, dass du kritisch bist, ob so eine Softwarevisualisierung insgesamt bei der Einarbeitung helfen könnte. Ansonsten ist meine allerletzte Frage immer nach der wichtigsten Anforderung an eine Softwarevisualisierung zur Einarbeitung, aber du würdest im Moment einfach sagen, dass man das nicht brauch?

B: Ja, genau. Ich verschließe mich da nicht. Also, wenn mir jemand zeigt, dass das weiterhilft, dann klar, gerne. Aber vielleicht bin ich da zu wenig Visualisierungsmensch und zu wenig Visionär um in dieser Visualisierung eine Zukunft zu sehen. Aber ich schließe nicht aus, dass man irgendetwas visualisiert, das mir weiterhilft. Es ist halt nur, dass die Ideen, die es bis jetzt gab, mir nicht weitergeholfen haben. Ich den Mehrwert nicht erkennen konnte. Für diesen Anwendungsfall Einarbeitung in Software. Ich sage jetzt mal ich habe die IslandViz schon oft auf irgendwelchen Events mit präsentiert, auf Konferenzen oder so. Das ist immer gut angekommen und ich glaube ich habe das durchaus auch immer begeisternd verkaufen können. Zum einen weil es halt eine coole Sache ist überhaupt eine Visualisierung zu machen. Das auch automatisch zu generieren aus einem Quellcode oder einer Softwarearchitektur. Und weil es schön ist, dass wir daran forschen, das ist ja auch ein Trend sowas zu machen. Und, das ist ja auch immer noch so eine Sache gewesen, dass man gesagt hat, der Anwendungsfall ist noch nicht klar. Das könnte die Einarbeitung sein, das könnte aber auch die Managementebene sein, der man verkaufen möchte "Hey, das ist echt komplex, was wir hier machen". Und dass das komplex ist, das sieht man, weil halt sehr viel passiert vor dem Auge. Auch das muss man etwas differenziert betrachten. Das ist natürlich auch abhängig davon, was man visualisiert. Jetzt habe ich mit RCE ein Projekt wo ich ganz viele Quellcodedateien habe, die alle untereinander Abhängigkeiten habe. Dadurch habe ich ganz viele Inseln und ganz viele Verbindungen zwischen den Inseln und das ist dann, wenn man das das erste Mal sieht vielleicht imposant. Das hat aber letztendlich nichts mit der Qualität oder der Funktion der Software zu tun. Letztendlich auch nicht mit der Qualität des Quellcodes. Weil, ich sag jetzt mal, wenn ich das will als Entwickler, dann kann ich ganz viele Klassen und Abhängigkeiten schaffen um ein schönes Bildchen hinzuzaubern und meine Software kann drei plus drei rechnen, um das mal zu vereinfachen. Und ich meine da ist es dann so, wir haben hier jetzt nicht gezielt entwickelt um so eine Visualisierung schön zu machen. Das ist schon real was man da sieht und wenn man dann sozusagen die richtigen Werte nimmt um zu visualisieren, dann gibt das ja schon das wieder, wie es ist. Wir haben eine komplexe Software und wir haben eine komplexe Implementierung. Das passt schon zusammen. Und dieser Wow-Effekt bei vielen, denen man das auf so Konferenzen mal aufgesetzt hat, die das gesehen haben, die waren beeindruckt. Auch Softwareentwickler, die auch verstehen, was abgebildet wird. Die sind dann schon beeindruckt. Und es haben auch viele gesagt, das sieht toll aus und das ist ja cool, dass man sowas macht. Also ich sperre mich da weiß Gott überhaupt nicht, aber diesen Punkt Einarbeitung, den sehe ich da noch nicht.

I: Alles klar. Hast du noch irgendwelche Fragen oder andere Anmerkungen? Ich wäre ansonsten durch mit meinen Fragen.

B: Ja, ne ich habe eigentlich auch nichts mehr und hoffe, dass ich nicht zu destruktiv jetzt war.

I: Auf gar keinen Fall, es kommt ja darauf an zu erfahren, was ihr als Nutzer dann davon denkt. Also vielen Dank für deine Zeit und Eindrücke.

## Interview 4

[…]

I: Hast du schon mal mit Softwarevisualisierungen gearbeitet?

B: Ja.

I: Was für Visualisierungen waren das und wofür hast du die benutzt?

B: Prinzipiell Visualisierungen, 2D, für eigene Software. Zum Beispiel um Verknüpfungen von Komponenten zu visualisieren, in der Dokumentation. Vollautomatisch quasi, also aus der Software generiert. Und IslandViz in der initialen Version.

I: Die IslandViz hast du gesehen oder daran mitgearbeitet?

B: Die habe ich gesehen. […]

I: Bei den 2D-Visualisierungen, die du eben erwähnt hast, hast du die selbst erstellt?

B: Ja.

I: Dann hast du das während der Arbeit an der Software auch selbst oft genutzt?

B: Das war dann halt fest eingebaut und wenn man die Dokumentation erstellt hat, dann war das fest eingebaut.

I: Und der Nutzen davon war dann, später die Verknüpfungen der Software zu erkennen?

B: Man konnte dann halt auf ein großes Bild gucken und sehen, wie die einzelnen Bestandteile der Software ineinandergreifen. Also quasi verfolgen, wie die Dinge ablaufen in dieser Software.

I: Was war der ursprüngliche Gedanke, warum du das entwickelt hast? War das für dich selbst, um das nochmals nachzuvollziehen, oder hauptsächlich für andere?

B: Sowohl, als auch. Wir hatten viele Komponenten, die wir vielfältig verwendet haben. Mit vielen Daten die rein und raus gingen und man hat die halt um etwas zu berechnen (…) hat man halt diese Komponenten genommen und miteinander verbunden in irgendeiner Form. Auch mehrfach um dann am Ende Metriken zu erhalten. Man hat das quasi programmiert und diese Visualisierung oder Darstellung hat dann einfach gezeigt, wie die Komponenten miteinander verbunden sind. Dann konnte man im Endeffekt sehen, wenn ich vorne die Daten hier rein tue, welche Daten brauche ich, damit am Ende dieses Ergebnis rauskommt. Das war einfach eine Übersicht für mich, aber auch für die Nachfolgenden.

I: Du hast das ja selbst entwickelt, weshalb die Frage möglicherweise merkwürdig klingt: Die Visualisierung fandst du dann auch für deine Arbeit hilfreich?

B: Das hat mir oder uns auch selbst geholfen, beim aktiven entwickeln genau. Weil wir dann auch sehen konnten, haben wir irgendwo eine Komponente, die zu viel macht, die wir aufteilen müssen oder so etwas halt.

I: Also hat die Visualisierung dir bei der Entwicklung selbst und auch in der Rückschau geholfen?

B: Sowohl als auch, ja.

I: Und gibt es weitere Aufgaben wo du denkst, dass du von einer Visualisierung profitieren könntest?

B: So eine Architekturvisualisierung oder?

I: Zunächst mal noch gar nicht darauf limitiert, weil das kommt ja auch immer bisschen darauf an, an welcher Aufgabe man steckt. Es ist eher die Frage gemeint, ob du schon mal vor dem Quellcode saßt und dir dachtest, dass dir bei einer bestimmten Aufgabe eine Visualisierung weiterhelfen könnte.

B: Genau bei solchen Aufgaben, also wie hängen Komponenten zusammen. Wenn ich die Software nicht kenne muss ich wissen, wie hängen die einzelnen Bestandteile miteinander zusammen, wie werden die gegenseitig benutzt. Was benutzt was. Das ist immer die Herausforderung bei größerer Software das herauszufinden. Vor allem, wenn das nicht simpel zu lesen ist. Manchmal sind da viele Dinge, die passieren so automatisch und dann sieht man halt nicht, wie das initial funktioniert. Das sind dann einfach nur lose Komponenten, aber die werden dann trotzdem miteinander verbunden intern. Man sieht das aber nicht direkt. Da ist sowas dann hilfreich, wenn man direkt sieht, okay.

I: Für Bacardi habt ihr das aber nicht?

B: Nein.

I: Und wie löst du zurzeit die Aufgaben, bei denen du denkst, dass sie durch eine solche Visualisierung unterstützt werden könnten?

B: Im Moment haben wir Prozesse, die wir manchmal ausarbeiten und visualisieren. Also als Prozessketten aufschreiben und dann implementieren wir die Bestandteile. Aber da gibt's noch keine Visualisierung, das ist manuell, leider.

I: Und an welchem Punkt wäre dann der Quellcode an sich geeigneter? So wie das gerade bei dir klang, gibt dir die Visualisierung einen Überblick über Zusammenhänge und danach arbeitest du wieder ganz normal am Quellcode weiter?

B: Also für mich ist das meistens so: Wenn ich dann feststelle, wir haben einen Teil, der ist zu komplex, was man vielleicht durch so eine Visualisierung erkennen kann. In dem ersten Beispiel war das so, da konnte man sehen, es kommt ganz viel rein und ganz viel raus, und irgendwie ist was doppelt und wir brauchen ein Teil davon auch an anderen Stellen. Das ist halt durch die Visualisierung klar geworden. Aber spätestens, wenn man dann den Teil identifiziert hat, der Probleme hat, muss man da dann halt reinschauen. Und irgendwo auch verstehen, was macht denn das und warum braucht es überhaupt alles das, was es als Eingabe hat? Dann muss man dann doch irgendwann in den Code gucken. Es ist mehr zum Identifizieren der Problemstellen, sag ich mal. Refactoring nennt man das.

I: Gehst du dann mit einer Vorahnung, wo die Problemstellen liegen könnten in die Visualisierung?

B: Ja. Also wenn ich weiß, das gilt für diesen Bereich, dann weiß ich, das grenzt es schon irgendwie ein und dann sind das halt zehn verschiedene Teile der Software, die irgendwie zusammenhängen. Das kann man schon relativ klar dann ausmachen, eigentlich. Sollte man. Es bringt mir nichts immer alles zu sehen. Hatten wir damals auch so gemacht. Also wir hatten eine Gesamtübersicht, die war immer sehr wuselig, da war sehr viel drin. Man konnte aber schön sehen, wo etwas zu komplex ist. Wir hatten aber auch für die einzelnen Berechnungen auch immer noch eine einzelne Übersicht.

I: Der Nutzerkontext, für den die neue Visualisierung entwickelt werden soll, ist die Einarbeitung in eine neue Software. Dabei wäre die erste Frage, wie oft du dich bereits in eine neue Software eingearbeitet hast. Also arbeitest du dich einmal ein, wenn du in eine neue Abteilung kommst, oder passiert das öfter?

B: Das ist unterschiedlich. Also es gibt bestehende Systeme, wie RCE oder in dem Fall Bacardi. Da muss man sich dann einarbeiten, verstehen was ist das alles, wie hängt das alles zusammen. Teilweise aber auch in einem weiteren Kontext. Es gibt ja unterschiedliche Schwerpunkte einer Software. Manchmal sind es ja fachliche Punkte, die man noch nicht versteht. Bacardi ist es auch mit Mechanik so ein Ding, da kenne ich mich halt auch nicht mit aus. Und das ist dann so ein fachlicher Bereich, da versteht man den Code nicht sofort. Man muss halt erst mal die Architektur drum herum, also wie ist das überhaupt aufgebaut, verstehen. Das ist dann erst mal der Aufwand. Manchmal ist es auch so, dass man irgendwelche Zusatzsoftware hat, die da benutzt wird in der Software. Und auch in die muss man sich erst mal einarbeiten. Einarbeitung kommt eigentlich an jedem Projekt nur an einer Stelle vor. Sofern es schon existiert.

I: Wenn du jetzt Bacardi als Beispiel vor Augen hast, wie hast du dich dort eingearbeitet? Hauptsächlich mit dem Quellcode oder gab es noch weitere Dokumentationen?

B: Vielfach war das Kollegen nerven. Beim ersten Punkt so, wo muss ich hin gucken. Also erkläre mir mal so grob, was hier denn überhaupt drin ist und wo das liegt. Verzeichnisstrukturen verstehen und was da wie wo organisiert ist. Das ist so der erste Schritt, damit man überhaupt weiß wo man reingucken muss, wenn man was Bestimmtes sucht. Dann nimmt man sich irgendwo eine Aufgabe und dann sucht man sich die Teile zusammen, die man braucht. Und dann versucht man irgendwie zu verstehen, wie denn da meinetwegen die Daten durchfließen oder wie die Dinge aufgerufen werden. Schrittweise. Das ist meistens mein Ansatz. Besser ist, wenn man schon fertige Tests hat. Dann kann man sich das zum Teil live angucken, wie das funktioniert. Dann kann ich wirklich sagen "führe das jetzt mal aus und zeige mir, wo du durch den Code läufst". Und dann sieht man auch logischerweise, weil das sehr viele Dateien sind, viele Punkte und dann springt der auch zwischen diesen Dateien. Dann bekommt man einen Eindruck davon, wie das funktioniert. Und wenn man dann den nächsten Workflow, was auch immer man da gerade macht, nimmt. Dann sieht man halt springt er hier rein und jetzt geht er dort rein und nimmt diesen Algorithmus oder was auch immer. Man erkennt dann so einen groben Fluss in der Software und dann kann man irgendwann auch damit anfangen, damit zu arbeiten.

I: Vorher hast du auch gesagt, dass je nachdem, was inhaltlich zur Software gehört, dass du dich da auch einarbeiten musst. Wann passiert das? Bevor du dir den Code überhaupt anschaust?

B: Im Groben, ja. Aber nicht im Detail. Wenn man dann irgendwann mal was machen soll, und man kommt an eine Stelle an der technischer Code ist, dann sorgt man entweder dafür, dass das der Kollege macht, der sich gut damit auskennt. Das bringt mir halt nichts, wenn ich da Orbit Mechanik machen muss, das kann der Kollege besser als ich. Wichtiger ist es an der Stelle zu wissen, wer ist dafür verantwortlich. Also wer kennt den Code an der Stelle und mit wem muss ich vielleicht reden, wenn ich da nicht weiter kommen.

I: Und woher weißt du das? Weil du die Kolleg*innen in deiner Abteilung eben kennst?

B: Ja im Grunde schon. Also jeder hat so seine Schwerpunkte. Also das sieht man auch immer sehr schön. Wo Kollegen arbeiten, welche Sachen die machen. Wo da Interessen und Schwerpunkte liegen, sag ich jetzt mal. Und manchmal ist das ja auch klar definiert. Bei uns ist das nicht so.

I: Wenn du dann anfängst dich einzuarbeiten, hast du eben gesagt, dass du Aufgaben löst. Das heißt du suchst dir zunächst erste kleine Aufgaben, an denen du dich versuchst, um die Software kennenzulernen?

B: Die bekommst du. Du kriegst halt irgendeine Aufgabe und du versuchst das halt umzusetzen und das dauert dann.

I: Schaust du dir davor aber erst mal die Softwarearchitektur oder den Aufbau der Software an oder fängst du mit dieser ersten Aufgabe an, startest dort in den Code und gehst dann von dort aus vor?

B: Man guckt erst mal vorher überhaupt alles durch. Was gibt es überhaupt. Wo liegt Doku. Man guckt sich die Dinge alle vorher an, erst mal oberflächlich. Liest vielleicht nochmal irgendwelche Sachen nach. Aber das ist erst mal ganz normale Einarbeitung für ein Projekt. Was ist das überhaupt, wer arbeitet hier und was machen wir hier überhaupt. Aber man geht nicht ins Detail. Man guckt sich nicht allen Quellcode an. Klar, man guckt da mal durch so, um einen Eindruck zu bekommen, wie das grob aussieht aber nicht im Sinne eines technischen Verständnisses.

I: Also würdest du sagen, dass eine Einarbeitung ins Projekt, also wer alles daran arbeitet etc., und die Einarbeitung in die Software zwei getrennte Prozesse sind?

B: Naja das baut aufeinander auf. Du musst das Projekt und wer an was arbeitet verstehen. Vorher kannst du nicht an der Software arbeiten. Also nicht im komplexen Sinne, das geht nicht. Weil du dann nicht weiß, wen du fragen musst oder wo du Informationen findest. Das muss man erst mal grob herausfinden und verstehen. Damit ist es ja auch noch nicht getan. Also eine Software kann man nicht einfach runterladen und daran arbeiten, das geht nicht. Ich muss auch erst mal diese Entwicklungsumgebung aufsetzen. Das ist ja auch nicht so trivial, da lernt man auch schon eine Menge, wenn das nicht gut dokumentiert ist.

I: Und dann löst du die erste Aufgabe. Entwickelt sich das Verständnis für die Software dann mit jeder Aufgabe weiter? Gibt es dann einen Punkt nach mehreren Aufgaben an dem du sagst du hast die Software verstanden?

B: Ne, es ist eher so, dass man irgendwann verstanden hat, wo die Dinge sind, wer was macht, wer die Ahnung hat und man ziemlich genau weiß, welche Datei man nicht verstanden hat. Eher so rum. Es ist eher so, dass, oh oh, jetzt muss ich an diese Datei ran. Da kenne ich mich nicht aus, aber ich weiß wen ich fragen kann.

I: Es gibt also keinen Zeitpunkt, an dem du sagst, die Einarbeitung ist vorbei? Wenn du in eine neue Stelle im Code schaust, musst du dich dort wieder neu einarbeiten?

B: Ja, das wird zwar immer weniger, und man weiß dann schon, ach hier muss ich nicht weiter gucken, da weiß ich was da passiert. Es wird natürlich effizienter, aber es gibt immer wieder stellen wo man sagt, was ist das jetzt hier. Was muss ich hier machen. Das ist aber unterschiedlich, das kommt immer aufs Projekt an. Bacardi ist da recht komplex und RCE stelle ich mir sehr ähnlich komplex vor.5

I: Du hast vorhin gesagt, dass du dir erst einen groben Überblick verschaffst, den Code grob liest. Welche Informationen über die Software musst du am Anfang wissen? Gibt es so einen Standardkatalog an Fragen, die du als erstes über die Software wissen willst?

B: Also es ist extrem davon abhängig, mit was man gerade arbeitet. Das Vorgehen ist da sicher anders bei Bacardi, was Python ist, und bei RCE, was Java ist. Aber man guckt sich auf jeden Fall die Struktur der Dateien, also Verzeichnisse an. Wie sind Dinge benannt, sowas ist meistens benannt. Meistens entwickelt sich ja immer so eine Art Sprache, ein System, wie man Dinge benennt. Und daran erkennt man auch immer schnell, wo man gerade ist. Das gehört ein bisschen zur Architektur und wenn man das gut macht, ist das sehr einleuchtend. Und damit bekommt man auch ein Gefühl dafür, wo Dinge sind. Da guckt man schon immer drauf, also wie ist das strukturiert. Schwierig wird es, wenn‘s das nicht ist.

I: Aber das würde man sehr schnell erkennen, wenn es nicht strukturiert ist?

B: Ja.

I: Für die Einarbeitung in die Software, ist es da am Anfang relevant, wie die Historie ist?

B: Ja, auf jeden Fall. Also finde ich schon.

I: Kannst du auch sagen, warum das relevant ist und was daran relevant ist?

B: Weil man in der Historie durchaus sieht, wo die Intention der anderen Entwickler liegt, finde ich. Das ist vielleicht ein bisschen Meta, aber es hilft einem bei der Zusammenarbeit mit den Kollegen. Es ist vielleicht nicht ganz die Softwareentwicklung, aber das gehört irgendwie auch dazu. Ist der soziale Aspekt dahinter. Man kann sehr gut sehen, wie die Kollegen arbeiten. Und wo es hergekommen ist vor allem, was war der initiale Sinn dieser Software und wie hat sie sich weiterentwickelt. Das ist manchmal schon sehr interessant. Also ich gucke auch gerne in ältere Dokumente rein, einfach nur um zu sehen, wo kommt das her. Und eigentlich ist das auch guter Stil, das zu dokumentieren. Aber es ist auch interessant, wie sich die Architektur verändert über die Zeit. Auch daran kann man sehen, wie sich die Software entwickelt über die Zeit. Auch daran kann man erkennen ob Software besser oder schlechter wird. Üblicherweise sieht man am Anfang sehr viele Dateien, das ist sehr chaotisch. Wenn man weiter geht, fangen so Module an, Namen werden einheitlicher, das kann man schon sehen.

I: Und das findest du hauptsächlich für die Kommunikation mit den anderen relevant?

B: Genau. Man weiß dann genau, wenn jemand anderes sagt, vor fünf Jahren haben wir mal (...) Wenn man dann noch nicht da war, aber man weiß dann, es war auch chaotisch vor fünf Jahren, also ist es vermutlich heute nicht mehr relevant. Oder ja stimmt, ist ein guter Punkt, wurde damals offensichtlich angesprochen, war wichtig, ist aber in Vergessenheit geraten. Solche Sachen gibt es ja auch hin und wieder. Oder was für Fehler haben wir gemacht. Also wenn der eine Kollege sagt, wir haben das schon mal so versucht und das funktioniert hier nicht, das kann ja ein valides Argument sein. Da kann man schlecht sagen, ne hast du Unrecht. Das muss man vielleicht mal nachschauen.

I: Du hast jetzt schon öfter erwähnt, dass es mehr oder weniger das A und O ist, mit anderen zusammenzuarbeiten und nachzufragen. Als du dich in Bacardi eingearbeitet hast, würdest du dann sagen, das war eher Teamarbeit?

B: Bei Projekten mit mehreren Leuten, auf jeden Fall, ja. Sich in große Systeme einarbeiten ohne die Kollegen, das geht nicht. Ich denke das bestätigen dir auch die Kollegen von RCE. Da ist man auf die erfahrenen Kollegen angewiesen.

I: Wenn du die ersten Aufgaben löst, machst du dir dann Notizen um dein Verständnis zu erweitern? Oder hoffst du, dass die Dokumentation gut genug ist und du daraus deine Informationen hast?

B: Letzteres hoffe ich nie. (lacht). Also ich hoffe es immer aber ich erwarte es nicht. Ist meistens eher mental bei mir. Ich mache jetzt keine Textnotizen, eigentlich nicht. Ganz am Anfang mache ich das bei der Übersicht - so es gibt hier folgende Dinge - aber das brauche ich schnell dann auch nicht mehr. Danach ist es eher so eine Idee von dem, was will ich machen. Vielleicht schreibe ich mir dann auf, welche Schritte ich zu tun habe für meine Arbeit jetzt. So grob, ich muss jetzt das machen, das machen, das machen und dann überlege ich mir, wo steht das denn. Aber direkt bei der Arbeit selbst - dann ist es eigentlich eher ein Workflow in den man ganz normal rein schreibt: Jetzt mache ich das hier, weil... Aber das schreibt man eher für die anderen und nicht für sich.

I: Und bis sich dieses mentale Bild entwickelt hat, von dem du gesprochen hast, schaust du dir am Anfang alles grob an?

B: Genau, und ich vergleiche auch mit Ähnlichem. Also, wenn das geht und meistens bekommt man Aufgaben, die schon mal in ähnlicher Form gemacht wurden. Und dann wird einem meistens auch gesagt, guck mal da und da und da rein, da ist das so ähnlich gemacht, und richte dich dann danach. Das ist so üblich. Und die Kollegen gucken nachher dann halt drüber und sagen dir: ne, wir machen das hier anders an der Stelle. Das hilft meistens.

I: Dann komme ich jetzt zu dem Punkt, an dem ich versuchen möchte die Themen der Einarbeitung und der Möglichkeiten einer Visualisierung zu verbinden. Hast du das Gefühl, dass dir eine Softwarevisualisierung gleich bei den ersten Schritten der Einarbeitung helfen könnte?

B: Ja so eine typische Visualisierung ist ja so eine Systemübersicht, also wirklich so eine UML-Systemübersicht oder vielleicht auch irgendwelche technischen Übersichten, die meistens zur Doku gehören, wenn sie gut gemacht ist. Und die wird auch meistens herangezogen um einem neuen Kollegen zu erklären, was sind denn die Sachen, die wir haben und um das dann darzustellen.

I: Und eine Visualisierung der Struktur und Architektur, durch die man sich auch selbst durcharbeiten kann?

B: Ich weiß nicht genau worauf du hinaus willst. Für die Einarbeitung finde ich die momentane Version, die hier gemacht wurde noch nicht ausreichend (redet von der IslandViz)

I: Genau, deshalb versuchen wir ja eine neue Visualisierung zu entwickeln, für die Einarbeitung. Aber würde dir vom Prinzip her eine Darstellung der Architektur helfen für die Einarbeitung?

B: Ich finde die Metapher gut, mir fehlt die Möglichkeit der Interaktion. Also für die Einarbeitung brauche ich jemanden, der mir dort Dinge zeigen kann. Es bringt mir nichts das aufzusetzen und mich durch die Dinge durchzuklicken, weil dann finde ich die Teile nicht. Also nicht in der aktuellen Variante. Da sehe ich halt okay es gibt hier so einen großen Block Inseln da und dann gibt es die Verbindungen da und da. Aber das ist noch nicht ausreichend genug. Wenn man das selbstständig machen wollen würde, dann müsste das Ding im Prinzip sowas unterstützen wie: Wenn ich jetzt hier etwas rein tue, was wird dann alles dadurch aktiviert. Man müsste dann quasi, was ich meinte, wenn man so Tests hat kann ich die Software an bestimmten Stellen ausführen und mir angucken, wie das da durch springt. Wenn das da drin wäre, könnte man die Visualisierung vielleicht eigenständig benutzen. Das wäre vielleicht eine Möglichkeit.

I: Aber wäre das auch ein Benefit gegenüber den Tests

B: Das wäre eine Ergänzung. Aber nur um eine - vielleicht schnellere - Übersicht zu bekommen. Wenn man das in so einer Software in einer Entwicklungsumgebung macht, dann springt man da zwar durch aber da öffnen sich im Prinzip ganz viele Textdateien. Und da muss ich halt gucken, jetzt hat sich die die und die geöffnet und wenn ich so eine Aktivierung machen könnte, wenn ich sage ich führe diesen Test aus, was wird da jetzt alles angesteuert? Und ich würde in einer Übersicht sehen, diese fünf sind das. Man könnte das vielleicht noch schrumpfen auf diese fünf und ins Detail gehen. Und könnte sich dann die einzelnen fünf anschauen. Das wäre interessant. Quasi eine Testausführung auf den Inseln. Also das wäre sehr sinnvoll zum Reinkommen.

I: Und würdest du die Visualisierung vom Quellcode trennen. So wie das jetzt in diesem Beispiel ist oder würdest du dir wünschen, dass man das besser verknüpfen kann? Wenn du zum Beispiel durch Interaktionsmöglichkeiten auf einen Teil der Visualisierung klicken könntest und dir den Quellcode anzeigen. Würde das nützen oder würdest du dir nur diese Visualisierung anschauen, sie danach beiseitelegen und dann einfach am Code weiterarbeiten?

B: Also es ist nicht schlimm, wenn man da mal rein gucken kann. Wenn man sich ein bisschen auskennt und man dann irgendwie eine große Insel sieht und dann kann ich halt einmal in den Code reingucken, wenn das geht. Da ist eine riesige Klasse und man guckt da einmal rein, scrollt da einmal durch. Da kriegt man relativ schnell den Eindruck - das ist aber komplex. Das ist dann ein Indikator dafür, da mal genauer reinzugucken, aber das würde ich dort dann nicht mehr machen.

I: Bei dieser Inselmetapher werden ja Klassen als Häuser dargestellt. Und würde dann nicht auch schon reichen einfach hohe Häuser darzustellen und du würdest erkennen, dass das komplex ist?

B: Es gibt bei statischer Codeanalyse so Metriken für Komplexität. Wie viele Schleifen sind da drin, wie viele Bedingungen. Ich finde dann Zahlen eindrücklicher. Wenn man irgendwie sagen kann, komplex finde ich, wenn da 40 Schleifen drin sind und diese Klasse hat halt mehr. Oder die Klasse hat mehr als 30 Sprünge in  Bedingungen oder so. Das wäre als Zahl wesentlich sinnvoller als Angabe. Dass man so Metriken darüber einholen kann. Also klar, man kann die vielleiht auch irgendwie visualisieren aber die Zahl sollte irgendwie immer da oder zumindest abfragbar sein.

I: Kannst du dir auch vorstellen, dass man zusammen an so einer Visualisierung arbeitet? Gibt es einen Benefit davon, dass sich das mehrere Personen gleichzeitig anschauen können?

B: Für eine Einarbeitung fände ich das sogar sehr wichtig. Also wenn man das mit diesen Tests kombiniert, wäre das eine wunderbare Sache, dass man sagen kann, komm setze dir so ein AR-Ding auf und ich setz mir eins auf. Und dann sagt man, pass auf du arbeitest jetzt hier und wenn du diesen Test ausführst, dann siehst du hier aktiviert sich was und dann sind diese Komponenten beteiligt. Dann kann man das halt zeigen. Wichtig ist dann halt, die Komplexität immer weiter zu reduzieren. Es reicht halt nicht da rein zu zoomen, sondern es muss dann auch wirklich irgendwie, dass man sagen kann, okay jetzt zeig mir aber auch irgendwie halt nur die relevanten Teile an oder so. Sonst wird es einfach zu viel.

I: Also, dass man Dinge ausblenden kann?

B: Ja, genau.

I: Gibt es noch weitere Funktionen, wie Dinge ausblenden zu lassen, die du dir so spontan wünschen würdest von einer Visualisierung?

B: Vielleicht der Punkt, dass man eine Art Screenshot machen kann. Also irgendeine sinnvolle 2D-Repräsentation von dem, was man sich erarbeitet hat, als Reminder mitzunehmen. Ich stelle mir jetzt gerade vor, ich habe mir das alles zusammengesucht, ich habe ein paar Inseln und die habe ich mir gehighlightet. Und dann setzte ich diese Brille wieder ab und setze mich an meinen Rechner und denke mir - wie war das nochmal? Das wäre dann ganz praktisch, wenn man dann sagen kann, schicke mir mal kurz ein Bild von dem was ich als letztes gesehen habe. Dann kann ich da nochmal kurz drauf gucken, so als Reminder quasi. Das wäre interessant.

I: Und generell, gerade hast du viel von VR und AR gesprochen. Es gäbe ja auch die Möglichkeit die Visualisierung einfach am PC zu machen. Was würdest du denn eher nutzen?

B: Wenn ich die Visualisierung mit dem Tisch hätte? Dann würde ich die am PC nutzen, weil ich faul bin. Also für mich alleine würde das reichen.

I: Und glaubst du auch, wenn man das am PC zeigen könnte - das muss ja gar keine Visualisierung mit einem Tisch sein, sondern einfach irgendeine Art der Visualisierung. Würde das auch reichen? Weil man könnte das auch gemeinsam anschauen.

B: Ja klar, natürlich.

I: Und auch beim gemeinsamen Arbeiten, da würdest du eher den PC wählen statt AR, wie du es vorher erwähnt hast?

B: Ich denke schon, ja. Weil der Wechsel zwischen jetzt gucken wir mal in den Code. Und man hat halt so viele Features, die kann man nicht in so eine Visualisierung packen. Die sind einfach schöner in einer IDE zu erklären, weil man da auch schneller navigieren kann, das ist einfach so. Und der Wechsel wäre halt einfacher. Das eine Programm klein machen, aufs nächste gehen, weiter erzählen, nochmal zurückgehen "Jetzt sind wir hier gerade an der Stelle". Das geht dann einfacher, das ist einfach so. Aber zum Einarbeiten mit zwei Leuten kann man das trotzdem nutzen mit einer AR oder VR Brille zur Übersicht.

I: Okay, wir haben jetzt die ganze Zeit mehr oder weniger über die IslandViz auch gesprochen. Und du hast die Metapher auch erwähnt. Hast du lieber eine Metapher bei der Visualisierung oder würdest du eine abstraktere Darstellung bevorzugen?

B: Ich finde die IslandViz, was das angeht eigentlich schon ganz praktisch. Also weil das diese Abstraktion dieser Hierarchien sehr schön darstellt und mit Kugeln, Quadrate, was auch immer wird das immer sehr, sehr abstrakt. Das ist eigentlich genauso abstrakt wie irgendwelche Tabs mit Text. Da tut sich dann nichts mehr. Also dann kann ich auch meine Tabs angucken, in welcher Reihenfolge, die sich geöffnet haben. Also für so eine Übersicht und um zu verstehen, ach so die Häuser sind die kleineren Sachen, als die Packages. Ich weiß nicht inwieweit man das dann nutzen kann, das müsste man ausprobieren. Also wie lange das dann sinnvoll ist. Irgendwann weiß man halt, was das ist und braucht das vielleicht gar nicht mehr, aber so als Einstieg vielleicht. Ich denke man kann relativ schnell auf so eine vereinfachte Variante wechseln, die man am Rechner hat. Und irgendwann braucht man das halt auch nicht mehr. Ich könnte mir eher vorstellen, dass das dann interessanter ist im Nachgang, so eine Visualisierung. Wenn man die mit mehr als zwei Leuten betrachtet in einem Teammeeting. Wenn man aktiv daran Dinge erarbeitet mit mehreren Leuten gleichzeitig. Um ein gemeinsames Verständnis zu haben, worüber man gerade redet. Damit man was zeigen kann. Das ist besser über eine Visualisierung als über einen Code.

I: Und wenn du sonst meintest, dass man das außerhalb von solchen Teammeetings später nicht mehr so nutzen würde. Denkst du, auch bei einer neuen Aufgabe drei Jahre später oder so, dann hast du die Abhängigkeiten und wie alles zusammenhängt im Kopf? Dann würde dir das auch nicht mehr nützen, nochmal rauszoomen zu können um zu sehen, wo du gerade steckst?

B: Das eher nicht. Maximal, wenn man eine Historie darstellen kann in der Architektur. Da könnte man bei so einer Inselmetapher sehr schön sehen, wie hat sich die Arbeit, die wir gemacht haben ausgewirkt und gibt es jetzt irgendwelche neuen Hotspots, wo wir vielleicht mal gucken müssten. Das wäre noch interessant, da im Nachgang mal drauf zu gucken. Gemeinsam zu sagen, guck mal hier, das Ding da hinten wird immer größer, irgendwas machen wir da falsch. Dafür ist es sicherlich nützlich, weil das sieht man in normalen Entwicklungsumgebungen selten. Zumindest mit den Mitteln, die wir in so einer Umgebung haben. Es gibt andere Möglichkeiten das auch rauszufinden, wenn man möchte.

I: Dann jetzt am Ende noch meine letzte Frage, die alles nochmal bisschen zusammenfassen soll: Was muss so eine Visualisierung können, damit du die das nächste Mal nutzt, wenn du dich wo einarbeitest?

B: Kollaboration durch alle. Also jeder kann da Dinge tun. Und halt hoch selektiv. Dass es Dinge einschränken kann. Dass ich auch mehr Informationen holen kann, wenn ich sie bauche oder Informationshinweise. Und sei es wo noch weitere Doku liegt, oder sowas. Das sind eigentlich so die Dinge, die man haben möchte dann, wenn man es dann nutzt. Für einen alleine, ja, kann man mal am Anfang machen. Aber das ist natürlich subjektiv. Also ich brauche das dann nicht.

I: Alles klar. Hast du noch irgendwelche Fragen oder Anmerkungen zu einem der Punkte?

B: Eine Sache noch, was  halt wichtig wäre für mich, für so eine Visualisierung ist die Nutzbarkeit. Das muss halt einfach handhabbar sein. Ich will mir dann keinen Kopf machen, wie ich da jetzt meine Daten reinbekomme. Ich will einfach sagen: Nehme dieses Projekt und es sollte auch bitte egal sein, was das ist. Ob das jetzt RCE mit Java ist oder Python. Ich will dem einfach relativ einfach sagen können, nehme dieses Projekt und mappe dieses Islandidee folgendermaßen. Relativ simpel, quasi. Und dann baut der mir das und zeigt mir das an. Momentan ist genau das nicht der Punkt. Ich selbst kann Steam starten und kann die Software starten, aber ich weiß, dass man das nicht einfach austauschen kann und das ist dann eigentlich auch so ein Feature, das man haben will. Also, dass man das nicht auf einen Use Case begrenzt, sondern viel dynamischer gestaltet. Ich muss das spontan nutzen können. Wenn ich erst irgendwelche Leute fragen muss, kannst du mir das mal umbauen, damit das jetzt funktioniert. Das bringt nichts, dann passiert das nicht. Da würde ich zehn Mal schneller mich durch meine Texteditoren klicken.

I: Dann Dankeschön für dein Einschätzungen und deine Zeit!

## Interview 5

[…]

I: Hast du schon einmal mit Softwarevisualisierung gearbeitet?

B: Also ich habe halt die IslandViz kennengelernt. […]

I: Okay. Kam es aber schon mal vor, dass du vor dem Quellcode saßt und dir dachtest, dass eine visuelle oder graphische Darstellung jetzt irgendwie praktisch wäre und dir helfen würde?

B: Ja, das ist echt eine gute Frage. Ich finde das auch echt schwierig. Besonders jetzt Software wie RCE in irgendeiner Form zu visualisieren, dass man jetzt wirklich einen Mehrwert davon hat. Also klar, man sitzt oft, gerade am Anfang, wenn man noch wenig Programmiererfahrung hat, dann sitzt man davor und versteht noch wenig bis gar nichts erst mal. Man fühlt sich auch so ein bisschen erschlagen. Demzufolge würde eine Visualisierung schon etwas bringen. Aber wie die jetzt genau aussehen soll, das stelle ich mir auch schwierig vor. Ich meine gut, ich kenne jetzt bisher halt nur die IslandViz. Ob das einem so gut weiterhilft, weiß ich nicht, muss ich sagen.

I: Die IslandViz visualisiert ja die Architektur der Software, also auch Zusammenhänge. Aber es gibt ganz verschiedene Formen der Visualisierung. Zum Beispiel eine Hotspotanalyse des Codes. Dass beispielsweise rot dargestellt wird, was schon sehr oft bearbeitet wurde in letzter Zeit.

B: Ah, eher wie ein Hilfsmittel.

I: Ja genau. Ich sage das nur, dass wir nicht die ganze Zeit von der IslandViz ausgehen.

B: Ja, ich brauche auf jeden Fall noch andere Vorstellungen, was es so gibt. […]

I: Es gibt zum Beispiel Darstellungen, dass der ganze Code extrem klein dargestellt wird. Über Pixel beispielsweise. Und dann wird über Farben dargestellt, was wie alt ist oder oft verändert wurde.

B: Ja das ist ja doch mal interessant, vor allem dann vielleicht nicht zum Einstieg sondern als Entwickler. Weil es interessant ist zu sehen, was habe ich jetzt schon lange nicht mehr angepackt. Wo sind vielleicht Überarbeitungen notwendig. Also das kann ich mir schon vorstellen, wenn es da vielleicht so gehighlightet wird. Also eher dann so zur Qualitätssicherung des Codes.

I: Und hast du dir aber schon selbst manchmal Dinge aufgemalt, also Diagramme über Zusammenhänge?

B: Ja habe ich schon manchmal gemacht am Anfang, aber schon eher wenig.

I: Also arbeitest du am liebsten einfach mit dem Code an sich?

B: Ja. […]

I: Gerade hast du schon erzählt, dass du dich jetzt das erste Mal richtig in ein größeres Softwareprojekt eingearbeitet hast, bei RCE?

B:Ja, genau.

I:Und du bist jetzt hier auch für das Team RCE eingestellt. Dann ist auch davon auszugehen, dass du dich erst mal in kein anderes Softwaresystem hier einarbeitest, oder?

B:Ja genau, das ist meine Planung. Ich denke, wenn man in ein anderes Projekt switchen möchte, hier beim DLR könnte man das schon machen. Aber ich bin erst mal in meinem Team und da möchte ich natürlich auch bleiben.

I: Alles klar und zur Einarbeitung bei RCE. Da hast du hauptsächlich mit dem Quellcode gearbeitet? Oder gab es noch andere Dokumente?

B: Also es gibt ja den Developer Guide, der vom Team geschrieben wurde. Da stehen so ein paar Dinge drin für den allerersten Anfang. Tatsächlich, habe ich so angefangen, dass mir Kollegen den Einstieg ermöglicht haben. Erst mal hat man irgendwie einen kleinen Bugfix. Man sieht das in der GUI. Das war erstmal etwas GUI-seitiges, oder ich wusste, was jetzt dort so zu tun ist. Aber dann den Einstieg zu finden, wo im Code ist jetzt die Stelle, wo ich hin muss, um das jetzt zu fixen. Da bist du als Anfänger komplett überfordert, wenn du die ganze Struktur nicht kennst. Die ganze Packagestruktur ist das. Also die ganzen Namen sind das ja auch. Vielleicht könnte man also überlegen, dass man auch einen besseren Einstieg in das Wording bekommt. Weil das ist ja gewachsen in diesem Projekt. Im Prinzip war dann also der Einstieg, dass ich dann Kollegen gefragt habe: "Wo muss man da jetzt überhaupt danach suchen?" und die haben mir dann so grob gezeigt wo man Suchen muss und welche Stelle. Und sobald man die Stelle dann gefunden hat ist es eigentlich nur noch Programmierung. Also wir verbringen tatsächlich sehr viel Zeit auch damit, die richtige Stelle zu finden, wo etwas zu fixen ist. […] Da sind natürlich so Hilfsmittel für Programmierer, dass sie schneller an die richtige Stelle kommen, wie auch immer das aussehen soll, kann ich mir nicht vorstellen, aber dafür gibts dann andere Leute die sich da Gedanken machen können, aber so etwas wäre sehr hilfreich.

I:Nur, dass ich das richtig verstehe. Du hast zunächst so ein Issue bekommen und suchst dann nach der richtigen Stelle im Quellcode?

B:Genau, wo man also im Quellcode Anpassungen machen muss, damit das gelöst wird. Das ist tatsächlich für mich als Anfängerin und neu in der Programmierung, war das die größte Herausforderung. Ich merke, ich werde jetzt besser damit, alleine diese Stellen zu finden im Code. Aber das war die größte Herausforderung am Anfang. Und das hat tatsächlich nur geklappt durch Unterstützung von Erfahreneren Mitarbeitern. […]

I:Und bevor du überhaupt zu diesem Issue kamst. Ich weiß nicht, ob dir das zugeteilt wurde?

B:Ja genau.

I:Okay, also bevor dir das zugeteilt wurde, was waren die ersten Informationen? Hast du dir die GUI angeschaut und dich durchgeklickt?

B: Ja, ja klar. Genau richtig. Das allererste war bestimmt, mindestens zwei Wochen rein sich die Oberfläche anschauen und die Oberfläche verstehen, das ist klar. Also, wenn man jetzt eine Software hat, die eine GUI hat, dann denke ich ist das immer der Einstige, dass man sich erst mal mit der GUI vertraut macht.

I: Alles klar, und danach bekommst du dieses Issue zugeteilt, hast die passende Stelle im Code mit der Unterstützung von anderen gefunden. Bist du so vorgegangen, dass du dann mehrere Issues gelöst hast?

B: Genau. Im Kleinen angefangen und so versucht ein Bild aufzubauen.

I: Wenn man sich jetzt doch kurz an die IslandViz erinnert, da werden ja auch Hauptkomponenten angezeigt. Ist sowas etwas, das man sich anschaut? Hast du da einen Überblick drüber?

B: Ja gut, die Komponenten, die es in RCE gibt, die haben ein bestimmte Packagestruktur. Die wird einem relativ schnell klar. Zumindest von der Komponentenseite, wie das halt aufgebaut ist. Da gibt es einmal die GUI-Implementierung. Das ist dann ein eigenes Package. Dann gibt es Execution, was passieren soll, wenn diese Komponente ausgeführt wird, und dann gibt es Common Package, was so eine Zwischenstelle zwischen den beiden ist, also Code, den beide Packages verwenden. So ist erst mal unsere grobe Struktur. Und dann gibt es noch einen Testordner, für irgendwelche Tests, okay. Das ist diese Struktur, die wird bei uns auch im Developer Guide beschrieben. Also die hat man sich auch vorher vielleiht schon mal angelesen. Das kann man dann in Eclipse schön sehen. Dann gibt es natürlich außer den Komponenten noch andere Bereiche. Alles, was im Hintergrund läuft. Da war ich jetzt bisher selbst noch nicht so unterwegs. Weil wir haben natürlich auch im Team Schwerpunkte. Die Seite, die eher so GUI implementiert. Die Seite die eher Backend programmiert. Die haben dann halt so ihre anderen Packages. Da ist mir die Struktur jetzt auch noch nicht so richtig vertraut, weil ich damit jetzt auch noch nicht so arg in Berührung gekommen bin. Man lernt das immer dann, wenn man dort hinkommt. So, wie war nochmal die Ausgangsfrage, ich habe vielleicht etwas den Faden verloren.

I: Davor hatte ich gefragt, ob man diese Hauptkomponenten zum Überblick braucht.

B: Ja, die sind halt von der Packagestruktur klar zu erkennen.

I: Wie kann ich mir die Arbeit mit dem Developer Guide vorstellen? Hast du dir das am Anfang einmal durchgelesen oder passiert das immer wieder?

B: Doch, doch, da schaue ich schon ab und zu nochmal rein. Also erstmal gehts darum, wie man jetzt Eclipse richtig aufsetzt. Den Checkstyle, keine Ahnung, was wir da alles für zusätzliche Plugins noch haben. Und da gucke ich immer mal wieder noch rein, wenn man da in der Situation ist, dass man einen neuen Rechner bekommt zum Beispiel. Dann muss man das alles wieder neu machen und hat natürlich viel wieder vergessen. Aber ansonsten, während der alltäglichen Arbeit braucht man den dann eigentlich nicht mehr.

I: […] Was würdest du denn sagen, ist der Zielzustand? Du hast […] dein erstes größeres Feature, ist das der Endpunkt der Einarbeitung?

B: Ich glaube tatsächlich, irgendwie wird man immer noch was dazu lernen. Das hört jetzt auch nicht auf, glaube ich. Zeiten, wie lange man jetzt braucht für die Einarbeitung hängt natürlich auch vom Vorwissen und so ab. Aber ich stelle auch immer mal fest, wenn man ältere Kollegen fragt, die schon seit Jahren im Team sind, die kennen jetzt auch nicht jede Codestelle. Dann sagen die halt auch, okay an der Baustelle war ich noch nie dran. Ich kann da jetzt kurz gucken aber das ist ja so ein komplexer Code, man wird da ja jetzt nicht jemals jede Stelle gesehen haben. Von daher ob die Einarbeitung jetzt wirklich irgendwann aufhört. Das kann ich nicht sagen. Also bei mir persönlich, hat sie definitiv noch nicht aufgehört.[…] Aber ich fühle mich jetzt sicherer, mich im Code zurechtzufinden und eigenständiger die Aufgaben zu bearbeiten, ohne ständig Kollegen befragen zu müssen. […] Da habe ich mindestens ein halbes Jahr für gebraucht.

I: Vorhin hattest du angesprochen, wie die Packages benannt sind. Hast du da mittlerweile auch ein Gefühl dafür, wo das Naming herkommt?

B: Ja. Also nicht für jede Klasse, das ist ja auch echt individuell. Man muss auch wissen, RCE ist echt schwieriger Code, weil da so viele verschiedene Leute dran gearbeitet haben in den letzten Jahren. Das Team wechselt ständig und es gibt keine einheitlichen Richtlinien an was für Namenskonventionen man sich zu halten hat. Das heißt, das macht jeder so ein bisschen nach seinem gut Dünken. Es gibt natürlich so eine grobe Namensgebungsstruktur, die sich vielleicht am Anfang auch mal Leute überlegt haben. Die habe ich jetzt auch noch nie verschriftlicht gesehen, muss ich zugeben, aber man sieht dann natürlich auch, okay, so und so hat jetzt jemand dann den Namen vergeben, und dann versucht man das ähnlich zu machen. Aber wie gesagt, da da so viele Leute dran waren, ist das in keinster Weise einheitlich. Und das macht diesen Code auch nochmal zusätzlich schwer für den Einstige.

I: Kann man das im Code erkennen, wer das geschrieben hat? Gibt es so einen persönlichen Fußabdruck?

B: Wir haben bei jeder Klasse oben einen Header drin, wo man den Autor hinzufügt. Da ist dann eine Liste von Autoren und wenn man etwas ändert, dann fügt man sich halt hinten dran. Da gibt es dann natürlich Klassen, da stehen oben fünf, sechs Autoren. Von denen sind dann vier auch schon gar nicht mehr hier. Die kann man dann auch nicht mehr direkt fragen. Und man weiß halt auch nie, wie viel die zu der Klasse beigetragen haben, also das weiß man nicht.

I: Auch nicht, was genau?

B: Man könnte das natürlich im Submission Tracking. Da könnte man theoretisch nachverfolgen, wer hat was submitted. Aber ich meine, wer macht das jetzt. Etwas, das vor drei Jahren submitted wurde, da klemmt man sich ja jetzt nicht mehr dahinter und sucht danach. Aber theoretisch, da könnte man das natürlich nachverfolgen, wer was geschrieben hat.

I: Und um dir einen Überblick zu verschaffen, wie alles aufgebaut ist, ist die Historie der Software da auch relevant? Wann sich wo etwas verändert hat?

B: Ich glaube Historie könnte ich mir jetzt eher so Richtung Nachvollziehbarkeit vorstellen. Ich konkret schaue da jetzt nicht danach, aber ich könnte mir vorstellen, dass es (...) gerade jetzt bei uns im Team, die Leute die sich um organisatorisches kümmern, dass die dann auch eher mal da rein gucken. Vor allem natürlich, wenn irgendwelche Bugs hochkommen. Da wird [ein Kollege] bestimmt schon mal hingehen und gucken, wer hat jetzt wo was gemacht, was hat jetzt dazu geführt dass jetzt plötzlich das System lahm gelegt ist, sag ich mal. Also diese Extremfälle. […]

I: Also für dich ist der Status quo relevant?

B: Genau, ich gucke vielleicht schon, wer ist jetzt der Autor gewesen um vielleicht auch mal jemanden Fragen zu können. Oder man sieht dann auch so, diese Komponenten sind alle von der und der Person geschrieben. Dann erkennt man natürlich auch, die bauen die Klassen immer gleich auf. Wenn man jetzt an einer Baustelle dran ist, die die eine Person viel bearbeitet hat, dann hat man da vielleicht Wiedererkennungswert, wie etwas implementiert worden ist. Zum Teil sind nämlich auch gleiche Aspekte anders implementiert worden. Eben aufgrund der Tatsache, dass das verschiedene Leute waren.

I: Insgesamt hast du ja jetzt sehr viel erzählt, dass du immer wieder andere Leute gefragt hast oder geguckt hast, wer das gemacht hat, um zu fragen. Also würdest du sagen, dass diese Einarbeitungsphase eine Teamarbeit ist?

B: Ja auf jeden Fall.

[…]

I: Kann man sich komplett im Team überhaupt einarbeiten?

B: Am Anfang schaut man sich ja erst mal die GUI an und das kann man schon gemeinsam machen. Hinterher, wenn es an den Code geht, dann ist man eher so ein Einzelkämpfer. Natürlich immer mit Leuten fragen zwischendurch, klar. Aber dann muss man halt auch selbst die Struktur verstehen. Ja, ich glaube das ist aber tatsächlich eine Typfrage, wie die eigenen Vorlieben sind. Ich glaube es hilft ja immer allgemein, wenn man viel selbst macht. Und sich selbst durch den Code navigiert, anstatt, dass man dann mit anderen zusammen, und andere navigieren dann und dann hat man es ja selbst nichts gelernt. Also ich bin immer so der Typ, ich mach das lieber selbst.

I: Dann ist eigentlich auch schon der letzte Teil des Interviews gekommen. Hier versuche ich nochmal zusammenzubringen, ob du denkst eine Softwarevisualisierung hätte dir bei der Einarbeitung geholfen. Du hattest ja gesagt, dass die GUI eigentlich der erste Teil ist und du dich damit eine Weile beschäftigst. Aber kannst du dir dann danach vorstellen, dir so eine Softwarevisualisierung anzuschauen bevor du dich an den Code setzt?

B: Also eine Visualisierung um die GUI zu verstehen, das fände ich nicht schlecht muss ich sagen. Auch, was die RCE GUI anbelangt, da sind mir auch noch gar nicht alle Dinge so unbedingt klar, immer noch nicht. Und auch, wenn man da die Kollegen fragt, die sagen auch, da gibt es manche Feature oder so, die sind seit Jahren nicht angepackt worden, die kennt dann auch keiner mehr. Also da eine Visualisierung, was bezweckt eigentlich die GUI, was gibt es für Einstellungsmöglichkeiten und wieso, weshalb, warum. […] Sowas fände ich schon sehr, sehr hilfreich, das könnte ich mir auch als Visualisierung durchaus vorstellen. Dass man sich durch die GUI navigieren kann. Weil auch die GUI jetzt anhand unseres User Guides, der ist auch teilweise überarbeitungswürdig. Das reicht auch nicht allein mit dem User Guide die GUI zu lernen und zu verstehen. Wir haben da zwar schon einen relativ guten User Guide, wir haben auch Examples und sowas. Aber das könnte ich mir gut vorstellen. Code-seitig finde ich es nach wie vor echt schwierig.

I: Und da nochmal zur Softwarearchitektur. Würdest du sagen, du verstehst die Softwarearchitektur auch ohne Visualisierung oder ist es eher so, dass du denkst du musst die Softwarearchitektur auch einfach nicht auf diesem Level verstehen?

B: Also ich würde nicht behaupten, dass ich die schon verstehe. Das glaube ich nicht, das wäre vermessen. Aber ich würde eher zweiteres sagen, was du gerade genannt hast. Ich brauche es im Moment nicht so. Und ja, ich habe da bei der IslandViz jetzt nicht so den Mehrwert für mich gesehen.

I: Bei der IslandViz war ja die komplette Software dargestellt und die Zusammenhänge. Wenn man sich vorstellt, dass du an einer Stelle im Code sitzt und dir dann eine Visualisierung sagen könnte, wie hängt jetzt diese spezifische Stelle mit anderen Dingen zusammen? Wenn man jetzt weg geht von dieser großen Übersicht, könnten kleinere Ausschnitte helfen?

B: Also ich will das auch alles ja nicht so komplett ab tun, weil ich dafür einfach glaube ich auch noch zu wenig Erfahrung habe. Mir fällt das dann jetzt etwas schwer. Ich möchte das nicht komplett schlecht reden oder so. Ich habe einfach so einen Fall einfach noch nie gehabt. Denkbar ist das schon, dass das wichtig sein könnte. Wenn man so ein Package hat dann zu wissen, wie sind jetzt die Dependencies. […]

I: Aber wie die Dependencies von den Packages sind, das könntest du auch über den Code rausfinden?

B: Ja.

I: Aber für dich, wie relevant ist es, das zu wissen?

B: Für mich jetzt aktuell noch gar nicht so relevant. Also das was ich jetzt mache. Aber ich weiß jetzt auch nicht, ob ich schon das richtige Verständnis von Dependencies habe, ja? Aber es ist ja irgendwie, wenn ich da halt ein Package benötige, das nicht zu Verfügung steht, dann gibt es in Eclipse ja „organize inputs“ dann holt der dir automatisch den Import, den du brauchst. Und du bekommst direkt den Hinweis das und das ist private, das kannst du an dieser Stelle nicht benutzen. Dann switcht du da hin im Code und machst das eben public, wenn du der Ansicht bist, du brauchst das jetzt. Der Code bietet dir da relativ viele Möglichkeiten. Der unterstützt einen da komplett dabei, sodass ich diese Struktur eigentlich gar nicht brauche. Wenn ich das richtige Verständnis davon habe, meine ich, dass ich das eigentlich nicht brauche.

I: Gerade hast du noch davon erzählt, dass an Videos gearbeitet wird für die GUI. Videos laufen dann automatisch durch, vielleicht so was man anklickt und was dann passiert. Kannst du dir vorstellen, dass es Vorteile gibt, wenn man das selbst machen kann?

B: Ja, ja auf jeden Fall. Vor allem GUI-seitig kann man da glaube ich viel machen. Aber ja, dass man auch so interagieren kann und man dann Erklärungen bekommt, wenn du jetzt diesen Button klickts, dann passiert das und das. Und dass man das dann irgendwie ausführen kann oder halt auch nicht. Und dann erhält man verschiedene Ergebnisse. Wenn man sich da eben selbst durchklicken kann, das wäre cool.

I: Eine Frage noch zur Darstellung. Einfach mal angenommen es würde dann doch eine neue Visualisierung über Strukturen oder ähnliches geben. Findest du diese Metaphern, wie bei der IslandViz oder Städtemetaphern oder Planeten oder Landschaften ähnliches hilfreich? (zeigt Bilder). Oder es gäbe auch eher so abstraktere Varianten.

B: Also ich fände diese Bilder viel zu abstrakt (a). Also tatsächlich finde ich so eine Metapher eigentlich gut. Das hilft einem schon. Also man muss natürlich einmal erklärt bekommen, wofür stehen jetzt die Inseln, die Häfen, die Städte und so weiter. Aber dann ist das besser zu verstehen als das hier zum Beispiel (c). Gut, da wird es bestimmt auch eine Erklärung dafür geben und mit unterschiedlichen Farben und so. Aber das würde mir so jetzt wenig sagen. Also eigentlich finde ich die Herangehensweise über so eine Metapher nicht schlecht.

I: Und rein vom Medium; es gibt Ansätze der Visualisierung am PC aber auch in AR und VR. Wenn du dir das frei aussuchen könntest, hättest du da eine Präferenz?

B: Also wenn das eine gute Visualisierung ist, die einem was bringt, dann würde ich mir natürlich auch VR oder AR gut vorstellen können.

I: Super, das waren eigentlich alle meine Fragen. Hast du noch Fragen oder Anmerkungen?

B: Also ich hoffe, dass ich dir weiterhelfen konnte, sonst fällt mir jetzt selbst gerade nichts mehr ein. Wenn du noch Fragen hast, dann kannst du einfach auch nochmal auf mich zukommen.

B: Ja super, dann vielen lieben Dank!

# Eidesstattliche Erklärung

Hiermit erkläre ich, Jana-Sophie Effert, dass ich die unter der Betreuung von Prof. Dr. Sebastian Pannasch vorliegende Arbeit selbstständig verfasst und keine anderen als die angegebenen Hilfsmittel verwendet habe. Alle angeführten Zitate habe ich kenntlich gemacht.

Ort, Datum Unterschrift