Praktikumsbericht
Visualisierung von Software-Evolutionsgraphen

Cynthia Rapp, 557972
Rochusstraße 351
50827 Köln

01.03. - 30.06.2019
80 Arbeitstage

Deutsches Zentrum für Luft- und Raumfahrt
Linder Höhe
52247 Köln
www.dlr.de

Hochschule für Technik und Wirtschaft
Willhelminenhofstraße 75A
12459 Berlin
www.htw-berlin.de

Zusammenfassung

Im Rahmen des Ingenieurinformatik-Studiums an der *Hochschule für Technik und Wirtschaft (HTW)* in Berlin ist ein Pflichtpraktikum von mindestens zwölf Wochen abzuleisten. Dieses habe ich nun vom 01.03. - 30.06.2019 in der Einrichtung *Simulations- und Softwaretechnik* beim *Deutschen Zentrum für Luft- und Raumfahrt (DLR)* in Köln absolviert. Ich war in der Abteilung *Intelligente und verteilte Systeme* tätig, in der ich an einer Visualisierung von Software-Evolutionsgraphen gearbeitet habe. Das Ergebnis ist eine webbasierte Anwendung, mit der sich Softwarearchitektur in ihrer Evolution visualisieren lässt. Diese Ausarbeitung enthält alle Schritte, die mich zum Ergebnis geführt haben, in Form eines Praktikumsberichts.

Tabellenverzeichnis

Abbildungsverzeichnis

Inhaltsverzeichnis

1 Einleitung

In diesem Kapitel werden die Arbeitsumgebung, das Projektziel und die Rahmenbedingungen meiner Arbeit erläutert, um den Kontext des Projekts deutlich zu machen. Außerdem wird ein Einblick in die Thematik geliefert, damit die Hintergründe und die Wichtigkeit der Arbeit in diesem Bereich klarer werden.

1.1 Arbeitsumgebung

Im Folgenden werden einige Informationen und Fakten über die Arbeitsumgebung dargelegt, in dem das Unternehmen und anschließend die Abteilung ausführlich beschrieben werden. Es wird, neben den Zahlen und Fakten des eingetragenen Vereins, auch auf die verschiedenen Geschäftsfelder, die Finanzierung und Einstiegsmöglichkeiten näher eingegangen. Im Anschluss wird auf die Schwerpunkte, die Mitarbeiter und den Aufbau der Abteilung in Köln, in der ich tätig war, eingegangen.

1.1.1 Deutsches Zentrum für Luft- und Raumfahrt

[1] Der Deutsche Zentrum für Luft- und Raumfahrt e.V. fungiert nicht nur als nationale Raumfahrtagentur, sondern auch als Projektträger und Forschungseinrichtung. An 20 Standorten mit insgesamt 40 Instituten und Einrichtungen wird in den Bereichen Luft- und Raumfahrt, Energie, Verkehr, Sicherheit und Digitalisierung geforscht und entwickelt. 2017 zählte das DLR 8.125 Mitarbeiter, davon waren 4.723 Wissenschaftliche und die übrigen Nicht-Wissenschaftliche, die gemeinsam an den weltweit 178 Forschungsanlagen arbeiten.

Die Themengebiete, mit denen sich die Mitarbeiter hier beschäftigen, reichen von allen naturwissenschaftlichen und ingenieurtechnischen Disziplinen über Medizin bis hin zu Kommunikation und Informationstechnik. Im Bereich der Luft- und Raumfahrtforschung stehen Effizienz, Sicherheit und Umweltbewusstsein im Vordergrund, deshalb wird vermehrt auf Digitalisierung, Automation und neue Kommunikations- und Navigationstechnologien gesetzt. Das DLR besitzt außerdem die größte Forschungsflugzeugflotte Europas. Die Interessen und Ziele des Verkehrs decken sich mit denen der Luft- und Raumfahrtforschung, um die Mobilität auch am Boden zu verbessern und zu digitalisieren. Im Bereich der Energie möchte das DLR bei der Umstellung der Energiesysteme auf klimafreundliche und risikoarme Energiequellen helfen.

Durch den Auftrag der Bundesregierung werden alle nationalen und internationalen Aufgaben des Raumfahrtmanagements vom DLR durchgeführt,

etwa ein eigenes Raumfahrtprogramm und seinen Beitrag für die europäische Weltraumagentur *ESA (European Space Agency)*. Als einer der größten Projektträger Deutschlands unterstützt das DLR überwiegend öffentliche Auftraggeber wie die Bundesregierung und die europäische Kommission mit Analysen und Beratungen und in der Forschung.

Die Gesamtfinanzierung des Vereins setzt sich zu beinahe gleichen Teilen aus Drittmitteln und institutionellen Förderungen zusammen. Die Forschung wird zu 90 Prozent aus staatlichen Mitteln und zu 10 Prozent von den Ländern finanziert. Zusätzliche Erträge werden durch die Arbeit als Projektträger und durch das Raumfahrtmanagement erzielt. Das meiste Geld wird in die Raumfahrtforschung investiert, und gleich darauf in die der Luftfahrt.

Das DLR bietet neben seinen Stellenausschreibungen auf der Webseite auch duale Bachelorstudiengänge, Praktika und Abschlussarbeiten für einen frühen Berufseinstieg an. Zusätzlich gibt es an vielen Standorten das *School Lab*, in dem interessierte Schüler die Möglichkeit haben verschiedenste Experimente rund um Wissenschaft und Technik durchzuführen.

1.1.2 Abteilung Intelligente und verteilte Systeme

[2] Eine wichtige Einrichtung ist die Simulations- und Softwaretechnik, die sich mit der Entwicklung und Bereitstellung von innovativen Softwaretechnologien beschäftigt, um diese intern und extern zu vertreiben. Die Mitarbeiter befassen sich mit Projekten und Themen, die im gesamten DLR und auch von außen angefragt werden.

Die Einrichtung lässt sich in drei Abteilungen unterteilen: Intelligente und verteilte Systeme (IVS), *High-Performance Computing* und *Software für Raumfahrtsysteme und interaktive Visualisierung*.

[3] Die Abteilung IVS beschäftigt sich mit der Forschung an Softwaresystemen und betreibt Informationsvisualisierung, Software-Engineering und -Analytik. Die Mitarbeiter der Arbeitsgruppe *Software Engineering* sitzen an den Standorten Köln, Braunschweig, Oberpfaffenhofen und Berlin. Viele dieser Mitarbeiter arbeiten an *Remote Component Environment (RCE)*, welches multidisziplinären Teams die Zusammenarbeit erleichtern soll. Die verteilte Software wird als Integrationsumgebung in Projekten verwendet und somit stetig erweitert. Die Forschungsgebiete dieser Abteilung reichen von Software Repository Mining, über Machine Lerning, bis hin zu Visual Computing, wozu auch die Visualisierung von Software und Graphen und die visuelle Datenanalyse gehören.

1.2 Einführung in die Thematik

Im Laufe der Entwicklung einer Software, die teilweise jahrelang dauert und ein großes Team von Entwicklern fordert, findet eine Evolution in der Architektur statt, die letztendlich nicht mehr der ursprünglich geplanten entspricht. Das liegt vor allem daran, dass sie ständig weiterentwickelt wird.

Zu Beginn eines Auftrags sollte ausführliche Planung und Recherche stattfinden. An dieser Stelle wird auch die Softwarearchitektur mit Hilfe von UML-Diagrammen geplant. Das reicht aber häufig nicht mehr aus, und sobald die Software eine Veränderung in ihrem Lebenszyklus durchmacht, sind diese Diagramme nicht mehr aktuell und man müsste neue erstellen, um den aktuellen Stand korrekt und verständlich für alle darstellen zu können. Es hat nur Vorteile für das Projekt, wenn der Entwickler zu jedem Zeitpunkt die Architektur des Ganzen im Blick behalten kann, an der das gesamte Team arbeitet und die mit dem Umfang der Software auch an Komplexität gewinnt. Auch neue Entwickler, die inmitten des Entwicklungsprozesses einsteigen, können so den aktuellen Stand, den Verlauf der Entwicklung und die Softwarearchitektur, an der sie arbeiten werden, erfassen.

Hinzu kommt hier, dass die Software auf einem reichhaltigeren Komponenten-Framework aufbaut, der *Open Source Gateway initiative (OSGi)*. Es dient zur Erweiterung der Modularisierung von Code, die bei *Java* für besonders große Projekte zu schwach ist. OSGi ist eine Spezifikation von Java, um eine komponentenorientierte Entwicklung zu unterstützen und Services abzurufen und bereitzustellen. Das Framework unterteilt die Java-Applikationen in Module und verwaltet diese und deren Abhängigkeiten über den gesamten Lebenszyklus.

Die Wichtigkeit der Visualisierung im Bereich Software und ihrer Architektur hat mit ihrer Komplexität und ihrem Umfang zugenommen. In diesem Bereich gibt es schon einige Möglichkeiten; was dabei aber oft außer Acht gelassen wird, ist genau der Aspekt der Evolution.

Die Vorteile, die ein Tool zur Abbildung der Evolution mit sich bringen würde, sind nicht nur Übersichtlichkeit und die Nachvollziehbarkeit jedes Zeitpunkts der Architektur, die dem gesamten Team während der Entwicklung hilft. Auch die Möglichkeiten im Anschluss des Projekts können eine Rolle spielen. Aus jedem Projekt nimmt man neues Wissen und Methoden für das nächste mit; genauso aus jedem Release für den nächsten. Man könnte statistische Auswertungen durchführen; der Teamleiter könnte sofort sehen, an welcher Stelle es Probleme gab, welche Komponente am aufwändigsten war und wie komplex, und inwiefern sich die Ist- von der Soll-Architektur verändert hat. Aus diesem Grund bedarf es in diesem Bereich noch an Entwicklung.

1.3 Ziel der Arbeit

Ich hatte mich auf ein konkretes Projekt beworben und anschließend wurden entsprechend meiner Fähigkeiten und meines Vorwissens der Umfang und die Rahmenbedingungen festgelegt. Das Thema war die Visualisierung der Evolution von Softwarearchitektur. Mein Ziel war die Entwicklung einer interaktiven Webanwendung, in der die Softwarearchitektur zu allen Zeitpunkten ihrer Evolution in 2D visualisiert wird.

Der verwendete Datensatz stammt aus den `Commits` des *Gits* eines DLR-internen Projekts und wurde in Java basierend auf OSGi entwickelt. Die Daten werden aus der Graphdatenbank *Neo4j* gewonnen und dann auf einer Webseite dargestellt. Es handelt sich dabei um einen strukturierten Graphen, bestehend aus Knoten und Kanten, welche ich auch als solche darstellen wollte, anstatt eine Metapher zu verwenden. Die Zielgruppe bestand folglich in erster Linie aus den Entwicklern und Neueinsteigern der dargestellten Software und den Projektleitern, die sich ebenfalls mit der Materie auskennen.

Es sollten einige Interaktionsmöglichkeiten eingebunden werden, etwa ein Filter, Selektion und Navigation. Geplant war eine Suchleiste mit automatischer Vervollständigung, einfache und eindeutige Navigation innerhalb eines Commits, sowie zwischen den Commits, und verschiedene Optionen zum Filtern in den einzelnen Ansichten. Des Weiteren sollte eine Art Zeitraffer entstehen, der jederzeit abgespielt und pausiert werden kann wie ein Video: Es spielt einmal alle Commits einer Ansicht in zeitlicher Reihenfolge ab, um den direkten Vergleich von Anfang bis Ende mit den entsprechenden Veränderungen in der Evolution angezeigt zu bekommen.

Da die Architektur nicht in einer einzigen Ansicht dargestellt werden kann, sollte in mehrere Ansichten unterteilt werden, welche entsprechend des Modells von Java und OSGi gewählt werden. Die Startansicht würde alle `Bundles` des letzten bzw. aktuellsten Commits darstellen, ein Ansichtswechsel soll dann die `Packages`, `Classes` oder `Services` des angewählten Commits zeigen. Und bei einem Commitwechsel soll die gewählte Ansicht bestehen bleiben, nur der Zeitpunkt ändert sich. Über eine Zeitachse soll die zeitliche Veränderung der Softwarearchitektur deutlich sichtbar verändert dargestellt werden können, beispielsweise durch Ausgrauen oder Hervorheben von Elementen, die sich mit der Zeit verändert haben. Es soll zusätzlich die Möglichkeit geben, in einer kombinierten Ansicht nur bestimmte selektierte Bereiche in ihrer Entwicklung anzusehen. Das Wichtigste im Bezug auf die Darstellung der Evolution waren Eindeutigkeit und Übersichtlichkeit, die ich durch eine Positionsübergabe der Knoten beim Commitwechsel gewährleisten sollte.

2 Übersicht der Tätigkeiten

Folgende Tabelle enthält die ausgeführten Praktikumstätigkeiten stichwortartig zusammengefasst in der Reihenfolge ihrer Ausführung.

Tabelle 1: Wochenweise Übersicht der Tätigkeiten

No.	Woche	Tätigkeiten
01	01.03.-10.03.2019	• Dienst-Laptop und Arbeitsplatz einrichten • Dienstreise nach Berlin für die Einarbeitung • Datensatz erhalten und in Neo4j einbinden • Recherche zu Neo4j und Webanwendungen • Xampp installieren und localhost einrichten
02	11.03.-17.03.2019	• HTML lernen • Cypher für Neo4j-Abfragen lernen • Recherche zu Visualisierungstools
03	18.03.-24.03.2019	• Javascript lernen • NeoVis.js für die Visualisierung testen • D3.js für die Visualisierung testen
04	25.03.-31.03.2019	• Daten aus Neo4j extrahieren mit dem Neo4j-Javascript-Driver
05	01.04.-07.04.2019	• Erste Visualisierungsversuche mit Vis.js
06	08.04.-14.04.2019	• Visualisieren der Bundles eines Commits • Zeitstrahl implementieren und integrieren
07	15.04.-21.04.2019	• Herstellen einer Verbindung zwischen Zeitstrahl und Cypher-Abfragen • Erstellen des Webseitenlayouts • Einbauen einer Commitwechsel-Funktion • Abfragen und Visualisierung der Services
08	22.04.-28.04.2019	• Ansichtenwechsel und Commitübergabe • Veränderung im Commitwechsel feststellen
09	29.04.-05.05.2019	• Abfragen und Visualisierung der Packages • Abfragen und Visualisierung der Classes
10	06.05.-12.05.2019	• Legende für Knoten erstellen • Veränderungen im Commitwechsel darstellen
11	13.05.-19.05.2019	• Autocomplete Searchbar implementieren • Optimieren der Anzeige für Selektiertes • Benutzerdefinierte Ansicht hinzufügen
12	20.05.-26.05.2019	• Datenbank-Login bauen • Zeitraffer implementieren

13	27.05.-02.06.2019	• yFiles for HTML lernen und einbinden
14	03.06.-09.06.2019	• Austauschen der Bibliothek für die Graph-Visualisierung durch yFiles
15	10.06.-16.06.2019	• Austauschen der Bibliothek durch yFiles
16	17.06.-23.06.2019	• Graph-Visualisierung an yFiles anpassen und optimieren
17	24.06.-30.06.2019	• Graph-Visualisierung anpassen • Letztes Fixing und Optimierung

Wie sich der Tabelle entnehmen lässt, habe ich mich zuerst etwa vier Wochen mit der Vorbereitung, bestehend aus Recherche, Einlesen, dem Aneignen von Grundkenntnissen in neuen Sprachen und dem Testen möglicher Bibliotheken für die Umsetzung beschäftigt. In dieser Zeit habe ich den Großteil der Spezifikation und der Entscheidungen für die Umsetzung des Projekts getroffen. Aber auch noch im Laufe der Implementierung habe ich mich häufiger entscheiden oder vorherige Entscheidungen ändern müssen.

Nachdem ich die Vorbereitung abgeschlossen hatte, konnte ich mit der Extraktion der Daten beginnen. Nachdem ich dann die erste Visualisierung eines Graphen von Bundles eines Commits erreicht hatte, beschäftigte ich mich mit dem Zeitstrahl. Anschließend begann ich mit der Webseitengestaltung, dem Aufbau der Webseite und deren Elementen, um eine funktionierende Seite mit ersten Visualisierungsergebnissen vorzeigen zu können.

Ich fuhr mit der wichtigsten Aufgabe fort, der Visualisierung der Evolution, indem ich die Veränderungen im Commitwechsel zuerst feststellte und anschließend darstellte. Wenn ich nicht weiter kam, schob ich leichtere Funktionen dazwischen, etwa eine Legende der Knoten oder die Visualisierung weiterer Ansichten.

Etwa 6 Wochen vor Praktikumsende, als ich meine bisherigen Ergebnisse dem Abteilungsleiter präsentierte, legte er mir nahe, noch einmal eine andere Bibliothek zu versuchen, da die von mir gewählte für den Zeitraffer nicht optimal war. Weshalb ich an diesem Punkt begann, mein Programm umzuschreiben und die neue Bibliothek, die er mir empfohlen hatte, einzubinden.

In den letzten 2-3 Wochen beschäftigte ich mich noch intensiv mit Anpassungen, Optimierungen und einigen Fehlerbehebungen, wie dem noch nicht richtig funktionierenden Login oder dem Zeitraffer.

3 Beschreibung der Tätigkeiten

In folgenden kategorisch zusammengefassten Kapiteln werden die einzelnen Tätigkeiten des vorangegangenen Kapitels näher erläutert und ausführlicher beschrieben. Diese lassen sich in zwei Bereiche unterteilen: die Vorbereitung, die alle notwendigen Schritte bis zur Umsetzung enthält, und die Implementierung, in der die tatsächliche Umsetzung des Projekts beschrieben wird.

3.1 Vorbereitung

Mein erster Arbeitstag bestand hauptsächlich darin, zusammen mit dem IT-Manager meinen Arbeitsplatz und Dienst-Laptop einzurichten und mir die verwendeten Programme anzueignen, wie *Outlook* als Email-Programm und Kalender, und das dazugehörige *Lync* als internes Kommunikationstool, das später zu *Skype for Business* wechselte. Da sowohl der Abteilungsleiter als auch meine interne Praktikumsbetreuerin zu dem Zeitpunkt nicht in Köln waren, flog ich an meinem zweiten Arbeitstag nach Berlin, wo mich die beiden einarbeiteten, in die DLR-interne Politik, Hierarchien, Regularien und alles weitere Wissenswerte für neue Mitarbeiter.

Anschließend sollte ich noch den Datensatz, auf dem meine Visualisierung aufbauen sollte, von meiner Betreuerin erhalten. Sie hatte den Datensatz bereits bei sich in Neo4j integriert und wollte ihn nun an mich weitergeben, damit ich mit der Arbeit beginnen konnte. Zuerst versuchten wir verschiedene Datentypen wie `.graphml`, `.cypher`, `.db` oder `.dump`, aber keiner davon wollte sich bei mir einbinden lassen; sie waren entweder unvollständig oder gar nicht vorhanden. Auch über das Terminal im Neo4j-Desktop oder durch Veränderung der `Settings` ließ sich der Graph weiterhin nicht ansprechen, bis die Graph-Datenbank abstürzte und nicht mehr starten wollte. Nach einem Neustart des Rechners und der Reinstallierung des Programms schickte sie mir den Datensatz in ein paar Überordnern als `.zip`, und ich ersetzte diesen Ordner bei mir durch ihren. Da ich nun den Datensatz und ein funktionierendes Neo4j hatte, konnte ich mit meiner Arbeit beginnen.

Um mein Projektziel zu erreichen, brauchte ich vor der Umsetzung ein Konzept. Dieses konnte ich nur mit ausreichender Recherche und durch Einlesen in die Thematik entwickeln. Danach konnte ich auch erst wissen, was für Tools oder Programme ich brauchte und welche Sprachen ich noch erlernen musste, um eine Webseite aufzubauen. Ohne jegliche Vorkenntnisse begann ich mit umfangreicher Vorbereitung, die in den folgenden Kapiteln näher erläutert und beschrieben wird.

3.1.1 Recherche und Einlesen

Ich habe mich als erstes ein bisschen mit dem Datensatz und dem Neo4j-Desktop und -Browser auseinandergesetzt. Neo4j verwendet die eigene `Graph Query Language` *Cypher*, mit der man Abfragen und Bearbeitungen am Datensatz vornehmen kann. Ich schaute mir erst einmal die im Browser eingebetteten Lerneinheiten „Learn about Neo4j“ und „Jump into code“ an, die einen Grundkurs in Graphen und Graphdatenbanken sowie in Cypher boten.

Ich habe mir den Datensatz mit seinen 60945 Knoten und 115512 Kanten eine Weile angeschaut, indem ich diese nach `Node Label` oder `Relationship Type` sortieren und anzeigen ließ; dadurch habe ich auch einen Eindruck der Struktur bekommen.

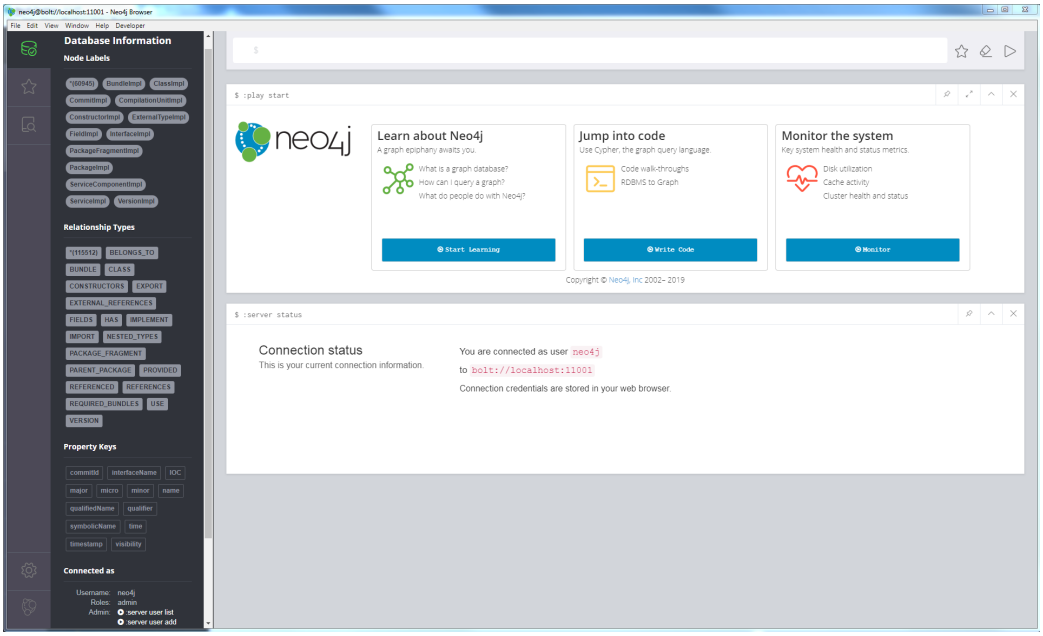

Abbildung 1: Neo4j-Browser

Abbildung 3 zeigt den Neo4j-Browser direkt nach dem Öffnen am Graphen. Links sind wichtige Informationen zum aktuellen Datensatz aufgelistet, wie die Nodel Label, die Relationship Types und die `Property Keys` die im Graphen vorkommen, zusammen mit der totalen Anzahl an `Nodes` und `Relationships`. Darunter befindet sich der Benutzer, der gerade verbunden ist. Diese Information kann man auch im `connection status` ablesen, genauso wie die Serveradresse, mit der der User verbunden ist. In der Mitte befinden sich die bereits erwähnten Lerneinheiten und ein konkretes Beispiel, und darüber der Editor, in dem man seine Abfragen oder Befehle in Cypher

formulieren kann. In dem Menü am linken Seitenrand befinden sich unter dem Graph noch die Favoriten oder gespeicherten Skripte, Dokumentation zu Neo4j und Cypher, die Browser Einstellungen und das Impressum.

Nachdem ich mich damit beschäftigt hatte, habe ich versucht das Datenbankschema des Datensatzes anhand der Knoten und Kanten des Graphen zu rekonstruieren, damit ich eine Grundlage hätte, auf die ich zurückgreifen konnte. Bald fand ich heraus, dass man sich das Schema einfach über den Cypher-Befehl `call db.schema()` anzeigen lassen konnte, wie in folgender Abbildung.

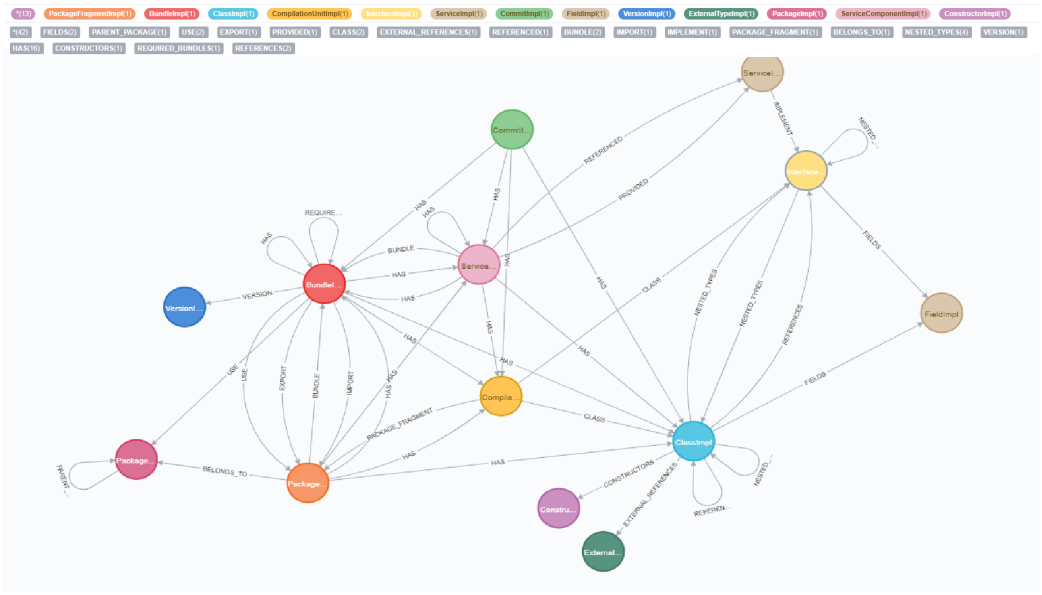

Abbildung 2: Datenbankschema aus dem Neo4j-Browser

Zum Einlesen in die Thematik verwies mich meine Betreuerin auf zwei Master-, eine Bachelorarbeit und ein Paper, die alle von Mitarbeitern aus der Abteilung verfasst worden waren.

Als erstes las ich die Bachelorarbeit von Marlene Brüggemann, mit dem Titel „Visualisierung von mit OSGi-Komponenten realisierten Softwarearchitekturen im 3-dimensionalem Raum mit Virtual Reality“[4], in der es vor allem um das OSGi-Konzept, die Arten der Datenvisualisierung, und die visuellen Variablen und Metaphern ging. Sie hat ein Konzept für eine eindeutige Darstellung von Softwarearchitektur in einer Virtual Reality Umgebung mit Benutzerinteraktion und -navigation entwickelt. Die Umsetzung hat sie ausführlich dokumentiert und mit Aufnahmen der Umgebung ergänzt. Durch ihre Arbeit habe ich einen Eindruck von OSGi bekommen, womit ich bisher noch nicht gearbeitet hatte, und auch von der Wichtigkeit von Visualisierung

im Bereich von Software und ihrer Architektur.

Die Masterarbeit von Tobias Marquardt hingegen hat mir auch hinsichtlich der Entwicklung meines eigenen Konzepts weitergeholfen. Sie trägt den Titel „Extraktion und Visualisierung von Beziehungen und Abhängigkeiten zwischen Komponenten großer Softwareprojekte“[5] und beschäftigt sich unter anderem mit den Stufen der Datenvisualisierung und den verschiedenen Ansichten für eine strukturierte Übersicht. Anders als in meinem Fall, musste zuerst noch die Datenerfassung erfolgen, dafür listet er in seinem Konzept einige Datenquellen auf, die er nutzen konnte, um Informationen für seine Visualisierung zu erhalten. Die Historie ist zusätzlich zu Bundles, Services und Classes, eine weitere Ansicht, doch genau diese konnte aus zeitlichen Gründen nicht umgesetzt werden. Die Unterteilung in diese Ansichten habe ich ähnlich übernommen, nur dass ich die Historie in jede der anderen integriert habe. Als Tool für die Umsetzung wählte er *D3.js*, weshalb ich mir dieses auch später näher anschaute. Außerdem habe ich mich von seinem Aufbau der Webseite, von den Bereichen der Navigation, Kontextinformation und Konfigurationen und von seinen Filter- und Selektionsoptionen inspirieren lassen.

Eine weitere Masterarbeit, die ich zu diesem Thema zur Inspiration und Informationsbeschaffung gelesen habe, war die meiner Betreuerin Lynn von Kurnatowski mit dem Titel „Visualisierung der Evolution von Softwarearchitektur“[6]. Die nötigen Grundlagen zur Thematik werden hier erklärt, ein Konzept erstellt und die Umsetzung ausführlich beschrieben, was mir bei meiner Arbeit eine große Hilfe war. Sie erwähnte einige neue Dinge im Zusammenhang mit der Visualisierung von Softwarearchitektur, wie Graphdatenbanken und die Evolution der Architektur. Allerdings hat sie mir von den Tools oder Bibliotheken, die sie verwendet hatte, abgeraten, vor allem wegen des hohen Kostenfaktors. Die vorgeschlagenen Interaktionsmethoden habe ich weitgehend übernommen, wie Filterung, Navigation und Selektion.

Das Paper „Visualizing Modules and Dependencies of OSGi-based Applications“[7] beschäftigt sich mit den entwickelten interaktiven Visualisierungstools, die OSGi-basierende Programme mit ihren Modulen und Abhängigkeiten visualisieren können. Dabei gehen die Verfasser wieder auf OSGi ein, auf die Optionen in Bezug auf Dimension und Kontext, und auf die Arten der Darstellung der Daten. Dieses Paper gibt einen guten Überblick, was bisher in diesem Bereich geschehen ist, was für Möglichkeiten es gibt und was in Zukunft noch passieren könnte.

Anhand der Vorarbeiten, Ideen, Ansätze und Konzepte der gelesenen Arbeiten habe ich mich bei der Recherche leiten lassen. Ich habe mir die verschiedenen Tools und Möglichkeiten angeschaut, die in den Arbeiten aufgelistet und beschrieben waren, um zu sehen, welche sich auch für meine Anwendung eignen könnten.

3.1.2 Sprachen lernen und Testen

Ich wusste, dass ich als erstes einen Webserver brauchte, um alles Weitere darauf aufzubauen und andere Tools zu integrieren. Allerdings kannte ich mich mit den Möglichkeiten nicht aus, weshalb ich meine Betreuerin fragte. Ich begann dann mit *XAMPP*, das Ganze aufzusetzen. Nachdem ich herausgefunden hatte, wie und wo man welche Art von Dateien erstellen und ablegen musste, um eine Webseite aufzubauen, musste ich noch herausfinden, mit welchen Sprachen man nach welcher Struktur arbeitete.

Die erste neue Sprache, mit der ich mich ausführlich beschäftigte, war *Hypertext Markup Language (HTML)*, da diese die Grundlage für die Entwicklung einer Webseite darstellt. Durch Einführungs-Tutorials und explizite Suchen konnte ich eine erste Webseite aufbauen und mit den Elementen spielen. Dann fand ich heraus, dass sich alles Designtechnische, dass ich zuvor im `head` integriert hatte in einer extra Datei in *Cascading Style Sheets (CSS)* speichern und verändern ließ. Um jegliche Logik oder Funktionen zu integrieren, schrieb ich *Javascript*, vorerst im HTML-`body`, später dann in einer separaten Datei. Ich arbeitete an einer einfachen Webseite mit einigen Elementen und Funktionen, um die Sprachen etwas zu lernen und Ergebnisse zu sehen, bis ich zufrieden war. Ich hatte nur mit einem einfachen Editor gearbeitet, bis mir dann ein Arbeitskollege *Webstorm* von *Jetbrains* empfahl, was ich mir als Studentin kostenlos herunterladen konnte. Allerdings musste ich mich da auch erst einmal einarbeiten, aber letztendlich hat es mir die Arbeit deutlich erleichtert und für mehr Effizienz gesorgt.

Mit Cypher musste ich mich beschäftigen, da ich nur damit auf den Datensatz in der Datenbank zugreifen konnte. Die Abfrage-Sprache ähnelte *SQL* vom Aufbau, sollte aber einfacher zu lesen und verstehen sein. Da das offizielle Manual von Neo4j etwa 85 Webseiten zum Durchklicken hatte, beschränkte ich mich auf die wesentlichen Kapitel und auf das, was ich auch wirklich benötigen würde. Im Neo4j-Browser konnte ich Abfragen schreiben oder generieren lassen und mich so durch den Datensatz arbeiten und testen. Der Aufbau der von mir verwendeten Cypher-Abfrage für die Extrahierung der Daten lautet:

```
MATCH (n:NodeLabel) –[r:RelationshipType]-> () WHERE Bedingung RETURN n,r
```

`MATCH` sucht nach den Knoten und Kanten, die diese Node Label und/oder diese Relationship Types enthalten. In die runden Klammern kann dann auch dasselbe zweimal, zwei verschiedene oder kein Node Label eingetragen werden. In die eckigen Klammern kommt ein Relationship Type oder auch nicht. Mindestens eine der drei Klammern muss aber etwas Vorhandenes enthalten, damit die Abfrage ein Ergebnis liefert. Das `WHERE` ist eine optionale

Ergänzung, um die Abfrage zu spezifizieren oder näher einzugrenzen. Ein `RETURN` muss wiederum in der Abfrage stehen, damit Cypher weiß, was es ausgeben soll, die Kanten, die Knoten oder beides. Dahinter könnte man optional weitere Veränderungen in der Ausgabe hinzufügen, wie die Sortierfunktion `ORDER BY` oder die Eingrenzung um eine bestimmte Anzahl mit `LIMIT`. Die Ausgabe im Browser ist meist die Darstellung eines Graphen, kann aber auch tabellarisch erfolgen. Während der Abfrageformulierung überprüft der Browser, ob diese Sinn macht, und warnt schon vor der Ausführung über mögliche Fehler. Das und die generell leichte Handhabung mit dem Neo4j-Browser hat die Übung mit der Sprache deutlich erleichtert.

Nachdem ich mich mit den Sprachen auseinandergesetzt hatte, um in jeder das Grundwissen für den Aufbau einer Webseite und den Umgang mit dem Graphen in Neo4j zu besitzen, fing ich an, die Tools zu testen, die in Frage kamen. Ich hatte mir die Bibliothek D3.js bereits notiert gehabt aus der Masterarbeit von Tobias, wollte aber erst einmal selbst recherchieren, ob es vielleicht inzwischen etwas Neues oder Besseres gab.

Tatsächlich stieß ich bei meiner Suche auf eine Javascript Bibliothek, die genau für mein Vorhaben konzipiert worden war. *NeoVis.js* konnte die Daten aus Neo4j extrahieren und diese auch direkt visualisieren. Es gab allerdings noch keine ausführliche Dokumentation zu dieser Bibliothek, weshalb ich mich an dem einen Beispiel des Herausgebers und einem dazu passenden `How-To`-Video orientieren musste, um sie einzubinden. Ich testete das Ganze auf meiner zuvor erstellten Webseite, aber das Ergebnis stellte mich nicht zufrieden. Ich konnte die Optionen den Graphen betreffend nicht verändern. Letztendlich habe ich in der Bibliothek nach den `Default`-Optionen gesucht und diese dort verändert. Allerdings wusste ich nicht, welche Optionen es überhaupt gab, was das Anpassen des Graphen an meine Vorstellungen schwierig gestaltete. Ich konnte beispielsweise nicht herausfinden, wie ich die Überschneidung der Kanten vermeiden konnte.

Da mir der Umgang mit NeoVis nicht gut gefallen hatte, entschied ich mich D3.js zu testen, um zu sehen, ob das besser funktionieren würde. Die Bibliothek war für Visualisierungen jeglicher Art besonders gut geeignet, allerdings musste ich erst einmal Daten haben, die ich damit darstellen konnte. Ich versuchte ein Beispiel zu finden, in dem jemand die Daten extrahiert und dann dargestellt hatte, aber ohne Erfolg. Dann versuchte ich die Beispiele, die ich im Internet zu Graph-Visualisierungen mit D3.js fand, nachzuvollziehen. Ich übertrug eines der Beispiele auf meinen `Code` mit Dummy-Daten, aber ich bekam keine Visualisierung heraus, weshalb ich mich nicht lange mit der Bibliothek beschäftige und aufgab. Aber ich traf die Entscheidung, Extrahierung und Visualisierung voneinander zu trennen und separate Tools bzw. Bibliotheken zu verwenden, so wie ich es auch mit D3 hätte tun müssen.

Da ich aber bei einer separaten Suche keinen Erfolg hatte, ging ich nochmal zurück zu der spärlichen Dokumentation, die ich von Neovis.js hatte, das sich aus dem *Neo4j-javascript-driver* und *Vis.js* zusammensetzte. Auf dieser Grundlage suchte ich erneut und fand sowohl ausführliche Dokumentation zu der Bibliothek als auch einige Codebeispiele zum Treiber. Ich schaute mir die Beispiele an und hatte schnell eine Übersicht vom Umfang der Arbeit, die mir übersichtlich vorkam. Die Dokumentation von Vis.js, war sehr ausführlich und deckte alle Funktionen, Demos und Beispielcode ab. Aufgrund der guten Dokumentation, der vielen Möglichkeiten und der einfachen Handhabung wählte ich für die Umsetzung des Projekts schließlich diese beiden Tools.

Während der Entwicklung stellte ich allerdings fest, dass die Eigenschaft des automatischen Renderns ein großes Problem in der Übersicht darstellte. Das Rendern folgt einem Algorithmus, der die Knoten und Kanten automatisch in die optimale Position bringt nachdem die Visualisierung erstellt und ausgeführt wird und das ließ sich nicht abschalten. Für die Ansicht der Bundles war das nicht besonders auffällig gewesen, da diese aus wenigen Knoten und Kanten bestand. Bei den anderen Ansichten wurde dieses Manko dann aber aufgrund der hohen Kantenanzahl umso deutlicher. Aus diesem Grund musste ich mir später noch eine andere Bibliothek anschauen.

Ich lud mir die kostenlose Evaluationsversion von *yFiles for HTML* herunter, die eine 60 tägige Lizenz enthielt. Das Paket liefert außerdem ausführliche Dokumentation, wozu ein „Developer's Guide“, Demos und Beispielcode gehören. Das Einbinden der Bibliothek war komplizierter als bei den bisherigen, da ich sonst einfach nur die Javascript-Datei direkt in den Ordner kopiert hatte, was aber nun nicht möglich war und mir deshalb viele Probleme verursachte, die ich aber mit etwas Hilfe lösen konnte.

3.2 Implementierung

Die Implementierung wurde zur besseren Planbarkeit in mehrere Arbeitspakete unterteilt und wird in folgenden Unterkapiteln weiter ausgeführt. Grob konnte ich in die Extrahierung der Daten und die Visualisierung einteilen. Die Visualisierung bestand dann wiederum aus zwei Teilen, den Graphen und dem Zeitstrahl. Die Gestaltung der Webseite bezüglich Aufbau und Aussehen habe ich zur besseren Übersicht in einem separaten Kapitel behandelt. Das letzte Unterkapitel behandelt die Logik, die hinter der gesamten Anwendung steckt.

3.2.1 Webseitengestaltung

Das erste was der Nutzer sieht, ist der Aufbau der Webseite. Bevor er irgendwelche Funktionen testet, ist eine Meinung gebildet. Ich wusste, dass die Aufteilung des Bildschirms in definierte Abschnitte sowohl für die Übersicht als auch die Handhabung eine wichtige Rolle spielen würde. Deshalb wollte ich diese Entscheidung so weit nach hinten schieben, bis ich auch etwas zu integrieren hatte und Funktionen vorzeigen konnte, um das Layout gleich testen zu können.

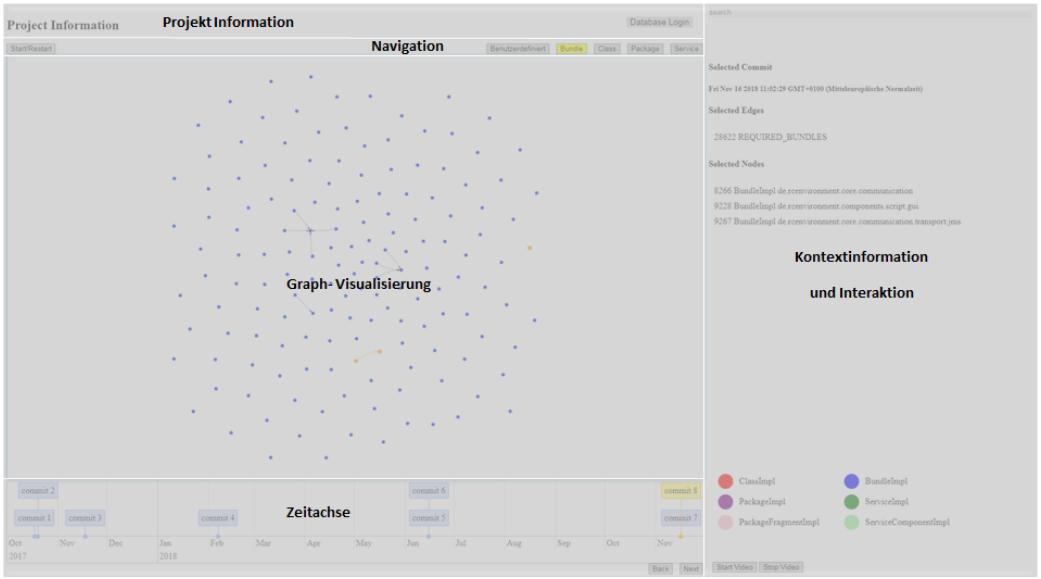

Abbildung 3: Aufteilung der Webseite in verschiedene Bereiche

Die Aufteilung in vier Abschnitte, ähnlich des Layouts der beiden Masterarbeiten, schien mir am sinnvollsten. Es gäbe einen Navigations-, einen Konfigurations-, einen Visualisierungs- und einen Informationsbereich. Das

habe ich im Groben so umgesetzt, allerdings musste ich die Anordnung der Bereiche im Dienste der Übersichtlichkeit etwas anpassen.

Wie in Abbildung 5 zu sehen ist, besteht die Webseite nun aus fünf Bereichen. Einer davon ist die Projektinformation, die sich am oberen Rand der Seite befindet. Dieser Bereich beinhaltet Informationen zum aktuellen Projekt und den Datenbank-Login. Darunter befindet sich die Navigation, bestehend aus dem Start-Button, mit dem sich die Visualisierung starten oder neu erstellen lässt und fünf weiteren Buttons, mit denen sich die Ansicht der Beschriftung entsprechend wechseln lässt. Direkt darunter befindet sich der Bereich der Visualisierung des Graphen und die Zeitachse. Unter dem Zeitstrahl habe ich noch zwei Buttons hinzugefügt, mit denen man vom aktuellen zum vorherigen oder nächsten Commit springen kann, wenn es einen gibt. Daneben, im Kontextinformations- und Interaktionsbereich, werden Informationen zum aktuellen Commit sowie zu den angewählten Kanten oder Knoten angezeigt, und dort wären auch die Interaktionsmöglichkeiten angesiedelt. Eine Suchleiste befindet sich am oberen Rand und eine Legende der Knoten am unteren. Unter der Legende befinden sich dann noch die Buttons zum Starten und Stoppen des Zeitraffers.

3.2.2 Extrahierung der Daten

Für die Extrahierung der Daten aus der Graphdatenbank wurde der Neo4j-Javascript-Driver verwendet. Diese Bibliothek ermöglicht es, mithilfe eines konkreten Cypher-Befehls die entsprechenden Daten aus der Datenbank abzufragen und das Resultat dann weiter zu verarbeiten.

Dafür musste der Treiber als erstes initialisiert werden, indem man ihm die IP-Adresse `localhost` zusammen mit dem `bolt`-Port aus Neo4j und eine Authorisierungsvariante übergibt. Die vorgeschlagene Variante ist die `basic authorization`, bei der man Benutzername und Passwort aus Neo4j übergibt. Den bolt-Port, den die Graphdatenbank aktuell nutzt, kann man unter dem Graphen einsehen, wenn er aktiv ist, d.h. den Status `RUNNING` anzeigt. Der Benutzername und das Passwort sind standardgemäß `neo4j`, man kann diese aber auch ändern oder neue im Browser hinzufügen, muss dann aber auch angeben, dass der neue Benutzer ein Administrator ist, sonst kann keine Verbindung aufgebaut werden. Man kann diese Benutzerdaten auch unter `connection status` einsehen, genau wie den aktuellen bolt-Port. Ich habe dem Treiber noch zusätzlich die Option `encryption:false` übergeben, um Protokollprobleme im Browser zu vermeiden.

Nachdem der Treiber initialisiert wurde, konnte ich eine Sitzung an ihm öffnen. Wichtig ist, dass man sowohl die Sitzung als auch den Treiber am Ende wieder schließt, um mögliche Aufhänger oder Fehler zu vermeiden. Mit

der vordefinierten Funktion `run()`, konnte ich eine Cypher-Abfrage direkt an Neo4j senden und das Resultat mit einer `forEach`-Schleife auslesen. Manche Cypher-Befehle im Code ergaben nicht die gleichen Ergebnisse wie im Neo4j-Browser. Diese musste ich dann solange umschreiben und testen, bis ich das gewünschte Resultat hatte.

Das `hardcoden` der Benutzerdaten war nur als eine vorübergehende Lösung zum Testen gedacht, weshalb ich später einen Datenbank-Login integrierte. Dieser fragt genau die Daten ab, die ich an den Treiber im Code übergeben hatte und erstellt diesen dann bei Bestätigung. Erst bei korrekter Eingabe würde sich die Seite vervollständigen, sodass man mit ihr interagieren kann. HTML bietet eine `form`, die sich abrufen lässt und verschiedene `input types` enthält. Für die Datenbank habe ich den type `url` gewählt, da dieser auf das Muster der Eingabe der Datenbank achtet und einen Hinweis gibt, wenn diese dem nicht entspricht. Beim Benutzernamen wählte ich `text` und beim Passwort `password`, welches bei der Eingabe automatisch durch Sternchen ersetzt, um vor fremder Einsicht zu schützen. Das Problem, welches ich nicht lösen konnte, war das Neuladen der Seite durch Eingabe eines inputs. Wenn man die Eingaben durch einen Button bestätigt hat, ruft dieser eine Funktion auf, die die Eingaben abspeichert, den Treiber damit erstellt und die Session initialisiert. Anschließend wird die erste Funktion `getcommits()` aufgerufen, welche ich beim hardcoden mit dem Aufruf der Seite aufrufen ließ. Nachdem aber die Eingaben bestätigt werden, die Form sich schließt und die Funktion aufgerufen wird, wird die Seite neu geladen, mit den unverschlüsselten input-Daten in der Browser-URL. Es würde wenig Sinn machen, die privaten Benutzerdaten verdeckt einzugeben, um sie danach in der URL ablesen zu können, aber ich fand nicht heraus, wie man das hätte verhindern oder ändern können. Außerdem wurde durch das Neuladen der Seite der Funktionsaufruf wieder rückgängig gemacht. Man sah solange es lud noch die Veränderung durch die Funktion, aber sobald es abgeschlossen war, war die Webseite wieder wie beim Seitenaufruf. Auch durch Hinzufügen eines Events wurde dieses Problem leider nicht gelöst. Der Login ist noch im Code vorhanden, allerdings auskommentiert und noch unbrauchbar, solange das nicht gefixt wird.

Die Abfragen für die Bundles und deren Beziehungen waren die einfachsten. Für die Knoten eines Commits konnte ich alle Bundles mit dem selben `timestamp` abfragen, für die Kanten musste ich den Relationship Type mitangeben, um die korrekten Beziehungen zu erhalten. Bezieht man das dann beispielsweise auf die Service-Ansicht, musste ich jede Art der Beziehung separat abfragen, um alle vollständig zu erhalten. Auch die Knoten habe ich entsprechend ihres Node Labels einzeln abfragen müssen.

Die einzelnen Commits, aus denen ich die timestamps erhalte, habe ich zu

Beginn einmal abgefragt und abgespeichert. Alle anderen Abfragen werden jedes Mal neu ausgeführt, wenn ein anderer Commit oder eine andere Ansicht gewählt wird. `CommitImpl` ist ein Node Label, und kann so nach dem gleichen Prinzip wie die anderen Nodes abgefragt werden.

```
session
    .run('MATCH (n:CommitImpl) RETURN n ORDER BY n.time')
    .then(function (result) {
        result.records.forEach( callbackfn: function (record) {
            id = record._fields["0"].identity.low;
            content = record._fields["0"].properties.timestamp;
            time = record._fields["0"].properties.time;
            date = datetodate(time);
            timestamp = getstamp(date);
            number = number + 1;
            zeitliste.add({id: timestamp,content: "commit " + number, start: date, title: content, label: id});
        });
        session.close();
    })
    .catch( onrejected: function (error) {
        console.log(error);
    });
```

Abbildung 4: Codeausschnitt - Extrahierung und Verarbeitung der Commits

Der vorangegangene Codeausschnitt in Abbildung 6 zeigt die Cypher-Abfrage, mit der die Commits als Knoten abgefragt und das Resultat weiter verarbeitet wird. Bei der Abfrage wird direkt in zeitlicher Reihenfolge sortiert, sodass die Nummer des Commits mit einem einfachen `Counter` hochgezählt werden kann. Aus dem Resultat der Abfrage lassen sich einige Eigenschaften entnehmen, wie die `ID`, der Name und der Zeitpunkt des Commits. Allerdings befinden sich zwei Zeitstempel unterschiedlicher Art in jedem Commit, die *Unix time* mal 1000, und die Zeit nach dem Format `Fri Nov 16 11:02:29 CET 2018`. Die Unix Time gibt die Sekunden an, die seit dem 01.01.1970 ab 00:00:00 Uhr vergangen sind. Dieser timestamp ist auch ein Property Key jedes anderen Knotens. Wie ich die beiden Zeitangaben weiterhin genutzt habe, wird in Kapitel 3.2.4 zum Zeitstrahl näher erläutert.

Welche Property Keys in den Visualisierungsdatensatz übergeben werden hängt davon ab, welchen Knoten man wählt. Die, die sich bei den verschiedenen Knoten unterscheiden, sind die Namen. Die Bundles haben beispielsweise einen `name` und zusätzlich noch einen `symbolicName`, während die Klassen stattdessen einen `qualifiedName` haben. Die Services haben zum Beispiel nur einen `interfaceName` und die Commits gar keinen Namen. Man musste also vorher wissen, welches Node Label man abfragt, um die richtigen Property Keys zu kennen, weshalb die meisten Funktionen ziemlich statisch und verbindlich geworden sind. Die Kanten waren eindeutiger, da man bei der Abfrage zwar den Relationship Type kennen musste, aber keine Property Keys existieren und Anfangs- und Endknoten vorher bekannt sind.

3.2.3 Visualisierung - Graph

Für die Visualisierung der Nodes und Edges der einzelnen Graphen habe ich die Bibliothek yFiles for HTML verwendet, welche eine schrittweise Anleitung für diesen Zweck bietet. Das Paket beinhaltet sogar eine Demo mit Code für eine Visualisierung mit Neo4j, die ähnlich dem Neo4j-Browser aufgebaut wurde.

Die Visualisierung eines Graphen muss ausgelöst werden. Entweder durch den Start-Button, der die Startansicht zeigt, oder durch Anwählen einer Ansicht oder eines Commits. Wählt man also als erstes eine Ansicht aus, wird der letzte Commit in dieser Ansicht angezeigt, und wählt man als erstes einen Commit aus, wird dieser in der Bundle-Ansicht dargestellt. Zu jedem Zeitpunkt gilt: ändert man die Ansicht, bleibt der Commit, und andersherum. Abbildung 7 zeigt einen Screenshot der Visualisierung in der Startansicht (die Bundles des letzten bzw. neuesten Commits).

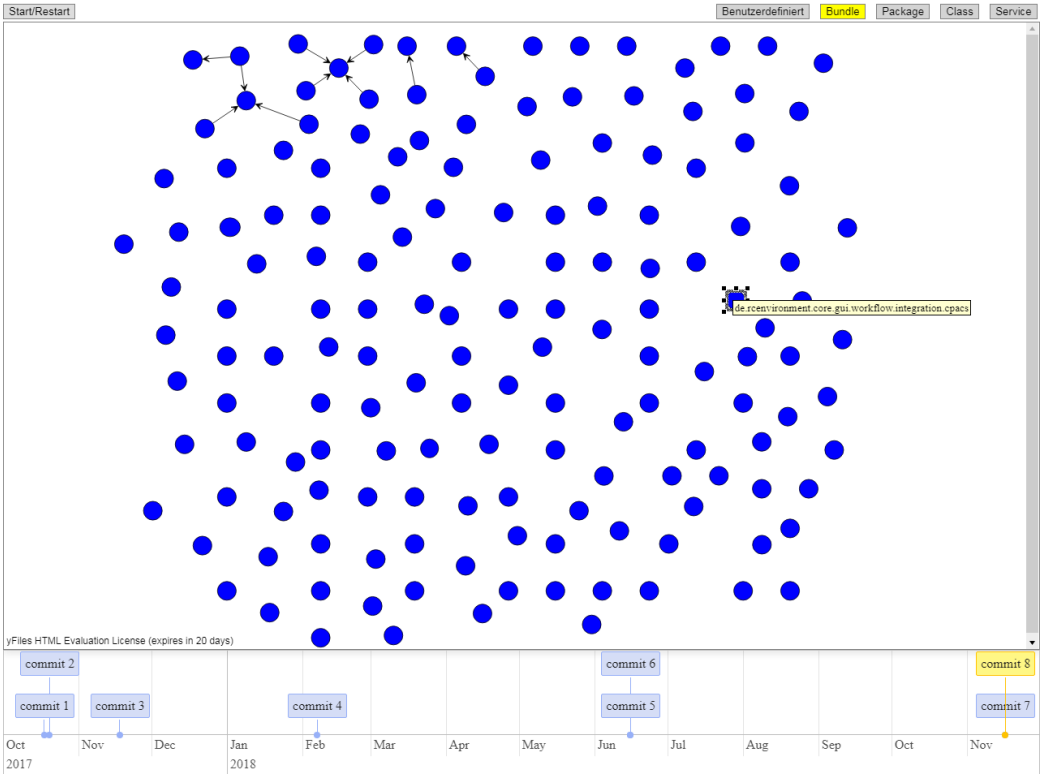

Abbildung 5: Screenshot der Startansicht

Um die Bibliothek ordnungsgemäß einzubinden und verwenden zu können, musste ich alle benötigten vordefinierten Komponenten aus yFiles importieren, unter anderem die Lizenz. Sie enthält Werte wie einen `Key` und ein Aus-

laufdatum und muss in den Code eingebunden werden, damit das Programm auch funktioniert. Als erster Schritt muss der Bereich der Visualisierung festgelegt werden, indem man den Namen des HTML-`tags` übergibt und darin den Graphen erstellt. YFiles bietet die Möglichkeit, ein Array als Datensatz zu übergeben und automatisch mit vordefinierten Layout-Parametern anzuordnen. Der `GraphBuilder` ist für das Erstellen des Graphen zuständig. Ihm werden die Datensätze übergeben und mit `buildGraph()` wird der Graph ausgegeben. Im Anschluss habe ich noch das Layout festgelegt und so anzeigen lassen, dass alle Knoten auf einmal im Bereich zu sehen sind. In diesem Fall wurde das `Organic` Layout mit einer Orientierung von oben nach unten ausgewählt, da das optisch am besten aussah.

Die einzelnen Ansichten sind so übersichtlich wie möglich gehalten, weshalb aufgrund der teilweise sehr hohen Anzahl an Nodes entschieden werden musste, welche hier jeweils dargestellt werden. Nicht alle im Datenmodell enthaltenen Arten von Knoten sind relevant für die Visualisierung. Würde man versuchen, sie alle zu integrieren, bräuchte man entweder mehr Ansichten oder weniger Knoten, um die Übersichtlichkeit und Benutzerfreundlichkeit weiterhin zu gewährleisten. Deshalb enthält die Bundle-Ansicht auch nur alle Bundles und die Klassenansicht nur Nodes mit dem Label `ClassImpl`. Hier hatte ich zwischenzeitlich überlegt, diese optional um ihre möglichen Konstruktoren und Interfaces erweitern zu können. Entweder über eine Checkbox, durch die sich visuell ein- oder ausblenden ließe, oder durch schriftliche Auflistung im Bereich der Kontextinformation. So hätte ich dann auch die `Compilation Unit` zur Package-Ansicht hinzufügen können. Diese enthält nun also alle Packages und `Package Fragments` des gewählten Commits. Die Services und `Service Components` werden dann in der vierten Ansicht dargestellt. Alle Daten dieser Ansichten werden auf die gleiche Weise extrahiert und dargestellt.

Im Laufe der Entwicklung kam eine benutzerdefinierte Ansicht zu den eingeplanten hinzu. Diese zeigt nur ausgewählte, relevante Bereiche in ihrer Evolution, die der Nutzer vorher auswählt. Wenn der Nutzer in einer der anderen Ansichten einen oder mehrere Nodes selektiert hat und anschließend in die benutzerdefinierte wechselt, werden nur diese dort angezeigt. Wenn man nun durch die Commits klickt, werden auch nur die angezeigten Commits in ihrer möglichen Veränderung gezeigt. Geplant war, hier eine Ausklapp-Option für über- oder untergeordnete Elemente einzubauen, was zeitlich aber leider nicht umsetzbar war. Selektiert man Knoten, werden diese abgespeichert, und wechselt man dann in die benutzerdefinierte Ansicht, werden diese übergeben. Nach einer Visualisierung werden die Daten als aktuelle Daten abgespeichert, auch in dieser Ansicht. Und wechselt man nun den Commit, werden die Informationen aus den aktuellen Daten gewonnen statt aus den

selektierten. Deshalb muss, obwohl man die Daten bereits hat, nochmals aus Neo4j extrahiert werden. So erhält man in jedem Fall die korrekte Visualisierung der Knoten.

Durch den Wechsel der Bibliothek zu yFiles haben sich einige Funktionen geändert und es musste umstrukturiert werden. Gegen Ende war dann nicht mehr genug Zeit, um das Multi-Selektieren funktionierend zu implementieren, weshalb sich zwar mehrere Nodes mithilfe der Steuerung-Taste anwählen lassen, aber nur einer abgespeichert und übergeben werden kann. Der Code der benutzerdefinierten Ansicht kann eine Liste auslesen, wenn eine übergeben werden würde. Man müsste somit nur das entsprechende Event einbauen und hätte dann wieder den gleichen Stand wie mit der vorherigen Bibliothek.

Im vergleich zu Vis.js war die Handhabung mit yFiles anfangs eher aufwändiger, da die meisten Funktionen nicht automatisch integriert und verknüpft sind, sondern separat implementiert werden müssen. Wenn man sich allerdings erst einmal eingelesen und die Bibliothek verstanden hat, ist die Arbeit damit aber einfacher und die Ergebnisse sind schneller und schöner.

Beispielsweise hatten die Elemente im Graphen keine integrierte Selektier-Funktion oder Tooltip. Beides musste man mit einem Event hinzufügen und richtig konfigurieren, um diese Funktionen gewährleisten zu können. Auch die Übergabe der Eigenschaften hat sich sehr unterschieden zur vorher genutzten Bibliothek. Man konnte nun so viele Parameter übergeben wie man wollte und diese dann auch jederzeit abfragen. Zuvor hatte ich beispielsweise die Art des Knoten als `group` übergeben, um davon die Farben der Knoten abhängig zu machen. Oder ich übergab einen `title`, was automatisch den Tooltip darstellte. Im Gegensatz dazu konnte ich einige Zeilen Code einsparen, wegen der direkten Übergabe der Daten. Zuvor musste ich diese immer wieder aus verschiedenen Datensätzen entnehmen und vergleichen, um das zu erhalten, was ich brauchte.

3.2.4 Visualisierung - Zeitstrahl

Während der Arbeit mit der Vis.js fand ich heraus, dass die Bibliothek neben Graphen auch eine Art Zeitstrahl erstellen konnte. Mit `vis.timeline()` ließ sich ähnlich der Visualisierung eines Graphen eine einfache Zeitachse darstellen.

Es werden ein `Container` bzw. Visualisierungsbereich, Daten und Optionen benötigt, um die `timeline` zu erstellen. In den Optionen muss ein Start- und Enddatum definiert werden, damit überhaupt initialisiert wird. Die Daten, die als Markierung auf der Achse eingezeichnet werden, müssen einem bestimmten Format entsprechen, damit sie auch als solche von der Bibliothek

erkannt werden. Dazu habe ich die Ergebnisse der Datenextrahierung der Commits in der forEach-Schleife mit der eigenen Funktion `datetodate(time)` konvertiert. Das richtige Format `2015-07-04T19:32:24` wurde aus der Unix time mal 1000 gewonnen, indem man sie mit Javascripts Funktion `Date()` konvertiert und dann jeden Wert einzeln entnimmt und korrekt zusammensetzt. Ich musste allerdings zusätzliche Abfragen einbauen, die im Falle einer einstelligen Zahl eine Null vorsetzte, da Vis.js diese sonst nicht interpretieren konnte. Das Ergebnis gab ich nur als Startzeit ohne eine Endzeit ein, da ich keine Zeitspannen darstellen wollte. Zusätzlich übergab ich als Daten an jeden Zeitpunkt den timestamp als ID, den Commit mit Nummer als Beschriftung, und die lesbare extrahierte Zeitangabe als `Title`. Der Title wird sichtbar, wenn man mit der Maus über das Feld geht. Abbildung 8 zeigt die visualisierte Zeitachse mit allen Commits, der letzte bzw. aktuellste Commit ist deutlich sichtbar angewählt, er entspricht dem Default-Commit.

Abbildung 6: Zeitachse mit allen 8 Commits des Datensatzes

Jeder Commit auf der Zeitachse hat ein `Select-Event` erhalten, das zum einen automatisch die Farbe der Markierung verändert, und zum anderen auf der rechten Seite des Bildschirms das entsprechende Datum und die Uhrzeit anzeigt, wenn es ausgelöst wird. Anschließend wird die gewählte Ansicht abgefragt und anhand dieser, mit Übergabe des neuen Zeitpunktes, der entsprechende Graph visualisiert. Falls noch keine Ansicht gewählt wurde, wird standardgemäß die der Bundles gewählt.

Beim Klicken eines der beiden Buttons `Next` oder `Back` wechselt der Commit zum nächsten bzw. vorherigen in zeitlicher Reihenfolge. Die noch aktuellen Daten des Commits werden für die spätere Verarbeitung abgespeichert. Außerdem wird die Liste der sortierten timestamps geladen und die Position des aktuellen Commits ermittelt. Die Position wird dann um eins erhöht bzw. verringert, um so den neuen aktuellen Zeitstempel zu erhalten, mit dem dann die Visualisierung erneut erstellt wird. Gleichzeitig werden wie beim direkten Anwählen eines Commits die Farbe der Markierung an der neuen Position verändert und Datum und Uhrzeit angepasst.

3.2.5 Logik

Einen klaren Ablauf zu schaffen, sodass sich die Funktionen nicht in den Weg kommen würden, und die Reihenfolge, in der man die Buttons anklickt, keinen Einfluss darauf haben würde, war meine erste große Herausforderung. Am sinnvollsten schien es mir, zuerst den Commit und anschließend die Ansicht abzufragen, damit dieser dann direkt der entsprechenden Graph-Visualisierung übergeben werden kann. Als erstes wird immer der Zeitstempel abgefragt, wenn dieser aber noch nicht existiert, weil kein Commit angewählt wurde, wird die Funktion `startover()` aufgerufen. Diese Funktion wird auch durch Klicken des Buttons mit der Aufschrift Start/Re-start ausgelöst, wodurch der letzte timestamp ausgewählt wird. So oder so, wenn man den Zeitstempel erhalten hat, wird im Anschluss die Ansicht abgefragt. Dabei wird lediglich die Farbe der fünf Buttons, mit denen man die Ansicht wählen kann, abgefragt und entsprechende Visualisierung aufgerufen und gestartet. Wenn die Ansicht verändert wird, ändert sich zuerst die Farbe des Buttons, sodass deutlich erkennbar ist, welcher gewählt wurde, und anschließend wird wieder der Zeitstempel abgefragt und dann die Ansicht. Es gibt für alle Fälle diese feste Reihenfolge (siehe Abbildung 9), sodass sich keine Auswahl gegenseitig rückgängig machen kann oder etwas außer Acht gelassen wird, auch wenn das an manchen Stellen eine Doppelung verursacht.

Die nächste Herausforderung bestand darin, die Veränderung zwischen den Commits beim Vor- und Zurückspringen festzustellen und anschließend auch in der Darstellung deutlich zu machen. Sowohl beim Vor- als auch beim Zurückspringen wird als erstes die aktuelle Position aus einer Liste der Zeitstempel jedes Commits ermittelt und anschließend überprüft, ob es überhaupt einen vorigen bzw. nächsten Commit gibt. Gibt es einen, wird erst wieder die Ansicht überprüft und normal weiter verfahren, bis eine neue Visualisierung entstanden ist. Anschließend werden die abgespeicherten Daten des zuvor angesehenen und die des aktuellen Commits anhand ihrer Namen verglichen und so die Unterschiede abspeichert. Von den Namen, die dann die Unterschiede darstellen, wird dann wieder die ID abfragt und anhand dieser sollten die Knoten farblich markiert werden. Es muss aber noch herausgefunden werden, ob die Veränderungen im aktuellen oder im vorherigen Commit stattgefunden haben, und entsprechend entschieden werden, ob hervorgehoben oder ausgegraut wird. Es wird in beide Richtungen verglichen und die jeweils längere Liste gibt Auskunft darüber, welche Funktion aufgerufen wird. Im Falle einer Ausgrauung von Knoten, da sie nicht mehr vorhanden sind, würden die Elemente anhand ihrer ID aus der Datenbank hinzugezogen und das Ganze neu gezeichnet werden. Wenn Knoten hervorgehoben werden sollen, weil sie im Commit neu dazu gekommen sind, würden die jeweiligen

Elemente einfach umgefärbt werden, sodass sie sich eindeutig unterscheiden. Sollte es keinen zeitlich nächsten oder vorherigen Commit geben, wird ein entsprechender Hinweis am unteren Rand des Bildschirms ausgegeben.

Das Darstellen der Veränderung konnte nie in der Visualisierung getestet werden, da im Datensatz, den ich erhalten hatte, keine sichtbare stattgefunden hatte. Die Feststellung von Veränderung konnte ich durch Ausgabe der Daten - in der `console` - an Verschiedenen Stellen im Code testen, welche theoretisch die richtigen Ergebnisse lieferten. Die Funktionen zum hervorheben und ausgrauen der Knoten wurden für Vis.js implementiert, wodurch sie bei der Umstellung auf yFiles aber umgeschrieben werden mussten. Ich konnte leider die Farbe pro Node zeitlich nicht mehr anpassen, weshalb auch die Legende nicht mehr ganz gültig ist. Ich habe die Farbe pro Ansicht bestimmen können, aber nicht nach Eigenschaften, wie ich es zuvor umgesetzt hatte. Die Funktionen zum Darstellen der Veränderung sind also zu diesem Zeitpunkt nicht mehr vorhanden.

Um eine funktionierende automatische Vervollständigung für die Suchleiste zu implementieren, mussten dynamisch Felder entstehen und sich mit passenden Ergebnissen füllen. Dafür entnehme ich die Namen aller Knoten des aktuell angezeigten Graphen, die global gespeichert werden, und übergebe diese Liste an die Suchleiste. Sobald ein Buchstabe eingegeben wird, wird der erste Buchstabe jedes Eintrags der Liste überprüft und alle Übereinstimmenden klappen zur Auswahl auf. Bei der weiteren Eingabe wird immer weiter der nächste Buchstabe verglichen und somit die Auswahl immer entsprechend eingeschränkt. Wählt man nun einen der Einträge aus, schließt sich die Liste, das Suchfeld wird geleert und der gewählte Knoten sollte selektiert werden, was durch die Umstellung auf die neue Bibliothek leider auch erst einmal wegfiel.

Der Zeitraffer kam als letzte Funktion zum Projekt hinzu. Ich wollte es durch einen Start-Button zu jedem Zeitpunkt auslösen und von dem Commit dann beginnen lassen, weshalb als erstes die aktuelle Position `pos` erfasst werden musste. Ich versuchte eine Schleife mit Timer zu schreiben, die durch die Liste der Commits durchläuft und dabei jedes mal die Funktion `Nextcommit()` aufruft. Sie ist diese zwar durchgelaufen, hat aber nicht jeden Schritt visualisiert, da die sie erst durchlief und nach der angegebenen Zeit alles auf einmal ausgab, was bei der Schnelligkeit dann nur auf die Visualisierung des letzten Commits hinauslief. Ich habe das Problem dann mit einem Rekursiven Aufruf lösen können. Die Funktion `(Play(i))(pos)` enthält einen `SetTimeout()` mit 6 Sekunden Wartezeit bis die Next-Funktion ausgelöst wird. Vorher wird aber noch ohne Event abgefragt, ob der Stop-Button gerade betätigt wurde um den Zeitraffer auch jederzeit anhalten zu können. Wurde dann die Funktion aufgerufen und der nächste Graph visua-

lisiert wird geprüft ob der Zähler i noch immer kleiner ist als die Liste der timestamps. Ist dies der Fall wird die Funktion selbst erneut aufgerufen mit dem neuen hochgezählten i, solange bis der letzte Commit erreicht ist.

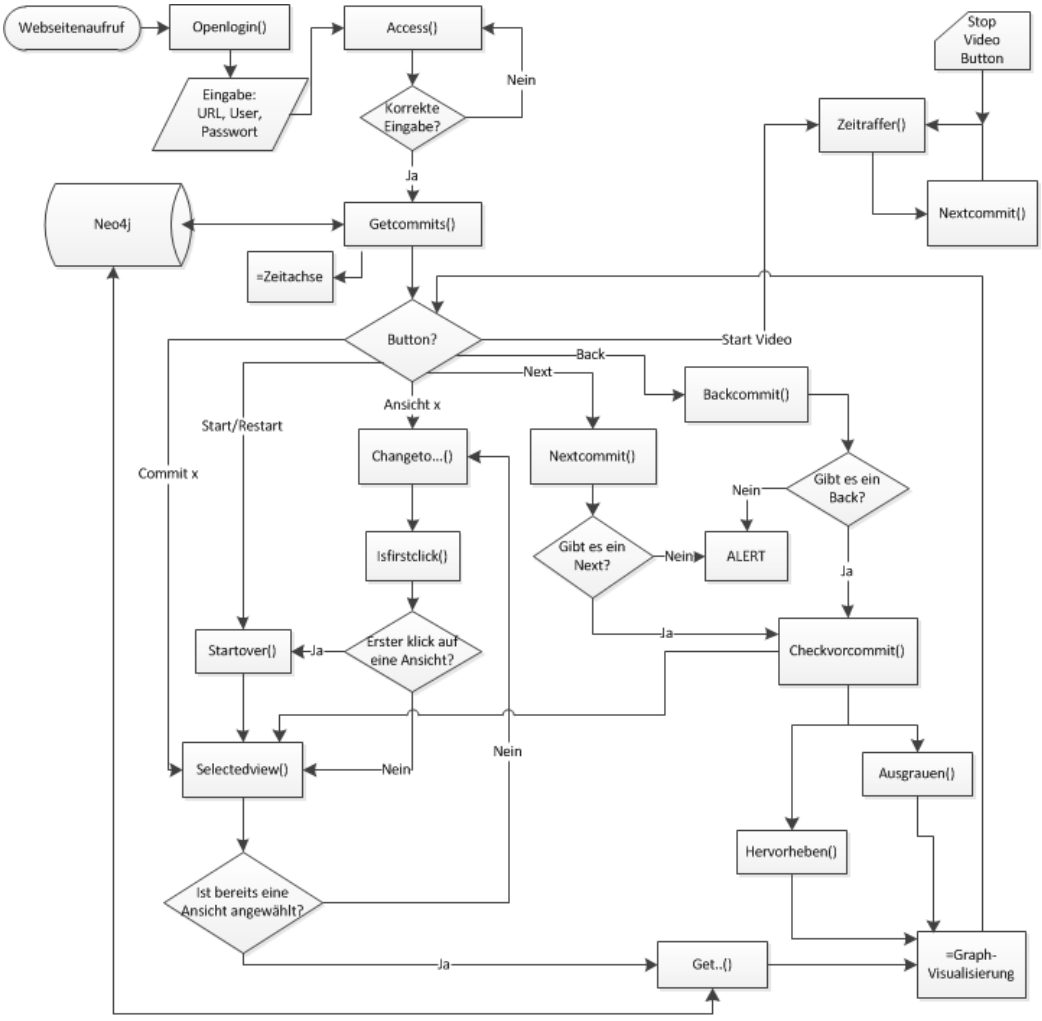

Abbildung 7: Grober Programmablaufplan der Webanwendung

Abbildung 9 zeigt den groben Ablauf des Programms und dessen Funktionsaufrufe. Dabei ist auf die unterschiedlichen Möglichkeiten je nach Reihenfolge der Betätigung der Buttons oder Einstellungen eingegangen worden. Bei korrekter Eingabe des Datenbank-Logins werden die Commits aus der Datenbank extrahiert und die Zeitachse visualisiert. Dann kommt es darauf an, welchen Button man wählt, in welcher Reihenfolge die Funktionen aufgerufen werden, aber alle führen zu einer entsprechenden Graph-Visualisierung.

4 Zusammenfassung

In diesem letzten Kapitel wird noch einmal zusammengefasst, was das Ergebnis des Praktikums war und was ich dabei gelernt habe. Außerdem wird ein kurzer Ausblick auf die Zukunft gegeben; was ich aus dem Praktikum mitnehmen werde und wie es möglicherweise weitergehen könnte, bezüglich des Projekts und meines weiteren Lebensweges.

4.1 Ergebnis und Erlerntes

In den 17 Wochen Arbeit ist eine Webanwendung entstanden, die ihre Anforderungen auf Grundlagen-Ebene erfüllt, mit viel Möglichkeit für Ausbau und Erweiterungen. Zu den erfüllten Anforderungen gehört die Extrahierung und Visualisierung der Softwarearchitektur in Form von Graphen zu allen Zeitpunkten ihrer Evolution. Es wurde ein Zeitraffer implementiert und die Veränderung in der Historie, sowie der manuelle Wechsel zwischen den einzelnen Commits und Ansichten. Durch Anwählen von Knoten oder Kanten lassen sich weitere Informationen abrufen, und über eine Suchleiste mit Autovervollständigung lässt sich direkt nach Knoten suchen. Die möglichen Ansichten der Architektur beschränkt sich auf Class-, Package-, Bundle- und Service-Ebene und eine zusätzliche benutzerdefinierte Ansicht, in der sich zuvor ausgewählte Knoten befinden. Zwei der wichtigsten nicht erfüllten Anforderungen sind die Positionsübergabe der Knoten und das Hervorheben oder ausgrauen der Veränderungen beim Commitwechsel.

Zu dem im Praktikum erlernten Dingen gehören alle Sachinhalte, Wissen und Sprachen, die mich zum Ergebnis geführt haben. Ich habe mich in das Thema ausführlich eingearbeitet und mich damit beschäftigt, neue Programmier- und Abfragesprachen gelernt, mich mit neuen Inhalten, Programmen und Bibliotheken auseinandergesetzt, und ich habe so eine eigene Webanwendung aufgebaut. Ich habe dieses Projekt selbstständig bearbeitet, weder mit Vorwissen noch mit Arbeitserfahrung. Am Ende nehme ich daraus sowohl die technischen Skills als auch neue Softskills mit. Dieses Praktikum hat mir einen guten Einblick in die Arbeitswelt und meine mögliche Zukunft gegeben.

Im Rückblick auf das an der HTW angeeignete Wissen habe ich mehr Neues gelernt als Altes angewendet, bis auf das gute Grundlagenwissen, ohne das ich dieses Projekt nicht hätte schaffen können. Das liegt aber auch an der Tatsache, dass sich meine Arbeit im Praktikum auf den Bereich Software Engineering beschränkt hat, welcher im Studium auch nur einen Bruchteil ausmachte. Neu erlernt habe ich, eine Webseite aufzubauen, und mich mit den dazu benötigten Sprachen Javascript, HTML und CSS auseinandergesetzt.

Ich habe gelernt, mit Graphdatenbanken umzugehen, speziell Neo4j inklusive der Abfragesprache Cypher. Außerdem habe ich mich in verschiedenste neue Themengebiete eingelesen und mir Wissen über die Visualisierung von Daten und über Graphen angeeignet.

Zu den „lessons learned“ für zukünftige Projekte gehören vor allem bessere Planung und Zeitmanagement. Ich habe im Laufe des Projekts immer wieder Entscheidungen überdenken müssen oder sie erst nach einem Arbeitspaket für das nächste getroffen. Somit konnte ich nicht immer so schnell arbeiten, wie ich wollte, und musste öfter anderes dazwischen schieben, was die Planung wieder zunichte machte. Jetzt weiß ich, wie wichtig die Planung wirklich ist, und das werde ich für kommende Projekte berücksichtigen. Das Thema Zeitmanagement gehört mit in die ausführliche Planung des Projekts, aber dazu kommt noch, wie hoch meine Prioritäten für die Arbeit liegen. Durch mein schlechtes Zeitmanagement bin ich gegen Ende sehr gestresst gewesen und habe nicht alle Anforderungen erfüllen können. Das sollte sich in Zukunft vermeiden lassen, vor allem durch volle Konzentration auf eine ausführliche Planung am Anfang des Projekts, an die ich mich dann auch halten muss.

4.2 Ausblick in die Zukunft

Ich habe durch die Vollzeit-Arbeit in einem Forschungsinstitut festgestellt, dass mir sowohl die 39-Stunden-Woche als auch der Bereich der Forschung im Moment eher weniger zusagen. Deshalb werde ich definitiv zunächst noch einen Master anstreben. Aber der generelle Bereich der Informatik, das Programmieren und Arbeiten am eigenen Projekt haben mir sehr zugesagt, weshalb sich hier auch der Studienschwerpunkt bilden wird. Ich denke, dass mir die Arbeit mit Datenbanken und Visualisierungen auch weiterhin den Weg bereiten wird, und ich kann mir gut vorstellen, später in diesen Bereichen zu arbeiten.

Mit dem aktuellen Stand des Projekts werde ich weiterarbeiten und darauf dann meine Bachelorarbeit aufbauen. Mein Ziel ist es, das Projekt weiterzuentwickeln und dabei den Fokus vor allem auf die Erweiterung des Datenbankschemas zu setzen. Die Programmierung wurde bisher auf einen festen Datensatz und dessen Schema beschränkt, weshalb die Webanwendung nur auf Software mit demselben Architekturaufbau angewendet werden kann. In meiner Bachelorarbeit mit dem Titel „Weiterentwicklung eines webbasierten 2D-Visualisierungstools für Software-Evolutionsgraphen zur Anwendung auf diverse Datenbankschemata“, soll auf den aktuellen Stand und alle relevanten Erweiterungs- und Optimierungsmöglichkeiten eingegangen werden. Sowohl die, die ich im Praktikum zeitlich nicht umsetzten konnte, als auch neue sinn-

volle Anforderungen sollen entwickelt werden. Im Vordergrund steht dann die Dynamisierung der bisher statischen Bindung an das Datenbankschema. Dafür muss mindestens ein Konzept entstehen, wie auf andere Architekturmodelle, andere Software und andere Sprachen erweitert werden kann.

Literatur

[1] *Das DLR in Zahlen und Fakten.* URL: https://www.dlr.de/dlr/Portaldata/1/Resources/documents/2019/DLR_FactsFigures_2017_DE_web.pdf. (Abruf: 18.04.2019, Diese Quelle bezieht sich auf alle Fakten dieses Kapitels).

[2] *DLR-Webseite der Einrichtung Simulations- und Softwaretechnik.* URL: https://www.dlr.de/sc/desktopdefault.aspx/tabid-1177/. (Abruf: 23.04.2019, Diese Quelle bezieht sich auf alle Fakten dieses Abschnitts).

[3] *DLR-Webseite der Abteilung IVS.* URL: https://www.dlr.de/sc/desktopdefault.aspx/tabid-1199/1657_read-3066/. (Abruf: 23.04. 2019, Diese Quelle bezieht sich auf alle Fakten dieses Abschnitts).

[4] Marlene Brüggemann. *Bachelorarbeit: Visualisierung von mit OSGi-Komponenten realisierten Softwarearchitekturen im 3-dimensionalem Raum mit Virtual Reality.* URL: https://elib.dlr.de/108828/1/Bachelorarbeit_MarleneBrueggemann.pdf. (Abruf: 18.04.2019).

[5] Tobias Marquardt. *Masterarbeit: Extraktion und Visualisierung von Beziehungen und Abhängigkeiten zwischen Komponenten großer Softwareprojekte.* URL: https://elib.dlr.de/105575/1/Masterarbeit_MarquardtTobias.pdf. (Abruf: 18.04.2019).

[6] Lynn von Kurnatowski. *Masterarbeit: Visualisierung der Evolution von Softwarearchitektur.* URL: https://elib.dlr.de/121897/1/complete.pdf. (Abruf:18.04.2019).

[7] Doreen Seider u. a. *Paper: Visualizing Modules and Dependencies of OSGi-based Applications.* URL: https://elib.dlr.de/110129/1/vissoft-toolpaper-osgivis_SeiderEtAl.pdf. (Abruf: 18.04.2019).