Lars Tralle
Matrikelnummer: 2783103

Visualisierung und Verwaltung von Entscheidungswissen in JIRA

Bachelorarbeit

Betreuung: Prof. Dr. Barbara Paech, Anja Kleebaum
Institut für Informatik
Ruprecht-Karls-Universität Heidelberg

27. Mai 2019

Kurzbeschreibung

Während der Softwareentwicklung treffen Entwickler kontinuierlich Entscheidungen. Das Wissen, das sie dabei aufbauen, wird als Entscheidungswissen bezeichnet und umfasst neben den getroffenen Entscheidungen auch die dahinterliegenden Entscheidungsprobleme, alternative Lösungsmöglichkeiten, sowie deren Vor- und Nachteile. Verlassen Mitarbeiter ein Projekt besteht die Gefahr, dass Wissen über ihre getroffenen Entscheidungen verloren geht. Um dies zu verhindern und neuen Entwicklern einen Überblick über bereits getroffene Entscheidungen zu geben, ermöglicht das ConDec JIRA Plug-In, Entscheidungswissen zu dokumentieren und mit Anforderungen oder Entwicklungsaufgaben zu verknüpfen. Ziel dieser Arbeit ist es, die Visualisierung und Verwaltung des Entscheidungswissens zu verbessern.
Als Alternative zur bestehenden Darstellung von Dokumentation und Zusammenhängen in Form eines Baumes wird eine Visualisierung als gerichteter Graph hinzugefügt. In dieser gibt es die Möglichkeit Elemente und Verknüpfungen zu verwalten. Zusätzlich ermöglichen es Filter, die Größe des dargestellten Baumes oder Graphen zu reduzieren, indem Elemente, die nicht den Filterkriterien entsprechen, ausgeblendet werden. Eine erste Evaluation der in der Arbeit entwickelten Erweiterung zeigt, dass die Visualisierung als gerichteter Graph sowie die Filtermöglichkeiten einfach zu verwenden sind und Zusammenhänge leichter erkennbar machen.

Abstract

During software development, developers are constantly making decisions. The knowledge that they build up is referred to as decision knowledge and includes not only the decisions made, but also the underlying decision-making problems, alternative solutions, as well as their advantages and disadvantages. If employees leave a project there is a risk that knowledge about their decisions will be lost. To prevent this and to give new developers an overview of decisions that have already been made, the ConDec mbox JIRA plug-in makes it possible to document decision knowledge and link it with requirements or development tasks. The goal of this work is to improve the visualization and management of decision-making.
As an alternative to the existing representation of documentation and contexts in the form of a tree, a visualization as a directed graph is added. In this visualization there is the possibility to manage elements and links. In addition, filters make it possible to reduce the size of the displayed tree or graph by hiding elements that do not meet the filter criteria. An initial evaluation of the extension developed in the work shows that the visualization as a directed graph as well as the filter options are easy to use and make contexts easier to recognize.

Eidesstattliche Erklärung zur Bachelorarbeit

Hiermit erkläre ich, dass ich diese Bachelorarbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt und die aus fremden Quellen direkt oder indirekt übernommenen Gedanken als solche kenntlich gemacht habe.

Ich habe die Grundsatze und Empfehlungen „Verantwortung in der Wissenschaft“ der Universität Heidelberg gelesen und befolgt.

Die Arbeit habe ich bisher keinem anderen Prüfungsamt in gleicher oder vergleichbarer Form vorgelegt. Sie wurde bisher nicht veröffentlicht.

Heidelberg, 27. Mai 2019	
Ort, Datum	Unterschrift

Inhaltsverzeichnis

Akronyme

ITS Issue-Tracking-System. 1, 5, 24, 26, 33

NFR Nichtfunktionale Anforderungen. 32

RQ Research Question. 9

SF Systemfunktionen. 23, 26–30, 37–43, 56, 69

ST Sub-Tasks. 23, 25, 26

TAM Technology Acceptance Model. 55

UT User-Tasks. 23, 25

VCS Versionskontollsysteme. 1

1 Einleitung

1.1 Motivation

In der Softwareentwicklung werden oft Entscheidungen getroffen, etliche jedoch nicht oder nur unzureichend dokumentiert. Diese unzureichende Dokumentation schafft verschiedene Probleme: Neue Entwickler benötigen mehr Zeit, um sich in bestehende Projekte einzuarbeiten. Zudem besteht die Gefahr, dass Entwickler, die aus einem Projekt ausscheiden, ihr Wissen mitnehmen. Dann braucht es Zeit, um von ihnen getroffene Entscheidungen zu rekonstruieren.

Wichtig für die Lösung dieser Probleme ist, getroffene Entscheidungen strukturiert zu dokumentieren und für andere Projektbeteiligte zugänglich zu machen. Ein erster Ansatz zur Entscheidungsdokumentation ist, diese in natürlicher Sprache zu verfassen, zum Beispiel in Kommunikationswerkzeugen. In vielen Softwareprojekten werden zudem Versionskontollsysteme (VCS) wie zum Beispiel git[1] verwendet. In diesen ist es möglich, Entscheidungen, die während der Entwicklung getroffen werden, in den Commit-Nachrichten zu dokumentieren.

Einen strukturierten Ansatz zur Dokumentation von Entscheidungswissen bietet das ConDec JIRA Plug-In. Dieses ist in das weit verbreitete Issue-Tracking-System (ITS) JIRA integriert. ConDec bietet die Möglichkeit, Entscheidungen basierend auf einem Problem, für das eine Lösung gefunden werden soll, explizit und strukturiert zu erfassen.

1.2 Ziel der Arbeit

Ziel der Arbeit ist es, Visualisierung und Verwaltung von Entscheidungswissen im ConDec JIRA Plug-In zu optimieren. Um diese Verbesserung zu erreichen, sollen Entscheidungswissen und weitere Anforderungsartefakte sowie die Zusammenhänge zwischen diesen als gerichteter Graph visualisiert werden.

[1] https://git-scm.com/

Durch die Anwendung von Filtern auf die neue Visualisierung und die bereits bestehende Baumdarstellung soll die Übersichtlichkeit verbessert werden und so Zusammenhänge zwischen einzelnen Artefakten leichter erkennbar werden.

1.3 Struktur der Arbeit

Zunächst werden in Kapitel 2 Grundlagen für ein besseres Verständnis dieser Arbeit vermittelt. Zu diesen gehören unter anderem Modelle zur Dokumentation von Entscheidungswissen (IBIS-Modell[1], Decision Documentation Model[2], [3]) und eine Übersicht über die bisherige Möglichkeit, Entscheidungswissen und seine Zusammenhänge im ConDec JIRA Plug-In als Baum zu visualisieren. Des Weiteren werden Grundlagen und Unterschiede von Graphen und Bäumen als Darstellungsform verknüpfter Informationen erläutert und es gibt einen Überblick über das Forschungsgebiet der Wissensvisualisierung.

Kapitel 3 beschreibt eine systematische Literaturrecherche, in der Literatur zu bestehenden Ansätzen zur Visualisierung von Entscheidungen in Softwareprojekten gesucht und verglichen wird. Ziel ist es, auf diese Arbeit übertragbare Ansätze zu finden. In Kapitel 4 werden Anforderungen an die neu entwickelte Visualisierung formuliert. Kapitel 5 beschreibt den Entwurf und getroffene Entscheidungen sowie gelöste Probleme während der Implementierung. Dieses Kapitel enthält auch die Ergebnisse in Form von Bildschirmfotos.

In Kapitel 6 wird beschrieben, wie die Qualität der entwickelten Software sichergestellt wird. Kapitel 7 enthält Beschreibung und Ergebnisse einer Evaluation, die Benutzbarkeit und Nützlichkeit des Prototypen sicherstellen soll. Als letztes werden in Kapitel 8 die Ergebnisse der Arbeit zusammengefasst und Probleme sowie mögliche Verbesserungen diskutiert.

2 Grundlagen

In diesem Kapitel werden Begriffe und Grundlagen für ein besseres Verständnis dieser Arbeit behandelt. Es werden Modelle zur Dokumentation von Entscheidungswissen und das ConDec JIRA Plug-In vorgestellt. Das derzeit im Plug-In verwendete Framework treant[1] ermöglicht nur eine Baumdarstellung von Entscheidungswissen bezogen auf Anforderungen und Entwicklungsaufgaben. In dieser Arbeit soll als Alternative eine Darstellung als gerichteter Graph entwickelt werden. Die Grundlagen beider Darstellungen und vor allem die jeweiligen Vor- und Nachteile werden beschrieben.

2.1 Dokumentation von Entscheidungswissen

Ein Modell, um Entscheidungen strukturiert zu dokumentieren, ist das IBIS-Modell von Kunz und Rittel [1]. In Abbildung 2.1 sind Komponenten und Zusammenhänge dieses Modells dargestellt. *Issues* beschreiben die Entscheidungsprobleme, *Positions* repräsentieren verschiedene Lösungsansätze und *Arguments* unterstützen eine Position oder erheben Einwände gegen sie.

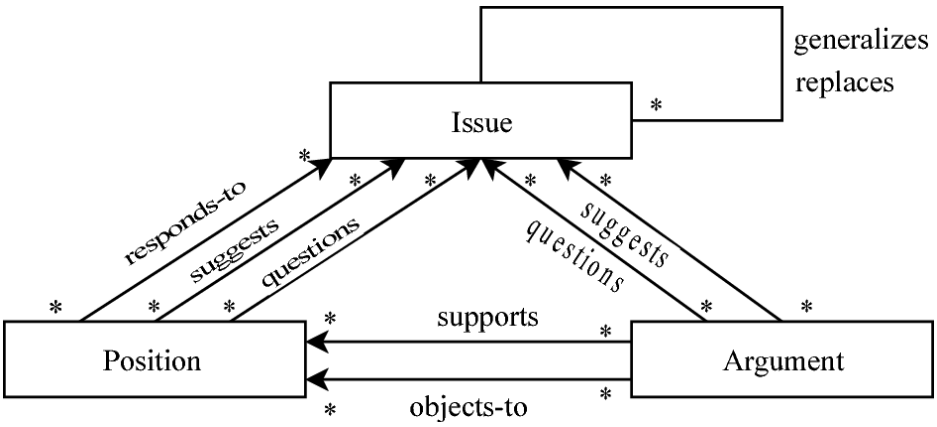

Abbildung 2.1: Darstellung der Wissenstypen und ihrer Zusammenhänge im IBIS-Modell [1]

[1] https://fperucic.github.io/treant-js/

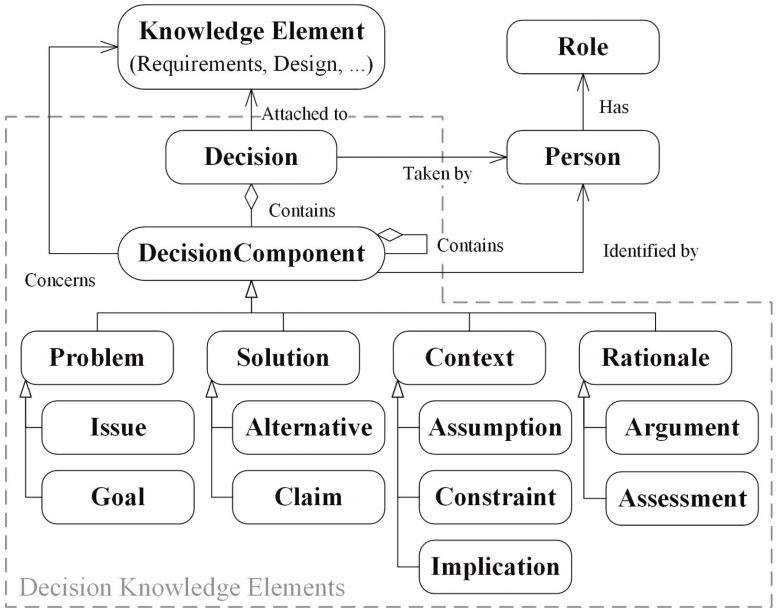

Abbildung 2.2: DecDoc Modell zur Dokumentation von Entscheidungen(*Hesse et al.*) [2], [3]

Hesse *et al.* beschreiben ein ausführlicheres Modell zur Entscheidungsdokumentation [2], [3] (Vergleiche Abbildung 2.2). In diesem dient eine *Entscheidung* (Decision) als Anker für das Entscheidungsproblem und kann einer Person in einer bestimmten Position innerhalb eines Projekts zugeordnet sein. Entscheidungen können mit weiteren Wissenselementen verknüpft sein und bestehen selbst aus einzelnen Entscheidungskomponenten (*Decision Components*):

- **Problem:** Neben dem eigentlichen Problem (*Issue*), das eine nicht den Anforderung entsprechende oder generell nicht zufriedenstellenden Gegebenheit beschreibt, kann auch ein Ziel (*Goal*) als Wunschlösung formuliert werden.
- **Lösung (*Solution*):** Die Lösung besteht aus Alternativen, die alle eine mögliche Lösung des Problems beschreiben. Neben diesen können auch Ansprüche (*Claims*) gestellt werden, um die Anforderungen an die Lösung zu präzisieren.
- **Kontext:** Der Kontext ermöglicht es, zusätzliche Annahmen (*Assumptions*), Einschränkungen (*Constraints*) und Auswirkungen (*Implications*) in Bezug auf die Entscheidung zu formulieren.
- **Begründung (*Rationale*):** Die Begründung enthält Pro- und Kontra-Argumente, die eine Position für oder gegen einzelne Bestandteile der Entscheidung vertreten. Auch wird hier eine Bewertung (*Assessment*) ermöglicht.

2.2 Das ConDec JIRA Plug-In

JIRA[2] ist ein weit verbreitetes ITS. In JIRA lassen sich viele Artefakte, die in der Softwareentwicklung entstehen, verwalten. Allerdings bietet es keine Möglichkeit, Entscheidungswissen strukturiert zu dokumentieren und abzurufen.

Kleebaum *et al.* definieren Anforderungen an ein Tool, das Entscheidungsdokumentation in Softwareprojekten unterstützt [4], [5]. Dieses Tool wird im Rahmen des CURES-Projekts[3] unter dem Namen ConDec entwickelt. Innerhalb der ConDec Tools gibt es das ConDec JIRA Plug-In, das Dokumentation, Verwaltung und Visualisierung von Entscheidungswissen in JIRA integriert.

Das Plug-In bietet zwei verschiedene Orte, um Entscheidungswissen anzuzeigen und zu verwalten:

- **Decision Knowledge Page:** Abbildung 2.3 zeigt die Decision Knowledge Page des ConDec JIRA Plug-Ins. In der Liste auf der linken Seite werden alle Entscheidungen eines vom Benutzer ausgewählten Wissenstyps angezeigt. Wird ein Element ausgewählt, werden im rechten Teil die mit diesem Element verknüpften Wissenselemente in Form eines Baums visualisiert. In dieser Visualisierung kann der Benutzer mit Hilfe von Drag-and-Drop oder eines Kontextmenüs Wissenselemente erstellen, bearbeiten und miteinander verknüpfen.

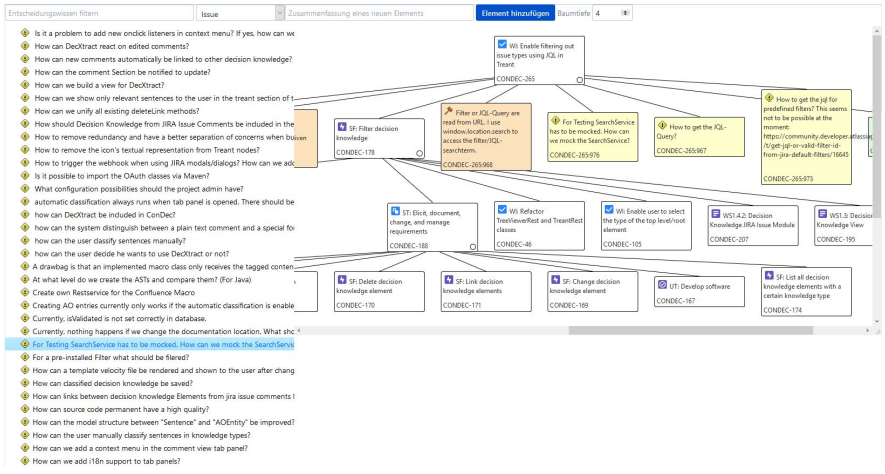

Abbildung 2.3: ConDec Decision Knowledge Page in Jira mit Listenansicht von Entscheidungselementen und Visualisierung

[2]https://de.atlassian.com/software/jira
[3]http://www.dfg-spp1593.de/cures/

- **JIRA Issue Modul:** Abbildung 2.4 zeigt das JIRA Issue Modul des ConDec Plug-Ins. In diesem wird das zu dem aktuell ausgewählten JIRA-Issue verknüpfte Wissen dargestellt. Auch inn dieser Ansicht ist ein Bearbeiten von Elementen und ihrer Verknüpfungen möglich.

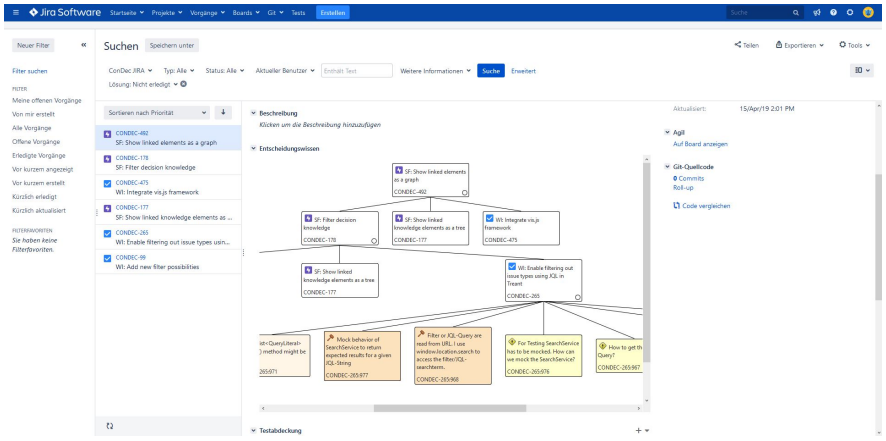

Abbildung 2.4: Jira Issue Ansicht mit Filtermöglichkeit und Modul zur Visualisierung von Entscheidungswissen

2.3 Grundlagen gerichteter Graphen und Bäume

Krischke und Röpcke [6] definieren Graphen und Bäume wie folgt:

Graphen sind Strukturen, die Objekte und ihre Zusammenhänge darstellen. Ein Graph $G(V, E)$ besteht aus einer Menge von Knoten (V), die die Objekte repräsentieren, und einer Menge von Kanten (E), die die Zusammenhänge zwischen Objekten darstellen. Ein Graph wird als gerichtet bezeichnet, wenn die Kanten in eine Richtung von einem Knoten ausgehend zu einem anderen zeigen. Dies wird mit Hilfe eines Pfeils dargestellt. Werden in dieser Arbeit Graphen erwähnt, sind damit immer gerichtete Graphen gemeint.

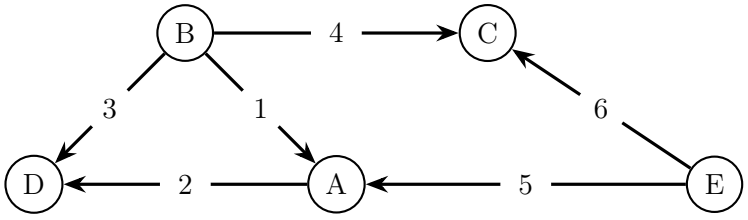

Abbildung 2.5: Beispiel eines gerichteten Graphen

Bäume sind Graphen, die bestimmten Einschränkungen unterliegen. So dürfen Bäume keine geschlossenen Kreise bilden, der Weg von einem Knoten zum anderen muss immer eindeutig sein. Von einem gewurzelten Baum spricht man, wenn es in einem gerichteten Baum einen Knoten gibt, der keine Eingangskante besitzt. Diesen Knoten nennt man auch die Wurzel. Werden in dieser Arbeit Bäume erwähnt, handelt es sich immer um einen gewurzelten Baum.

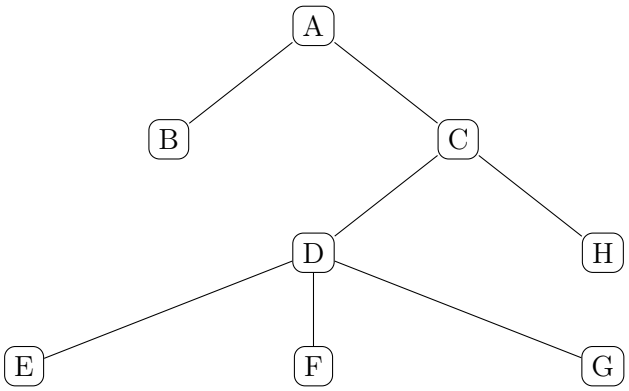

Abbildung 2.6: Beispiel eines ungerichteten Baums

2.4 Informations- und Wissensvisualisierung

Burkhard beschreibt in seiner Arbeit [7] Grundlagen für die Wissensvisualisierung und die Abgrenzung zur Informationsvisualisierung. Während Informationsvisualisierung dazu genutzt wird, große Mengen an Daten mit Hilfe von Software zu visualisieren und so neue Einblicke und Erkenntnisse zu ermöglichen, zielt Wissensvisualisierung darauf, mit visuellen Darstellungen die Übertragung von Wissen zwischen einzelnen Personen oder Gruppen zu verbessern. Zum Sicherstellen einer effizienten Übertragung von Wissen entwickelt Burkhard zehn Richtlinien. Zu diesen zählen unter Anderem:

- *„Compress your knowledge“*: Man soll sich auf die Qualität, und nicht die Quantität des zu visualisierenden Wissens konzentrieren.

- *„Present an overview and details“*: Eine Visualisierung von Wissen sollte immer erst einen Überblick geben, und weitere Informationen erst auf Anfrage des Nutzers liefern.

- *„Avoid decoration“*: Eine Visualisierung soll den Inhalt, und nicht sich selbst in den Mittelpunkt stellen.

3 Literaturrecherche

Diese systematische Literaturrecherche gibt anhand der gestellten Forschungsfragen einen Überblick über den Forschungsstand zum Thema Visualisierung von Entscheidungswissen. Neben den Forschungsfragen wird auch die Methodik der Recherche beschrieben und die Ergebnisse werden synthetisiert.

3.1 Forschungsfragen

Diese Forschungsfrage (Research Question (RQ)) ist mit Hilfe der Literaturrecherche zu beantworten:

- **RQ:**Welche Lösungen gibt es, Entscheidungen, das für ihr Verständnis nötige Wissen und deren Einfluss auf Softwareentwicklungsartefakte zu visualisieren?

Die Forschungsfrage dient dazu, einen Überblick über bestehende Tools oder Frameworks zu Dokumentation und Visualisierung von Entscheidungswissen in Software-Projekten zu schaffen.

3.2 Methodik

Für die Suche nach relevanter Literatur wird zwischen zwei Methoden unterschieden [8], [9]:

- Beim sogenannten Snowballing lässt sich über Zitate in und zu der bekannten Literatur weitere Literatur finden. Unterschieden wird zwischen vorwärts- und rückwärts gerichtetem Snowballing.
  Mit der ersten Methode, dem so genannten Forward-Snowballing, lässt sich Literatur finden, die auf die bekannte Literatur verweist. Die gefundenen Literatur ist also neuer als die Ausgangsliteratur.

Die zweite Methode, das so genannte Backward-Snowballing, wird genutzt, um Ergebnisse zu erhalten, die von der bereits bekannten Literatur zitiert werden.

- Bei der Suchterm-basierten Suche werden Suchterme und Suchmaschinen so gewählt, dass die Ergebnisse der Beantwortung der Forschungsfragen dienen.

Diese Literaturrecherche verwendet ausschließlich Snowballing, ausgehend von den Artikeln von Abad *et al.* [10] und Shahin *et al.* [11]. Diese werden zuerst gelesen und mögliche relevante Zitate für das Backward-Snowballing markiert. Um weitere Literatur zur Beantwortung der Forschungsfragen zu finden, wurden die Artikel in den Datenbanken von ACM[1] oder IEEE[2] aufgerufen. Die interessanten Zitate und Abhandlungen, die den bereits vorliegenden Artikel zitieren, werden dann nach Schlagwörtern und ihrer Zusammenfassung mit Hilfe von Relevanzkriterien auf ihren Beitrag zur Beantwortung der Forschungsfragen untersucht.

3.3 Relevanzkriterien

Die in Tabelle 3.1 aufgeführten Relevanzkriterien dienen dazu, die für die Beantwortung der Forschungsfragen relevante Literatur zu selektieren. Unterschieden wird in allgemeine Kriterien und spezielle Kriterien. Die allgemeinen Kriterien(ARK) sollen die Qualität der Literatur sicherstellen. Die speziellen Kriterien(SRK) dienen dem inhaltlichen Bezug zu den Forschungsfragen.

Tabelle 3.1: Relevanzkriterien für die Auswahl der Literatur

ID	Kriterium
RK1	Der Artikel ist in deutscher oder englischer Sprache verfasst.
RK2	Der Artikel ist für Mitglieder der Universität frei verfügbar.
RK3	Die Qualität des Artikels wird durch einen Review-Prozess sichergestellt.
RK4	Die Zusammenfassung weist darauf hin, dass mindestens ein Tool oder Framework zu Verwaltung und Visualisierung von Wissen beschrieben ist.

[1] http://portal.acm.org/
[2] http://ieeexplore.ieee.org/

Tabelle 3.2: Ergebnisse des Snowballings

Nr	Titel	Autor/innen	Jahr	Quelle	Ergebnisse	relevante Ergebnisse
1	Requirements Engineering Visualization: A Systematic Literature Review [10]	Zahra Shakeri Hossein **Abad**; Mohammad **Noaeen**; Guenther **Ruhe**	2016	IEEE Xplore	Forward 7 Backward 56	
2	Improving understandability of architecture design through visualization of architectural design decision [11]	Mojtaba **Shahin**; Peng **Liang**; Mohammad Reza **Khayyambashi**	2010	ACM	Forward 6 Backward 18	[12] [13]; [14]
3	A documentation framework for architecture decisions [12]	U. **van Heesch**; P. **Avgeriou**; R. **Hilliard**	2012	Elsevier	Forward 57 Backward 40	[15]
4	Does decision documentation help junior designers rationalize their decision? A comparative multiple-case study [15]	U. **van Heesch**; P. **Avgeriou**; A. **Tang**	2013	Elsevier	Forward 23 Backward 55	[16] [17]
5	A comparative study of architecture knowledge management tools[17]	Anthony **Tang**; Paris **Avgeriou**; Anton **Jansen**; Rafael **Capilla**; Muhammad Ali **Babar**	2009	Elsevier	Forward 44 Backward 73	[18]; [19]; [20]; [21]; [22]
6	SEURAT_Edu: A Tool to Assist and Assess Student Decision-Making in Design[16]	John **Malloy**; Janet **Burge**	2016	ACM	Forward 0 Backward 8	
7	A Web-based Tool for Managing Architectural Design Decisions [18]	Rafael **Capilla**; Francisco **Nava**; Sandra **Pérez**; Juan C. **Dueñas**	2006	ACM	Forward 42 Backward 25	
8	Software Architecture as a Set of Architectural Design Decisions [19]	Anton **Jansen**; Jan **Bosch**	2005	IEEE	Forward 140 Backward 22	

Nr	Titel	Autor/innen	Jahr	Quelle	Ergebnisse	relevante Ergebnisse
9	A rationale-based architecture model for design traceability and reasoning [20]	Antony **Tang**; Yan **Jin**; Jun **Han**	2007	Elsevier	Forward 160 Backward 49	
10	Knowledge Architect: A Tool Suite for Managing Software Architecture Knowledge [21]	Peng **Liang**; Anton **Jansen**; Paris **Avgeriou**	2009	Website der Universität Groningen[4]	Forward 20 Backward 11	
11	A Tool for Managing Software Architecture Knowledge [22]	Muhammad Ali **Babar**; Ian **Gorton**	2007	IEEE	Forward 152 Backward 33	
12	A Tool to Visualize Architectural Design Decisions [13]	Larix **Lee**; Philippe **Kruchten**	2008	Springer Link[3]	Forward 6 Backward 24	
13	Ontology-driven visualization of architectural design decisions [14]	Remco C. **de Boer**; Patricia **Lago**; Alexandru **Telea**; Hans **van Vliet**	2009	IEEE	Forward 6 Backward 18	

3.4 Ergebnisse

Tabelle 3.2 gibt einen Überblick über die durch Snowballing gefundene Literatur. Tabelle 3.3 gibt einen kurzen Einblick in Inhalte und Schlagwörter der einzelnen Artikel.

Alle Artikel erfüllen die Relevanzkriterien **RK1**, **RK2** und **RK3**. Die Ergebnisse Nummer 7 bis 13 beschreiben die Tools, die in den Arbeiten von Shahin *et al.* [11] und Tang *et al.* [17] verglichen werden. Auch die Arbeit von Malloy *et al.* [16] beschreibt ein Tool zur Dokumentation von Entscheidungen. Die Arbeiten von van Heesch *et al.* [12], [15] beschreiben ein Framework und seine Untersuchung in einer Studie. Sie alle erfüllen **RK4**, wie auch die Arbeiten von Shahin *et al.* [11], Abad *et al.* [10] und Tang *et al.* [17], da sie vergleichende Studien oder Literaturrecherchen zu Tools und Frameworks zur Entscheidungsdokumentation durchführen.

[3] Die Literatur von Springer ist auf der Seite des Verlags nicht kostenlos, lässt sich aber mit Hilfe von Google-Scholar(http://scholar.google.com) frei Verfügbar finden.

[4] https://www.rug.nl/research/portal/en/publications/knowledge-architect(8b394111-bd29-4af5-9d6a-1314b39f32c4).html

Tabelle 3.3: Inhalt der gefundenen Literatur

Artikel	Schlagworte	Zusammenfassung
[10]		Eine systematische Literaturrecherche, die einen Überblick über die Ansätze von Visualisierung im Requirements Engineering gibt.
[11]	Software Architecture, Design Rationale, Architectural Design Decision, Rationale Visualization	Eine Untersuchung bestehender Ansätze zur Modellierung, Erfassung, Visualisierung und Verwaltung von Architektur-Entscheidungs-Wissen. Außerdem entwerfen und validieren die Autoren eine Umsetzung basierend auf Compendium[5].
[12]	Software Architecture, Architecture Decisions, Architecture Knowledge Management, Architectural Viewpoints, Case Study, Architecture Framework	Vorstellung und Evaluation eines Dokumentations-Frameworks für Architektur-Entscheidungen mit vier Perspektiven, basierend auf dem ISO/IEC/IEEE42010-Standard.
[15]	Software Architecture, Architecture Decisions, Viewpoints, ISO/IEC/IEEE42010, Design Reasoning, Case Study	Durchführen einer Studie zur Überprüfung, ob das in [12] entwickelte Framework Junior-Entwickler dabei unterstützt, bessere Entscheidungen während des Designprozesses zu treffen. Als Ergebnis zeigt sich, dass Optionen systematischer durchsucht und evaluiert werden. Jedoch hilft es nicht, Anforderungen und Komplexität zu verwalten.
[17]	Architectural Knowledge Management Tool, Architectural Design, Design Rationale	Ein Vergleich von fünf Architektur-Wissens-Management Tools, um deren Vor- und Nachteile sowie ihre Konformität zu aktuellen Standards zu untersuchen.

[5] http://compendium.open.ac.uk/

Artikel	Schlagworte	Zusammenfassung
[16]	Design Rationale, Decision-Making, Learning Management	Beschreibung und Evaluation eines web-basierten Tools, das vor allem Studierenden helfen soll, verschiedene Optionen für Probleme, die im Prozess der Softwareentwicklung auftreten, zu erfassen und die bestmögliche Lösung für diese zu finden. Es zeigt sich, dass Studierende mehrere Alternativen in Erwägung ziehen und stärker über ihre Entscheidungsprozesse nachdenken.
[18]	Software architecture, architecture design decisions, software patterns, requirements, traceability	Beschreibung eines web-basierten Tools zum Erfassen und Verwalten von Architektur-Design-Entscheidungen.
[19]		Software-Architektur wird als ein Satz von Entscheidungen betrachtet. Die Dokumentation dieser Entscheidungen soll das Verschwinden des Wissens über die Architektur verhindern.
[20]	Design rationale, Architecture design, Traceability	Entwicklung eines Tool-Sets zur Erfassung von Architektur-Entscheidungen in UML. Dies soll ein besseres Verständnis ermöglichen.
[21]		Entwicklung der Tool Suite Knowledge Architect(KA). KA wird zum Erstellen, Nutzen Teilen und Organisieren von Architektur-Wissen verwendet. KA besteht aus dem Knowledge Repository, in dem das Wissen gespeichert wird, dem Document Knowledge Client zum Erfassen von Wissen in Microsoft Word, einem Excel- und Python Plug-In, dem Knowledge Explorer zum Visualisieren gespeicherten Wissens sowie dem Knowledge Translator, der zur Transformation zwischen verschiedenen Domänenmodellen genutzt werden kann.

Artikel	Schlagworte	Zusammenfassung
[22]		Beschreibung eines Tools zum Verwalten von Architektur-Wissen und -Rationalen. Wissen soll erfasst und genutzt werden, um so den Architektur-Prozess zu verbessern.
[13]		Vorstellung eines Tools, das das Finden und Analysieren von Entscheidungen durch deren Visualisierung ermöglicht und so die Qualität von Software-Architektur verbessern soll.
[14]		Aufzeigen, inwieweit Ontologie basierte Visualisierung von Architektur-Design-Entscheidungen externen Prüfern helfen kann, die Qualität von Software zu kontrollieren.

3.5 Synthese

In der gefundenen Literatur werden neun verschiedene Tools zur Erfassung und Verwaltung von Entscheidungswissen beschrieben. Diese wurden auf die Unterstützung von bestimmten Funktionalitäten und Ansichten untersucht. Als Kriterium für die Qualität der Tools wurde geprüft, ob diese evaluiert sind.

Die untersuchten **Funktionalitäten** sind basierend auf der Arbeit von Shahin *et al.* [11]:

- **Änderungsauswirkungsanalysen:** Dies ist eine Funktionalität, die die Möglichkeit bietet den Einfluss von Änderungen an Entscheidungen auf andere Entscheidungen oder Einträge zu erkennen und dem Nutzer zu zeigen.
- **Filtermöglichkeit:** Nutzer können die Einträge in der Datenbank nach gewissen Kriterien durchsuchen.
- **Nachverfolgbarkeit:** Der Weg vom Ursprung zu einer Entscheidung lässt sich nachvollziehen.

- **Quantitative Analyse:** Für verschiedene Alternativen wird überprüft, ob diese ausgewählten Qualitätsanforderungen entsprechen. Die Alternativen werden auf Basis der Ergebnisse eingestuft. Dies soll bei der Entscheidungsfindung helfen.

- **Gemeinsame Entwicklung:** Verschiedene Stakeholder können trotz unterschiedlicher Anforderungen an die Tools und trotz räumlicher Trennung gemeinsam an der Dokumentation von Entscheidungen arbeiten.

Die **Ansichten** sind basierend auf der Arbeit von van Heesch *et al.* [12]:

- **Decision-Relationship:** Entscheidungen und ihre Verknüpfungen untereinander werden visualisiert. Diese Ansicht ist für die Änderungsauswirkungsanalyse und einen Überblick über alle Entscheidungen hilfreich.

- **Decision-Chronology:** Diese Ansicht bietet als einzige eine zeitliche Komponente. Die Evolution der Entscheidungen wird visualisiert.

- **Decision-Stakeholder-Involvement:** In dieser Ansicht werden die Zuständigkeiten der verschiedenen Stakeholder im Entscheidungsprozess angezeigt.

- **Decision-Detail:** Für die Entscheidungen wird eine ausführliche Beschreibung angeboten. In dieser werden unter anderem das Entscheidungsproblem, die Lösung und Alternativen angezeigt.

Tabelle 3.4 gibt einen Überblick, welche Tools welche Funktionalitäten und Ansichten unterstützen und ob sie evaluiert wurden. Nachfolgend sind Umsetzungen von Funktionalitäten und Ansichten beschrieben und werden Beispiele in den Abbildungen 3.1, 3.2, 3.3, 3.4, 3.5 und 3.6 gezeigt.

Tabelle 3.4: Übersicht über die Features der verschiedenen Tools[6]

	Funktionalität					**Ansicht**				
Tool	**Änderungsauswirkungsanalysen**	**Filtermöglichkeit**	**Nachverfolgbarkeit**	**Quantitative Analyse**	**Gemeinsame Entwicklung**	**Decison-relationship**	**Decision-Chronology**	**Decision-Stakeholder-involvement**	**Decision-Detail**	**Evaluation**
SEURAT_Edu [16]	?	?	?	?	+	?	−	−	+	+
ADDSS [18]	+	?	?	?	+	+	+	+	+	+
Archium [19]	−	−	+	−	−	+	?	?	+	+
AREL [20]	?	?	+	?	?	+	?	?	?	+
The Knowledge Architect [21]	+	+	+	+	+	+	?	?	+	+
PAKME [22]	?	+	?	?	+	+	?	?	+	+
Kruchtens ADD Ontology Tool [13]	+	−	−	−	−	+	+	?	+	+
Ontology-Driven Visualization Tool [14]	+	−	−	+	−	?	?	?	?	+
Compendium-based ADD Tool[11]	+	+	+	−	+	+	?	?	+	+

[6] + Tool unterstützt Ansicht/Funktionalität; - Tool unterstützt Ansicht/Funktionalität nicht; ? keine Angabe

Änderungsauswirkungsanalyse
Es gibt zwei Ansätze, um die Änderungsauswirkungsanalyse in die Tools zu integrieren. Entweder werden die Zusammenhänge und so die Einflüsse von Änderungen in der Decision-Relationship-Ansicht angezeigt (Siehe Abbildung 3.1) oder es wird eine Ursache-Wirkungs-Matrix verwendet (siehe Abbildung 3.2).

Filtermöglichkeit
Die Tools, die diese Funktionalität unterstützen, bieten dem Nutzer eine Suchfunktion, mit der die zu visualisierenden Daten durchsucht werden können. Zusätzlich können die Ergebnisse noch nach den verschiedenen Typen des zugrundeliegenden Modells gefiltert werden.

Nachverfolgbarkeit
Zur Umsetzung dieser Funktionalität gibt es verschiedene Ansätze. AREL[20] erweitert die UML um entsprechende Verknüpfungen. Als Anwendung gibt es ein Plug-In für die kommerzielle Software Enterprise Architect[7]. Eine andere Möglichkeit ist die Visualisierung mit gerichteten Verknüpfungen verschiedener Einträge (siehe Abbildungen 3.3 und 3.4).

Quantitative Analyse
Das Knowledge Architecture Toolset [21] unterstützt diese Funktionalität durch das Excel Plug-In oder das Python Plug-In. Im Tool von de Boer *et al* [14] sind unterschiedliche Qualitätsattribute wählbar. Diese lassen sich auf einer Skala von 1 bis 100 priorisieren. Nach der Auswahl verschiedener Alternativen wird sowohl der positive als auch der negative Rang für die gewählten Qualitätsattribute berechnet (siehe Abbildung 3.5).

Gemeinsame Entwicklung
Alle Tools, die diese Funktionalität unterstützen, speichern ihre Daten zentral, so dass entweder über lokale oder web-basierte Programme darauf zugegriffen werden kann.

Decision-Relationship Ansicht
Die Zusammenhänge zwischen einzelnen Entscheidungen werden als Graphen visualisiert (siehe Abbildungen 3.1, 3.3 und 3.4). In diesem sind Entscheidungen als Knoten und Zusammenhänge als Kanten dargestellt.

[7] https://www.sparxsystems.de/

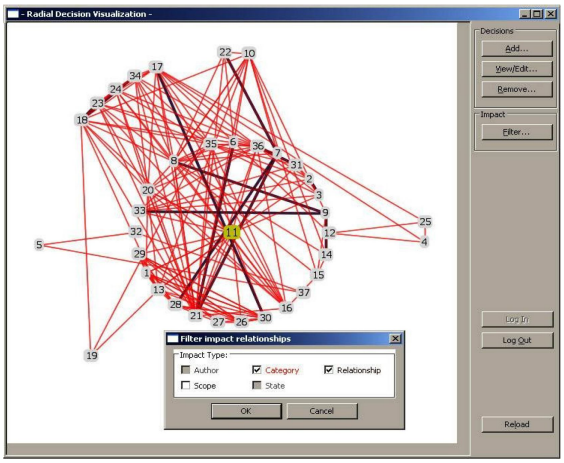

Abbildung 3.1: Visualisierung von Beziehungen (breite Linien) und Änderungsauswirkungen (dünne Linien) in Kruchtens ADD Ontology View [13]

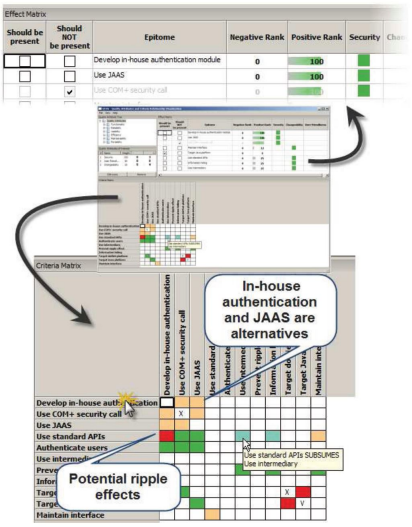

Abbildung 3.2: Ursache-Wirkungs-Matrix im Ontology-Driven Visualization Tool [14] als Umsetzung der Änderungsauswirkungsanalyse

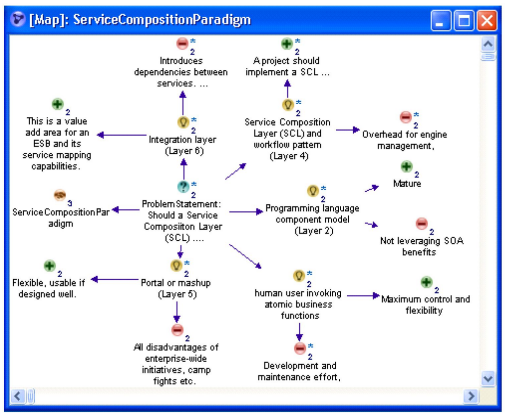

Abbildung 3.3: Visualisierung von Entscheidungen und deren Beziehungen zu anderen Einträgen im Compendium-basierten Tool[11]

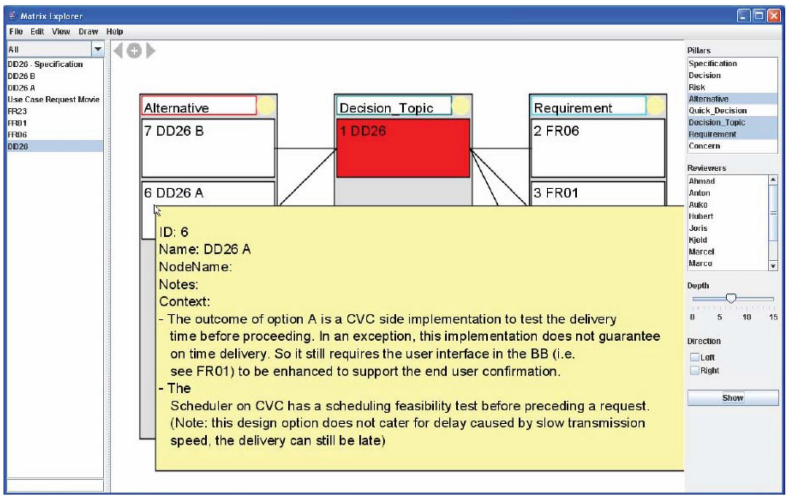

Abbildung 3.4: Knowledge Explorer des Knowledge-Architect Toolsets [21] als Beispiel für eine Decision-Relationship und eine Decision-Detail Ansicht

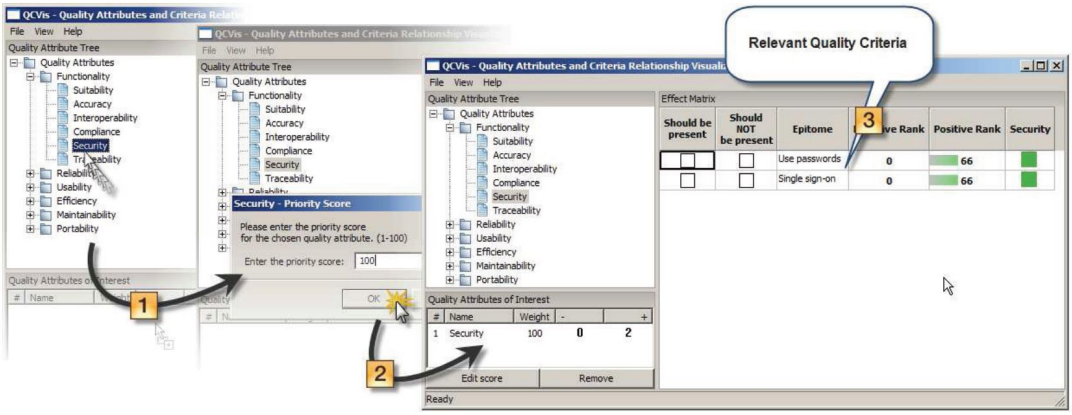

Abbildung 3.5: Quantitative Analyse im Ontology-Driven Visualization Tool [14]

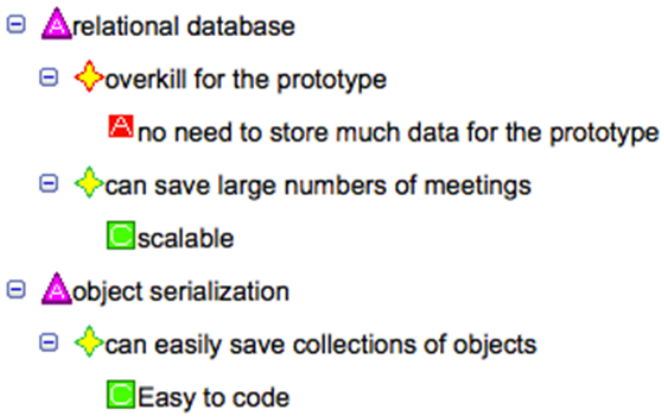

Abbildung 3.6: Entscheidungsbaum in SEURAT_Edu [16] als Beispiel für eine Decision detail Ansicht

Decision-Chronology Ansicht
In Kruchtens ADD Ontology Tool [13] lassen sich auf einer Zeitachse Erstellung und Zustandsänderung von Entscheidungen darstellen.

Decision-Stakeholder-involvement Ansicht
Bei ADDSS [18] werden den Nutzern bei der Registrierung verschiedene Rollen zugeteilt. Für jede Rolle lassen sich der Inhalt und die Visualisierung an die Bedürfnisse der jeweiligen Nutzer anpassen.

Decision-Detail Ansicht
Im Knowledge Architect Knowledge Explorer lassen sich Details zu dem jeweils gewählten Element in der Visualisierung der Beziehungen zwischen einzelnen Elementen anzeigen (siehe Abbildung 3.4). In anderen Tools, wie zum Beispiel ADDSS [18] oder SEURAT_Edu [16], werden die Informationen in tabellarischer Form angezeigt (siehe auch Abbildung 3.6).

Evaluation
Zur Evaluation der verschiedenen Tools wurden industrielle Fallstudien oder kontrollierte Experimente durchgeführt.

Van Heesch *et al.* [12] zeigen in ihrer Arbeit, dass sich für die Visualisierung von Entscheidungswissen Ansichten, aufbauend auf dem ISO/IEC/IEEE42010-Standard, sehr gut eignen, um die Dokumentation und Wiederverwendung von Entscheidungen in Softwareentwicklungsprozessen zu unterstützen.

In ihrer Studie [15] zeigt sich, dass dieses Framework sich sehr gut dazu eignet, bereits festgehaltenes Wissen zu finden und in Entscheidungsprozesse einzubeziehen. Allerdings zeigte sich auch, dass es nicht gut dazu geeignet ist, dieses zu erfassen, was jedoch auch auf einen mangelnden Schwerpunkt auf diese Aufgaben in der Ausbildung von Softwareentwicklern zurückzuführen ist.

Abschließend lässt sich festhalten, wie auch Abad *et al.* [10] feststellen, dass es zwar verschiedene Ansätze zur Visualisierung von Entscheidungen und Anforderungen gibt, es jedoch noch weiterer Forschung und Entwicklung auf diesem Gebiet bedarf.

3.6 Erkenntnisse für diese Arbeit

Zusammenfassend lässt sich festhalten, dass jeder der verschiedenen Ansätze zur Dokumentation von Entscheidungen Vor- und Nachteile hat. So lassen sich einige Funktionen und Ansichten auf das ConDec JIRA Plug-In übertragen, einige sind in der vorhandenen Implementierung bereits umgesetzt.

So erfüllt die Detailansicht für Entscheidungswissen, das als JIRA-Vorgang gespeichert wird, die Kriterien der Decision-Detail-Ansicht. Die Visualisierung mit Hilfe des Treant-Frameworks erfüllt zwar die Anforderungen für eine Decision-Relationship-Ansicht, jedoch wird aufgrund der ungerichteten Verknüpfungen in dieser die Nachverfolgbarkeit nicht unterstützt. Daher soll für die Visualisierung eine Visualisierungsbibliothek so gewählt werden, dass Verknüpfungen zwischen einzelnen Elementen gerichtet dargestellt und mit einer Beschriftung versehen werden können.

Es gibt auch noch keine Möglichkeit, durch Filter oder Suchen den Umfang des dargestellten Wissens einzuschränken und so leichter verständlich zu machen. Hier ist es möglich, entweder auf die in JIRA integrierte Suchfunktionalität zurückzugreifen oder die für Entscheidungswissen und dessen Visualisierung nötigen Filter direkt in der Visualisierungsumgebung anzubieten.

4 Anforderungen

In diesem Kapitel werden die Anforderungen für die neue Visualisierung von Entscheidungswissen im ConDec Plug-In beschrieben. Diese Anforderungen umfassen Beschreibungen von Anwendern in Form verschiedener Personae. Für die Aufgaben, bei denen die Nutzer unterstützt werden sollen, werden User-Tasks (UT) und zugehörige Sub-Tasks (ST) definiert. Aus diesen werden die Systemfunktionen (SF) abgeleitet, mit denen das System die Nutzer in ihren Aufgaben unterstützt. Des Weiteren werden messbare nichtfunktionale Anforderungen formuliert.

Die Anforderungen basieren auf der Ausschreibung der Arbeit und den Erkenntnissen der Literaturrecherche. Sie wurden während der Entwicklung durch Erfahrungen mit ersten Prototypen und den Möglichkeiten der verwendeten Frameworks erweitert.

4.1 Personae

Personae helfen durch die Beschreibung eines fiktiven Nutzers Anforderungen an die zu entwickelnde Software zu definieren. Tabelle 4.1 und 4.2 beschreiben zwei Nutzer in verschiedenen Rollen innerhalb eines Softwareprojekts. Neben einer Biographie werden auch Wissen, Bedürfnisse, Wünsche und Frustrationen der Nutzer beschrieben. Aus diesen Beschreibungen werden die typischen Arbeitsabläufe, die die Grundlage für die folgenden User- und Sub-Tasks bilden, abgeleitet.

Tabelle 4.1: Persona-Beschreibung Jon Doe

Name	**Jon Doe**
Job	Software Entwickler
Biographie	32 Jahre alt. Bachelorabschluss in angewandter Informatik. Arbeitet für eine mittelständische Softwarefirma als Full-Stack-Entwickler. Wird in der Firma je nach Bedarf für verschiedene Projekte eingesetzt.
Wissen	Benutzt meistens Java und Javascript, beherrscht jedoch auch PHP und Python. Hat Erfahrung mit der Verwendung von ITS und Versionskontrollsystemen.
Bedürfnisse	Möchte einen schnellen Überblick, welche Entscheidungen in einem Projekt seit seiner letzten Mitarbeit getroffen wurden.
Frustrationen	Lange Wartezeiten auf Fragen, welche Entscheidungen weshalb getroffen wurden. Unübersichtliche Visualisierungen von Zusammenhängen.
Wünsche	Möchte einen Überblick über Entscheidungen, die für seine Aufgaben relevant sind erhalten.

Tabelle 4.2: Persona-Beschreibung Jane Doe

Name	**Jane Doe**
Job	Anforderungsingenieurin
Biographie	48 Jahre alt. Diplom-Informatikerin mit mehrjähriger Berufserfahrung. Arbeitet in einer weltweit führenden Softwarefirma als Anforderungsingenieurin und ist in dieser Rolle auch für das Rationale Management verantwortlich.
Wissen	Verwendet verschiedene ITS. Kenntnisse in verschiedenen Standards und bewährten Methoden des Anforderungsmanagements.
Bedürfnisse	Möchte einen Überblick über verschiedene Lösungsmöglichkeiten und deren Vor- und Nachteile erhalten.
Frustrationen	Nicht dokumentierte Entscheidungen führen zu erhöhtem Arbeitsaufwand und damit Kostensteigerung für den Kunden.
Wünsche	Möchte einen Überblick, wie sich Entscheidungen gegenseitig beeinflussen.

4.2 Funktionale Anforderungen

In diesem Bereich werden alle Anforderungen an die Funktionalität der Visualisierung und Verwaltung von Entscheidungswissen formuliert. Der User-Task beschreibt die aus den Personabeschreibungen abgeleitete Hauptaufgabe der Nutzer.

4.2.1 Anwenderaufgaben

4.2.1.1 UT1 - Entscheidungsdokumentation

In Softwareprojekten werden Anforderungen erhoben und dann von Entwicklern implementiert. Während dieser Prozesse werden Entscheidungen getroffen oder Wissen über bereits getroffene Entscheidungen benötigt.

4.2.1.2 ST1.1 - Anforderungen betrachten

Entwickler betrachten während ihrer Arbeit Anforderungen für die zu entwickelnde Software. Dabei können sie sich zu den Anforderungen relevante Entscheidungen als Baum oder Graph anzeigen lassen und diese während der Implementierung berücksichtigen.

4.2.1.3 ST1.2 - Eigene Entscheidungen dokumentieren

Entwickler setzen die Anforderungen in ihrem implementierten Code um. Bei der Umsetzung getroffene Entscheidungen werden im Zusammenhang mit der jeweils bearbeiteten Entwicklungsaufgabe dokumentiert.

4.2.1.4 UT2 - Rationale Management

Rationale Management befasst sich mit der Dokumentation von Entscheidungen und ihren Zusammenhängen. Rationale Manager sind für die Korrektheit und Konsistenz des dokumentierten Entscheidungswissens verantwortlich.

4.2.1.5 ST2.1 - Entscheidungswissen verwalten

Das dokumentierte Entscheidungswissen muss regelmäßig auf seine Qualität und Aktualität geprüft werden. So wird sichergestellt, dass alle Verknüpfungen zwischen einzelnen Dokumenten korrekt und aktuell sind. Auch muss geprüft werden, dass die Dokumentation mit dem Quelltext übereinstimmt und zum Beispiel keine veralteten Entscheidungen im ITS vorhanden sind.

4.2.2 Systemfunktionen

Die Systemfunktionen(SF) in diesem Abschnitt beschreiben die Aufgaben des Systems, die nötig sind, um die Anforderungen umzusetzen. Zu jeder Systemfunktion werden Vor- und Nachbedingungen, Ein- und Ausgaben sowie Ausnahmen und Regeln definiert. Ein Element steht für ein Dokumentationsartefakt. Dies können zum Beispiel JIRA-Issues sein, in denen Entscheidungswissen oder Anforderungen dokumentiert sind, aber auch klassifiziertes Entscheidungswissen in Issue-Kommentaren.

4.2.2.1 Entscheidungswissen und verknüpfte Elemente als Graph anzeigen

Als Nutzer möchte ich mir alle von einem Element erreichbaren anderen Elemente und ihre Zusammenhänge in einem zusammenhängenden gerichteten Graph anzeigen lassen.

Tabelle 4.3: SF1 - Entscheidungswissen und verknüpfte Elemente als Graph anzeigen

Zugehöriger User-Task	ST1.1 - Anforderungen betrachten
Vorbedingung	Element existiert
Input	Wurzelelement
Nachbedingung	Graph wird angezeigt
Ausgabe	Ein gerichteter Graph, der alle vom gewählten Wurzelelement durch Verknüpfungen erreichbaren Elemente als Knoten und die Verknüpfungen als gerichtete Kanten enthält.
Ausnahmen	keine
Regeln	keine

4.2.2.2 Zwischen Baum- und Graphdarstellung umschalten

Als Nutzer kann ich entscheiden, in welcher Form das Entscheidungswissen und weitere Dokumente visualisiert werden sollen.

Tabelle 4.4: SF2 - Zwischen Baum- und Graphdarstellung umschalten

Zugehöriger User-Task	ST1.1 - Anforderungen betrachten
Vorbedingung	Wurzelelement und verknüpfte Elemente sind als Baum oder Graph visualisiert.
Input	Wurzelelement
Nachbedingung	Graph/Baum wird angezeigt
Ausgabe	Wurzelelement und verknüpfte Elemente werden in der jeweils anderen Darstellungsform visualisiert
Ausnahmen	keine
Regeln	keine

4.2.2.3 Elemente durch Filtern aus der Visualisierung entfernen

Als Nutzer kann ich durch Festlegen von Kriterien, die die Elemente erfüllen sollen, Elemente aus der Visualisierung ausschließen und so die Größe des Baums beziehungsweise Graphen reduzieren um eine übersichtliche Darstellung von relevantem Wissen erhalten.

Tabelle 4.5: SF3 - Elemente durch Filtern aus der Visualisierung entfernen

Zugehöriger User-Task	ST1.1 - Anforderungen betrachten
Vorbedingung	Element existiert
Input	Filterkriterien
Nachbedingung	Darstellung ohne Elemente, die nicht den Filterkriterien entsprechen, wird angezeigt
Ausgabe	Verknüpfte Elemente, die den Filterkriterien entsprechen werde,n als Baum oder Graph dargestellt.
Ausnahmen	keine
Regeln	Elemente, die nicht den Filterkriterien entsprechen, werden im Graph als Punkt dargestellt und im Baum durch transitive Verknüpfungen überbrückt.

4.2.2.4 Elemente miteinander verknüpfen

Als Nutzer kann ich verschiedene Elemente miteinander verknüpfen, um so eine vollständige Dokumentation von Entscheidungswissen sicherzustellen.

Tabelle 4.6: SF4 - Elemente miteinander Verknüpfen

Zugehöriger User-Task	ST2.1 - Entscheidungswissen verwalten
Vorbedingung	Zwei Elemente, die nicht miteinander verknüpft sind, existieren
Input	Zwei zu verknüpfende Elemente
Nachbedingung	Beide Elemente sind miteinander verknüpft
Ausgabe	Die Verknüpfung wird in der Visualisierung angezeigt. Im Graph wird einen neue Kante erzeugt, im Baum eine Kopie eines Elements unterhalb des anderen angezeigt.
Ausnahmen	keine
Regeln	Elemente können nicht mit sich selbst verknüpft werden

4.2.2.5 Verknüpfung von Elementen löschen

Als Nutzer kann ich die Verknüpfung zweier miteinander verknüpfte Elemente lösen, um so eine vollständige Dokumentation von Entscheidungswissen sicherzustellen.

Tabelle 4.7: SF5 - Verknüpfung von Elementen löschen

Zugehöriger User-Task	ST2.1 - Entscheidungswissen verwalten
Vorbedingung	Zwei miteinander verknüpfte Elemente existieren
Input	Die zu lösende Verknüpfung
Nachbedingung	Die Elemente sind nicht mehr miteinander verknüpft.
Ausgabe	Die Verknüpfung wird in der Visualisierung entfernt. Im Graph wird die Kante, im Baum das Element unterhalb des anderen gelöscht.
Ausnahmen	keine
Regeln	keine

4.2.2.6 Neues verknüpftes Element erstellen

Als Nutzer kann ich von einem JIRA-Issue ausgehend ein neues Element erstellen, welches direkt mit dem JIRA-Issue verbunden ist. Dies ist zum Beispiel hilfreich, um schnell verschiedene Lösungsvorschläge zu einem Problem zu erstellen.

Tabelle 4.8: SF6 - Neues verknüpftes Element erstellen

Zugehöriger User-Task	ST1.2 - Eigene Entscheidungen dokumentieren ST2.1 - Entscheidungswissen verwalten
Vorbedingung	JIRA-Issue existiert
Input	Daten des neuen Elements
Nachbedingung	Neues Element existiert und ist mit dem JIRA-Issue verknüpft
Ausgabe	Das neue Element sowie die Verknüpfung zum JIRA-Issue werden in der Visualisierung angezeigt.
Ausnahmen	keine
Regeln	keine

4.2.2.7 Element bearbeiten

Als Nutzer kann ich bereits erstellte Elemente bearbeiten. So können Fehler oder Änderungen an Elementen korrigiert werden, ohne ein zusätzliches Element zu erstellen.

Tabelle 4.9: SF7 - Element bearbeiten

Zugehöriger User-Task	ST2.1 - Entscheidungswissen verwalten
Vorbedingung	Element existiert
Input	Aktualisierte Daten
Nachbedingung	Daten des Elements sind angepasst
Ausgabe	Die Änderungen werden in der Visualisierung angezeigt.
Ausnahmen	keine
Regeln	keine

4.2.2.8 Element löschen

Als Nutzer kann ich existierende Elemente löschen. Dies ist hilfreich, da durch das Entfernen nicht mehr relevanter Elemente die Übersichtlichkeit der Visualisierung verbessert werden kann.

Tabelle 4.10: SF8 - Element löschen

Zugehöriger User-Task	ST2.1 - Entscheidungswissen verwalten
Vorbedingung	Element existiert
Input	Das zu löschende Element
Nachbedingung	Element und Verknüpfungen zu/von diesem Element sind gelöscht
Ausgabe	Das Element wird nicht mehr in der Visualisierung angezeigt.
Ausnahmen	keine
Regeln	keine

4.2.3 Domänendaten

Das Domänendatendiagramm in Abbildung 4.1 beschreibt, welche Daten in einem Graph repräsentiert werden. Der Graph besteht aus Knoten, welche in der Darstellung Anforderungen, Aufgaben oder Entscheidungswissen repräsentieren. Alle Informationen, die in einem Knoten dargestellt werden sollen, sind in diesem gespeichert. Alle Kanten im Graph stehen für eine Verbindung zwischen den Daten.

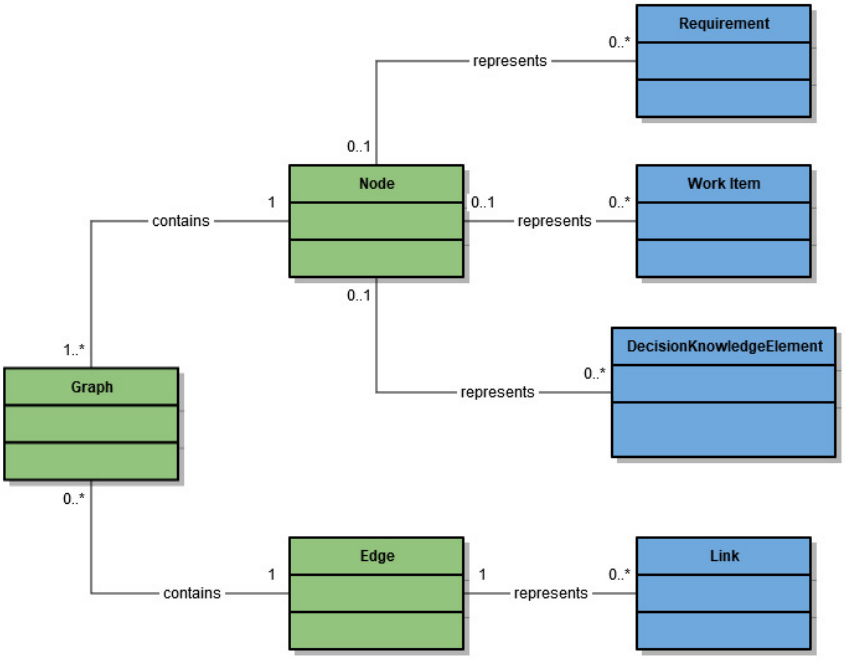

Abbildung 4.1: Domänendaten für eine Graphdarstellung

4.3 Nichtfunktionale Anforderungen

Nichtfunktionale Anforderungen (NFR) werden zum Beispiel im Standard ISO/IEC 25010:2011[1] definiert. NFR definieren messbare Qualitätsmerkmale von Software in Bezug auf ihre Nutzbarkeit, Wartbarkeit, Erweiterbarkeit und Sicherheit. Tabelle 4.11 gibt einen Überblick über die NFR, die vom Prototypen erfüllt werden sollen.

Tabelle 4.11: Übersicht der NFR für die Visualisierung von Entscheidungswissen*
*Merkmale und Beschreibungen wurden aus ISO/IEC 25010:2011 übernommen

Merkmal	Beschreibung	Qualitätsmaß
Funktionale Eignung	Grad, zu dem ein Produkt oder System Funktionen bereitstellt, die angegebene und implizierte Anforderungen erfüllen, wenn sie unter angegebenen Bedingungen verwendet werden.	Systemtests und Evaluation des Prototypen verlaufen erfolgreich
Leistungsfähigkeit	Die Leistung, die im Verhältnis zur Menge der unter den angegebenen Bedingungen eingesetzten Ressourcen erreicht wird.	Ladezeiten sollten unabhängig von der gewählten Visualisierung konstant sein.
Nutzbarkeit	Grad, bis zu dem ein Produkt oder System von bestimmten Benutzern verwendet werden kann, um bestimmte Ziele mit Effektivität, Effizienz und Zufriedenheit in einem bestimmten Nutzungskontext zu erreichen.	Evaluation des Prototyps
Wartbarkeit	Wirkungsgrad und Effizienz, mit dem ein Produkt oder System von den vorgesehenen Betreuern modifiziert werden kann.	Eine möglichst hohe Testfallabdeckung für neue Java Klassen und das Analysieren von Metriken

[1] https://www.iso.org/standard/35733.html Abschnitt 4.2 Product Quality Model

4.4 UI-Strukturdiagramm

Da UI-Strukturdiagramm in Abbildung 4.2 beschreibt die für diese Arbeit benötigten Arbeitsbereiche (Work-Spaces (WS)). In jedem Arbeitsbereich werden die benötigten Daten und die zur Verfügung gestellten Systemfunktionen beschrieben.

Da die in dieser Arbeit entwickelten Funktionen und Ansichten in das ITS JIRA integriert sind, bildet die Ansicht den äußeren Arbeitsbereich. Einzelne Vorgänge werden in JIRA in der Issue View (WS 1.1) angezeigt. Innerhalb dieser Ansicht werden für diese Arbeit zum einen die in JIRA integrierte Suchfunktion (WS1.1.1), zum anderen das ConDec Issue Module (WS1.1.2) benötigt. Im ConDec Issue Module sind die Ansichten für die Baumdarstellung (WS 1.1.2.1) und die Graphendarstellung (WS1.1.2.2) vorhanden. Das Wechseln zwischen diesen Ansichten wird von der Systemfunktion SF2 zur Verfügung gestellt.

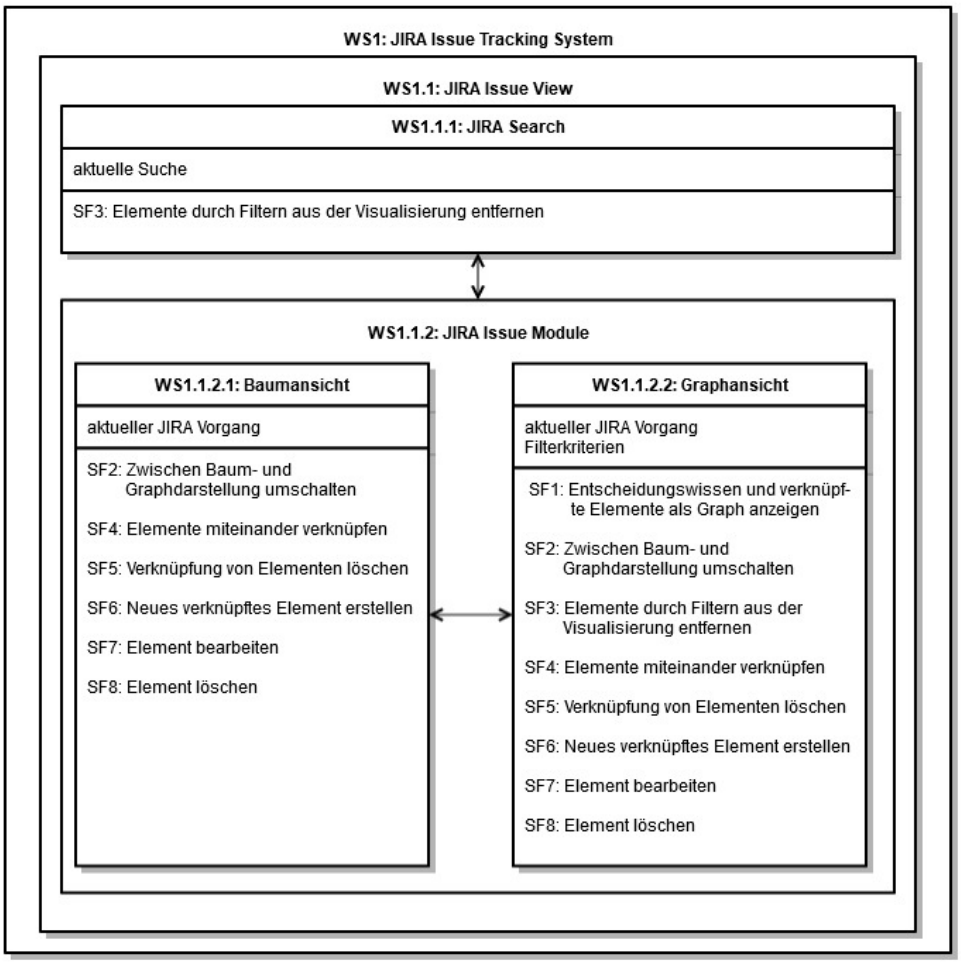

Abbildung 4.2: UI-Strukturdiagramm mit allen relevanten Arbeitsbereichen

5 Entwurf und Implementierung

In diesem Kapitel werden der Entwurf und die Implementierung der Anforderungen aus dem vorherigen Kapitel beschrieben. Zunächst werden die wichtigsten Bestandteile der bisherigen ConDec JIRA Plug-In Implementierung genannt. Dann werden Entscheidungen während des Entwurfs und der Implementierung beschrieben und die Ergebnisse präsentiert.

5.1 Bisherige Implementierung des ConDec JIRA Plug-Ins

Die für diese Arbeit wichtigsten Bestandteile der ConDec JIRA Plug-In Implementierung sind zum einen die REST-Schnittstellen. Diese bieten die Möglichkeit Daten zwischen Client und Server auszutauschen. Es gibt bereits REST-Schnittstellen zum Erstellen, Bearbeiten und Löschen von Elementen.

Auch gibt es Modellklassen für Entscheidungsdokumente und Verknüpfungen. Die Klasse *DecisionKnowledgeElement* ist hilfreich, da sie eine Möglichkeit darstellt Methoden und Attribute für alle verschiedenen Dokumentationsorte zu vereinheitlichen. Das gleiche gilt für die Klasse *Link*, da so Elemente unabhängig von ihrem Dokumentationsort miteinander verknüpft werden können.

5.2 Entwurf

In diesem Abschnitt werden zuerst wichtige Entscheidungen für die Implementierung der neuen Visualisierung und der Filtermöglichkeiten dargestellt. Danach werden Entscheidungen für die Umsetzung der einzelnen Systemfunktionen beschrieben.

5.2.1 Architekturentscheidungen

Da JIRA eine Web-Anwendung ist, die von den Nutzern in ihrem Browser genutzt werden kann, wird das vorhandene Apache-Velocity-Template erweitert. Neue Funktionalitäten für die Client-Seite werden in JavaScript implementiert. Die Entwicklung auf der Server-Seite wird in Java realisiert. Die Kommunikation zwischen Client und Server wird durch Erweiterung der REST-Schnittstellen umgesetzt.

Tabelle 5.1: Übersicht über Architekturentscheidungen

Entscheidungsproblem	**Entscheidung**
Welche Programmiersprachen kommen zum Einsatz?	Auf Serverseite wir Java verwendet! Auf Client Seite werden JavaScript, HTML und CSS verwendet!
Wie werden Daten zwischen Client und Server übertragen?	Durch eine Erweiterung der bestehenden REST-Schnittstelle

5.2.2 Entscheidungen zu Systemfunktionen

5.2.2.1 Entscheidung zu SF1 - Entscheidungswissen und verknüpfte Elemente als Graph anzeigen

Um das Entscheidungswissen als Graph zu visualisieren wird eine JavaScript-Bibliothek benötigt. Als Bibliothek für die Visualisierung wird vis.js[1] verwendet. Die Daten hierfür sollen auf der Server-Seite erstellt werden. Dabei soll die bestehende Graph-Klasse verwendet werden. Abbildung 5.1 gibt einen Überblick, welche Klassen auf der Serve- Seite implementiert werden sollen, um die Daten für die Visualisierung mit Vis zu realisieren.

Tabelle 5.2: Entscheidung zu SF1

Problem	Welche Bibliothek soll für die Visualisierung als Graph verwendet werden?
Entscheidung	Verwende die vis.js Bibliothek!
Vorteile	Open-Source unter MIT-Lizenz Sehr gut konfigurierbar Daten können auf Server Seite als JSON erstellt werden
Nachteile	Konfiguration sehr verschachtelt, kompliziert, diese auf der Server Seite umzusetzen
Alternativen	Es gibt viele weitere Bibliotheken zur Darstellung von Graphen

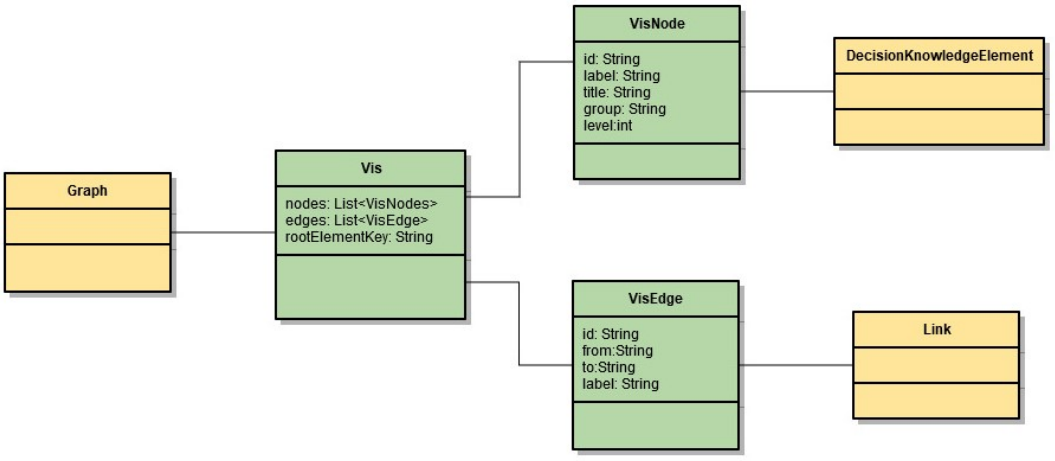

Abbildung 5.1: Entwurfsklassendiagramm für den Vis-Graphen

[1] http://visjs.org/

5.2.2.2 Entscheidung zu SF2 - Zwischen Baum- un Graphdarstellung umschalten

Da die Baumdarstellung erhalten bleiben soll, muss es eine Möglichkeit geben, zwischen beiden Darstellungen zu wechseln. Hierfür gibt es zwei Möglichkeiten: Entweder jeder Nutzer kann selbst entscheiden, welche Darstellung er bevorzugt, oder es wird in den Einstellungen für das Projekt eine Darstellung festgelegt. Für die Variante, dem Nutzer die Entscheidung zu überlassen spricht, dass jede der beiden Darstellungen je nach Situation Vorteile bietet und so jeder Nutzer für sich selbst entscheiden kann, welche Ansicht ihm besser gefällt. Die Einstellung soll persistiert werden, so dass beim Aufruf eines anderen Elements die gewählte Ansicht erhalten bleibt.

Tabelle 5.3: Entscheidung zu SF2

Problem	Wie soll zwischen den beiden Darstellungen umgeschaltet werden?
Entscheidung	Es wird ein Reiter zum Wechseln der Ansicht benutzt!
Vorteile	Jeder Nutzer kann nach seinen Vorlieben entscheiden. Je nach Element kann eine der Ansichten besser geeignet sein
Nachteile	Die Ladezeit verlängert sich minimal.
Alternativen	In den Einstellungen des Projekts kann die Visualisierung gewählt werden.

5.2.2.3 Entscheidung zu SF3 - Elemente durch Filtern aus der Visualisierung entfernen

Das Filtern von Elementen in der Visualisierung ist eine wichtige Funktionalität, um die Übersichtlichkeit der Visualisierung zu verbessern. Eine erste Entscheidung, die getroffen werden muss, ist, welche Filter zur Verfügung gestellt werden sollen. Für JIRA-Vorgänge bietet sich die in JIRA integrierte Such- und Filterfunktion an. Ein Vorteil diese Ansatzes ist, dass Nutzer Suchanfragen, die sie häufig benötigen, als eigene Filter speichern können. Um diese Filter auch auf Elemente anwenden zu können, die nicht als JIRA-Vorgänge gespeichert sind, müssen die entsprechenden Kriterien aus der Suchanfrage extrahiert und auf die jeweiligen Elemente angewendet werden.

Da es eine sehr große Anzahl verschiedener Kriterien gibt, wurde entschieden, Elemente, die nicht als JIRA-Vorgänge gespeichert sind, nur nach ihrem Vorgangstyp und dem Datum ihrer Erstellung zu filtern. Eine weitere Entscheidung ist, in der Graphenansicht eine weitere Filtermöglichkeit anzubieten, mit der neben dem Vorgangstyp und dem Erstellungsdatum auch der Dokumentationsort eingeschränkt werden kann.

Die Entscheidung, wie Elemente, die nicht den Filterkriterien entsprechen, dargestellt werden sollen unterscheidet sich zwischen der Baum- und Graphdarstellung. In der Baumdarstellung werden transitive Kanten erstellt, die die Knoten, die nicht den Filterkriterien entsprechen, überbrücken. In der Graphdarstellung werden die Knoten als kleine Punkte visualisiert, da sich so die Zusammenhänge besser überprüfen lassen. Eine Besonderheit in der Graphdarstellung ist das Filtern nach Entfernung vom Wurzelelement. Elemente, die diesem Kriterium nicht entsprechen, werden in einem gemeinsamen Knoten dargestellt, um die Größe des Graphen zu reduzieren.

Der Filter soll auf der Server-Seite überprüfen, ob ein Element den Kriterien entspricht, und soll genutzt werden, um zu einem bestimmten Element verknüpfte Elemente, die den Filterkriterien entsprechen, zu finden und für die Visualisierung bereitzustellen Das Entwurfsklassendiagramm ist in Abbildung 5.2 zu sehen.

Tabelle 5.4: Entscheidung zu SF3 - Verwendung des in JIRA eingebauten Filters

Problem	Wie sollen dem Nutzer die Filtermöglichkeiten angeboten werden?
Entscheidung	Benutze die in JIRA integrierte Such- und Filterfunktion!
Vorteile	Der Nutzer kennt diese Funktion bereits. Vom Nutzer können Suchanfragen als Filter gespeichert und bei Bedarf abgerufen werden. Elemente, die als JIRA-Vorgänge gespeichert sind, lassen sich sehr vielseitig filtern.
Nachteile	Für Elemente, die nicht als JIRA-Vorgang gespeichert sind, müssen alle Bedingungen einzeln überprüft werden.
Alternativen	Es kann ein eigenes Filtermenü erstellt werden.

Tabelle 5.5: Entscheidung zu SF3 - Filter in der Graphdarstellung

Problem	Soll es weitere Möglichkeiten geben Elemente zu filtern?
Entscheidung	Ja, füge eine Filterfunktion zur Graphansicht hinzu und mache sie mit den bereits eingestellten Filtern konsistent!
Vorteile	Es kann schnell nach den Kriterien, die auf alle Elemente anwendbar sind, gefiltert werden. Es kann zusätzlich nach Dokumentationsort und maximaler Entfernung gefiltert werden, was durch die in JIRA integrierten Filter nicht möglich ist.
Nachteile	Elemente im Graph sind nicht mit im Vorgangsnavigator angezeigten Vorgängen konsistent.
Alternativen	Filtern nur über die in JIRA eingebaute Möglichkeit.

Tabelle 5.6: Entscheidung zu SF3 - Gefilterte Ansicht im Baum

Problem	Wie sollen Elemente, die nicht den Filterkriterien entsprechen, im Baum dargestellt werden?
Entscheidung	Gar nicht. Elemente werden durch transitive Verknüpfungen verbunden!
Vorteile	Der Baum wird kleiner und übersichtlicher.
Nachteile	Es wird angezeigt, dass es einen Zusammenhang zwischen Elementen gibt, aber nicht, ob andere Elemente dazwischen liegen.
Alternativen	Elemente verkleinert anzeigen.

Tabelle 5.7: Entscheidung zu SF3 - Gefilterte Ansicht im Graph

Problem	Wie sollen Elemente, die nicht den Filterkriterien entsprechen, im Graph dargestellt werden?
Entscheidung	Elemente werden als kleine Punkte im Graph angezeigt!
Vorteile	Der Graph wird übersichtlicher. Es ist einfach nachzuvollziehen, wie Elemente miteinander zusammenhängen.
Nachteile	Die Anzahl an Knoten und Kanten im Graph ändert sich nicht.
Alternativen	Transitive Verknüpfungen wie in der Baumansicht nutzen.

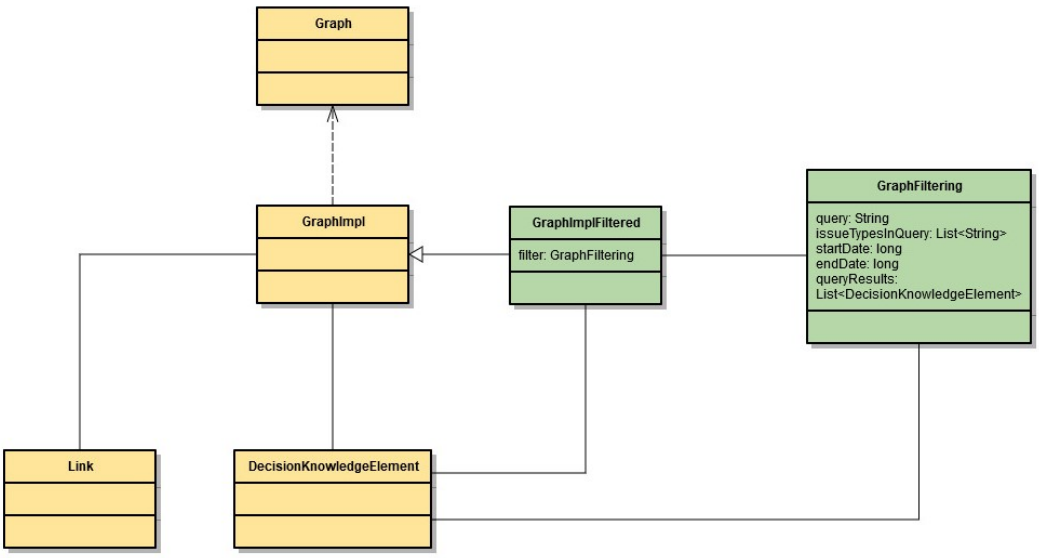

Abbildung 5.2: Entwurfsklassendiagramm für die Filterfunktionalität

5.2.2.4 Entscheidung zu SF4 - Elemente miteinander verknüpfen

Das Verknüpfen von Elementen ist derzeit über einen Dialog, der im Kontextmenü geöffnet wird, möglich. Dieses soll beibehalten werden. Die Möglichkeit, im Baum Elemente per Drag-and-Drop zu verknüpfen, ist in vis.js nicht mehr möglich, da der Nutzer dort die Knoten in der Ansicht verschieben kann. Daher soll zusätzlich die in vis.js vorhandene Funktion zum Hinzufügen neuer Kanten verwendet werden.

Tabelle 5.8: Entscheidung zu SF4

Problem	Wie sollen neue Kanten hinzugefügt werden?
Entscheidung	Es wird die in vis.js verwendete Funktion zum Hizufügen neuer Kanten verwendet!
Vorteile	Kanten können direkt in der Visualisierung, ohne Benutzung des Kontextmenüs, hinzugefügt werden.
Nachteile	Es sind zwei Mausklicks nötig, um das Hinzufügen von Kanten zu aktivieren.
Alternativen	Kanten können nur über das Kontextmenü hinzugefügt werden.

5.2.2.5 Entscheidung zu SF5 - Verknüpfung von Elementen löschen

Verknüpfungen sollen auch in der Visualisierung gelöscht werden können. In der Baumdarstellung ist dies über den Eintrag „Link zu übergeordnetem Element löschen“ möglich. Dies ist im Graph nicht mehr möglich, da Elemente mehr als eine eingehende Kante besitzen können. In vis.js gibt es die Möglichkeit, ausgewählte Elemente zu löschen. Diese wird genutzt, um vom Nutzer gewählte Kanten zu löschen. Bevor eine Kante gelöscht wird, wird der Nutzer über eine Warnung informiert und muss den Löschvorgang bestätigen.

Tabelle 5.9: Entscheidung zu SF5

Problem	Wie sollen Kanten im Graph gelöscht werden?
Entscheidung	Es wird die in vis.js verwendete Funktion zum Löschen von Kanten verwendet!
Vorteile	Einfacher als durch das Kontextmenü einen Dialog zu öffnen, um bei mehreren eingehenden Kanten die richtige zu löschen
Nachteile	Es sind vier Mausklicks nötig, um eine Kante zu löschen.
Alternativen	Über das Kontextmenü lassen sich alle eingehenden Kanten löschen.

5.2.2.6 Entscheidung zu SF6 - Neues verknüpftes Element erstellen

Es soll möglich sein, in der Graphdarstellung neue Knoten hinzuzufügen. Dies ist in der Baumdarstellung über einen entsprechenden Kontextmenüeintrag möglich. Diese Möglichkeit wird beibehalten, da im Graph nur zusammenhängendes Wissen angezeigt wird und ein neu erstellter Knoten ohne Verknüpfung zwar erstellt werden könnte, aber nicht in der Visualisierung erscheint.

Tabelle 5.10: Entscheidung zu SF6

Problem	Wie sollen neue Knoten hinzugefügt werden?
Entscheidung	Es wird wie in der Baumdarstellung die Möglichkeit über das Kontextmenü verwendet!
Vorteile	Die Funktion ist den Nutzern schon bekannt.
Nachteile	Inkonsistenz zum Hinzufügen neuer Kanten.
Alternativen	Knoten lassen sich über die vis.js-Funktion hinzufügen und werden ohne Verbindung zu Elementen in der Visualisierung angezeigt.

5.2.2.7 Entscheidung zu SF7 - Element bearbeiten

Zum Bearbeiten von Elementen ist in ConDec bereits ein Dialog vorhanden. Dieser wird wiederverwendet, kann aber nicht nur über das Kontextmenü, sondern auch über einen langen Klick auf einen Knoten geöffnet werden.

Tabelle 5.11: Entscheidung zu SF7

Problem	Wie sollen Knoten bearbeitet werden?
Entscheidung	Benutzt wird der vorhandene Dialog, dieser lässt sich jedoch auch über einen langen Klick auf einen Knoten öffnen!
Vorteile	Der Dialog ist dem Nutzer schon bekannt. Das Öffnen des Dialogs per langem Mausklick geht schneller als über das Kontextmenü.
Nachteile	Der Dialog könnte aus Versehen geöffnet werden.
Alternativen	Das Öffnen des Dialogs ist weiterhin nur über das Kontextmenü möglich.

5.2.2.8 Entscheidung zu SF8 - Element löschen

Elemente lassen sich bisher über das Kontextmenü löschen. Neben dieser Möglichkeit, die erhalten bleiben soll, wird auch die Möglichkeit hinzugefügt, Elemente über die von vis.js zur Verfügung gestellte Funktion Knoten zu löschen. Der Nutzer muss das Löschen eines Elements in einem Dialog bestätigen.

Tabelle 5.12: Entscheidung zu SF8

Problem	Wie sollen Knoten gelöscht werden?
Entscheidung	Es wird die in vis.js vorhandene Funktion zum Löschen von Knoten genutzt!
Vorteile	Konsistenz zum Löschen von Kanten.
Nachteile	Es sind mehr Mausklicks als beim Löschen über das Kontextmenü nötig.
Alternativen	Das Löschen von Elementen ist weiterhin nur über das Kontextmenü möglich.

5.3 Implementierung

In diesem Abschnitt zeigen Bildschirmaufnahmen der beiden Visualisierungen die Ergebnisse der Implementierung.

Die Abbildungen 5.3 und 5.4 zeigen die Darstellung ausgehend vom Element „How can we detect homologous points on stereo images?“ .Es ist gut zu erkennen, dass die Richtung einer Verknüpfung im Graph berücksichtigt wird, im Baum jedoch nicht. Die Anordnung in Hierarchien im Graph bleibt unabhängig vom gewählten Ausgangselement konstant, während das Ausgangselement im Baum immer als Wurzel angezeigt wird. Im Graph erkennt man auch die Beschriftung der Kanten, mit denen der Typ der Verknüpfung angegeben wird.

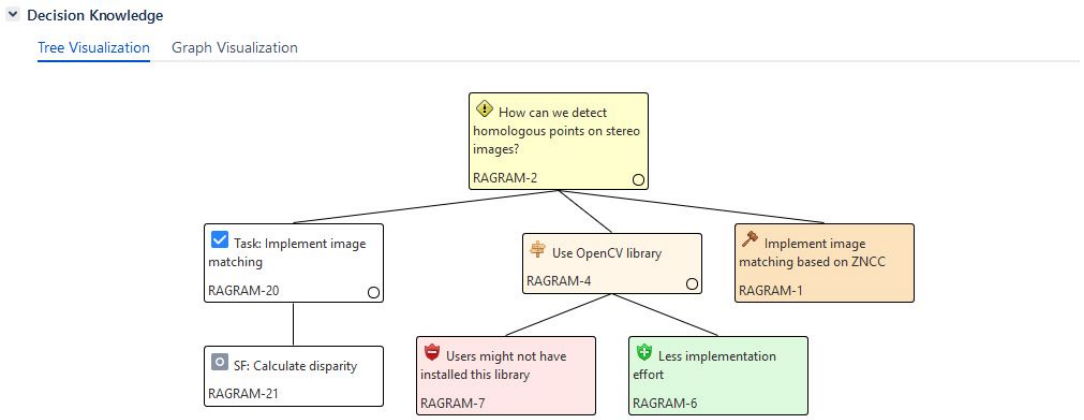

Abbildung 5.3: Die bisherige Visualisierung in Baumform

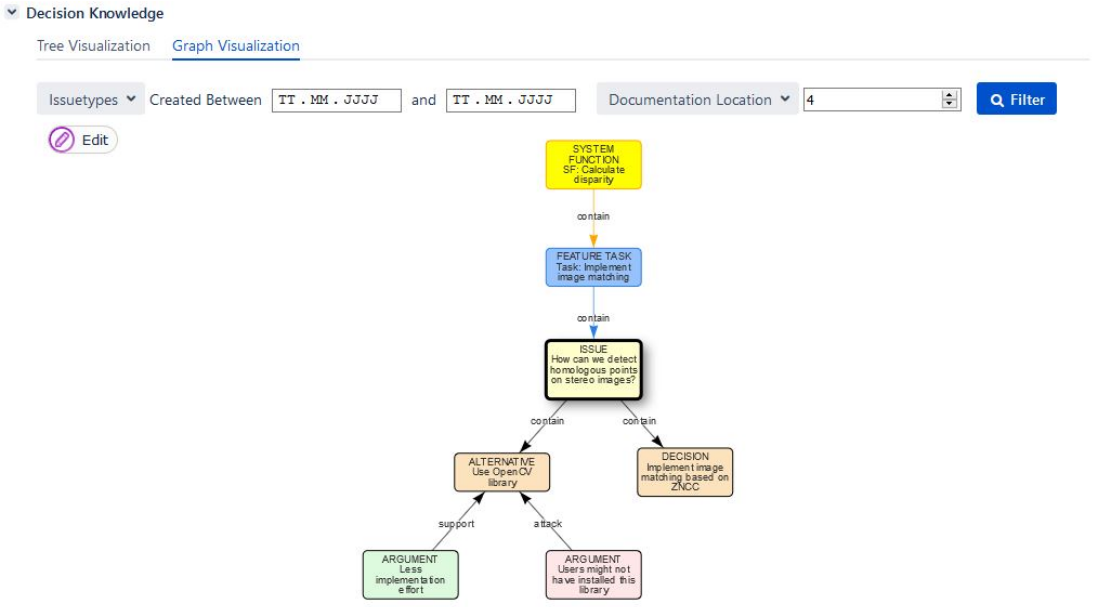

Abbildung 5.4: Die neue Visualisierung in Graphenform

Abbildungen 5.5 und 5.6 zeigen die Visualisierungen für das gleiche Ausgangselement wie zuvor, jedoch sind einige Elemente gefiltert. Man erkennt, dass die transitiven Verknüpfungen es im Baum unmöglich machen zu erkennen, ob und wie viele Elemente durch solch eine transitive Verknüpfung ersetzt werden. Im Graph werden die Elemente, die nicht den Filterkriterien entsprechen, als kleine Punkte dargestellt und erlauben so eine Nachvollziehbarkeit der Zusammenhänge. In der Baumdarstellung werden auch die beiden Argumente nicht angezeigt, da bei transitiver Verknüpfung nicht klar ist, ob sie sich direkt auf das Element beziehen, zu dem sie gerade verknüpft sind, oder zu einem anderen.

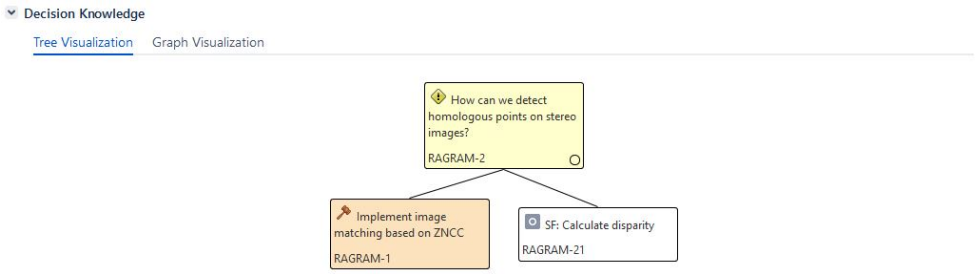

Abbildung 5.5: Anwendung von Filtern für Vorgangstypen in der Baumdarstellung

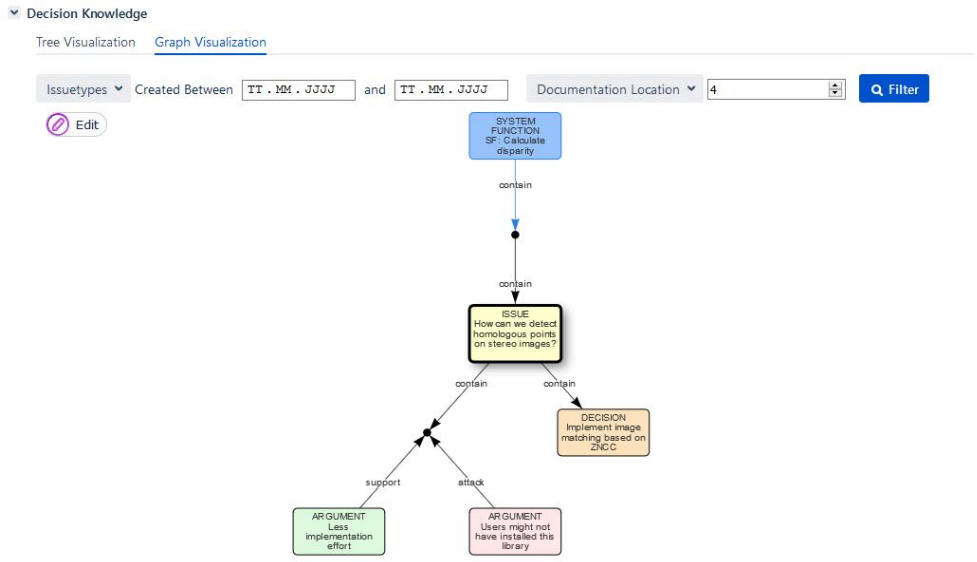

Abbildung 5.6: Anwendung der selben Filter in der Graphdarstellung

Abbildung 5.7 zeigt das Auswahlmenü für Vorgangstypen der in der Graphdarstellung eingebauten Filtermöglichkeit. In diesem Fall wurden im JIRA-Filter bestimmte Vorgangstypen ausgewählt. Beim Laden der Filterleiste in der Graphdarstellung werden diese mit der vorher getroffenen Auswahl konsistent gehalten.

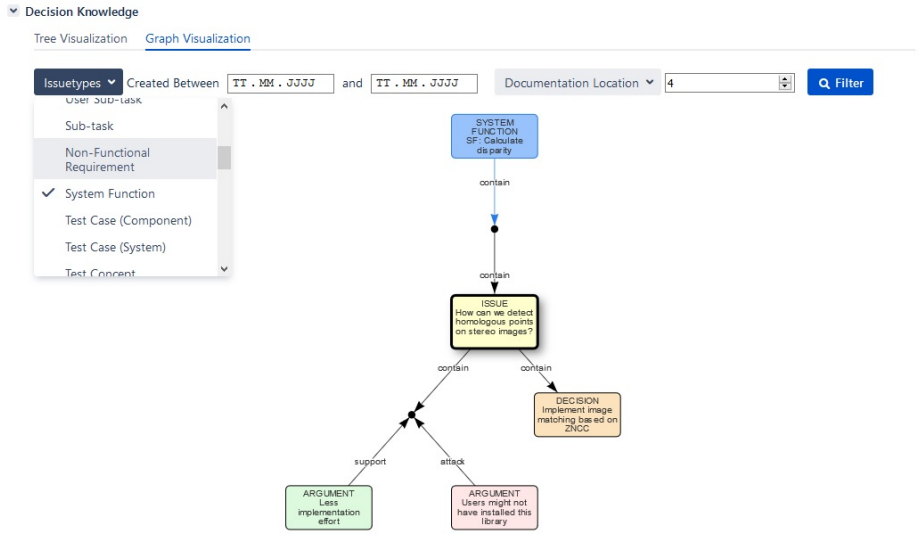

Abbildung 5.7: Konsistenz zwischen JIRA-Filtern und der Möglichkeit in der Graphdarstellung zu filtern

Abbildung 5.8 zeigt, dass für Knoten, die im Graph nur als Punkt dargestellt werden Informationen per Mouseover eingeblendet werden können.

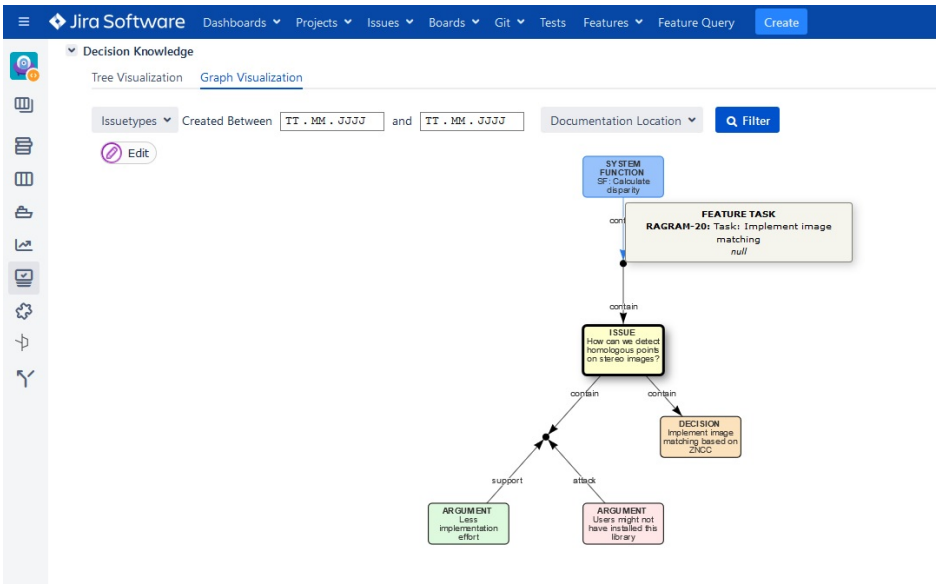

Abbildung 5.8: Informationen zu gefilterten Knoten lassen sich einblenden

5.3.1 Klassendiagramm

Abbildung 5.9 zeigt eine Übersicht über die neu implementierten Klassen (grün), ihre Attribute und öffentlichen Methoden und die Beziehungen zu existierenden Klassen im ConDec Projekt (gelb) sowie Klassen der JIRA-API (blau). Für eine bessere Übersicht wurden in bereits bestehenden Klassen die Attributte und Methoden nicht angegeben.

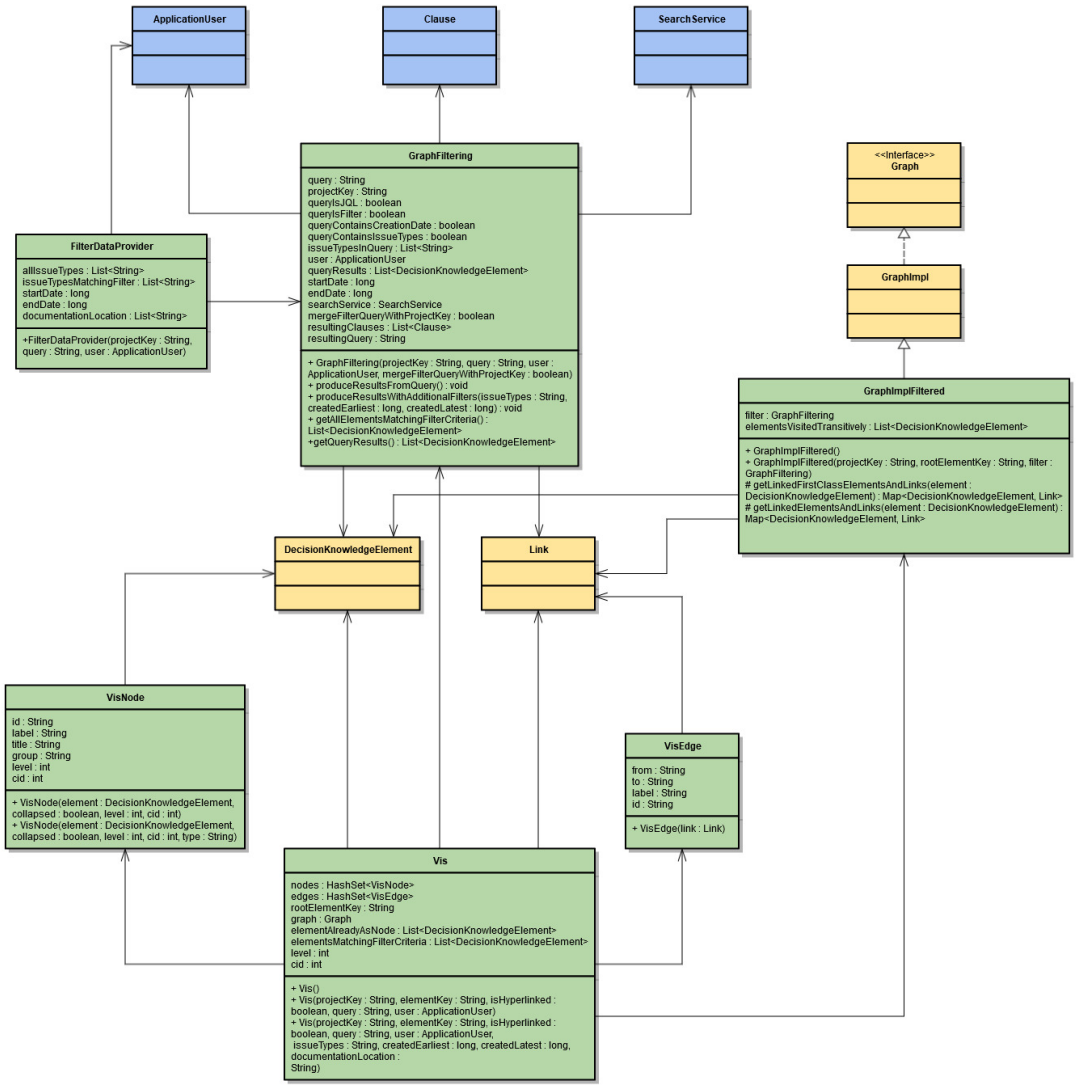

Abbildung 5.9: Klassendiagramm der neu implementierten Klassen

6 Qualitätssicherung

Dieses Kapitel beschreibt Maßnahmen zur Qualitätssicherung des entwickelten Prototypen. Es werden Komponententests beschrieben, um die korrekte Funktion einzelner Module zu überprüfen. Systemtests werden genutzt, um sicherzustellen, dass der Prototyp die Anforderungen der Systemfunktionen und die nichtfunktionale Anforderung an die funktionale Eignung erfüllt. Um die Leistungsfähigkeit zu testen werden, die Ladezeiten für Graphen unterschiedlicher Größe gemessen und mit den Ladezeiten der gleichen Daten in Baumform verglichen. Mit Hilfe statischer Codeanalyse wird der Code auf potentielle Fehler und Schwachstellen hin untersucht.

6.1 Komponententests

Komponententests dienen dazu, die korrekte Funktion einzelner Komponenten zu testen. Um Komponententests zu erstellen, werden gültige und ungültige Äquivalenzklassen-formuliert. Komponententests werden in junit[1] implementiert und unter anderem nach einem Commit zu GitHub ausgeführt. Da für Methoden, die Funktionen aus externen Bibliotheken, in diesem Fall das JIRA-API verwenden, das Verhalten dieser Funktionen für die Durchführung von Komponententests simuliert werden muss, werden nur Komponenten getestet, die keine Funktionen des JIRA-API verwenden.

6.2 Systemtests

Mit Systemtests wird das Verhalten des Prototypen innerhalb des Systems getestet. Die Schritte, die in Tabelle 6.1 beschrieben sind, werden durchgeführt und es wird überprüft, ob das System die erwartete Ausgabe erzeugt. Ist dies der Fall, gilt der Test als bestanden. Die Systemtest wurden auf dem CURES-Server[2] in einem eigenen Projekt durchgeführt.

[1] https://junit.org/junit5/
[2] https://cures.ifi.uni-heidelberg.de/jira

Tabelle 6.1: Übersicht über Systemtests

Nr	Eingabe	Erwartete Ausgabe	Anforderungen	Erfolgreich
1	Elemente werden als Graph oder Baum im JIRA Issue Module (WS1.1.2) angezeigt. Mit Hilfe des Reiters wird zwischen den Ansichten umgeschaltet	Es wird die jeweils andere Ansicht mit den gleichen Daten angezeigt	SF1, SF2	Ja
2	Ein anderer JIRA-Vorgang wird ausgewählt und im JIRA Issue View (WS1.1) angezeigt.	Der Vorgang wird in der vorher gewählten Darstellungsform angezeigt.	SF2	Ja
3	In der JIRA Search (WS1.1.1) werden Filterkriterien festgelegt und das Wissen als Baum dargestellt.	Elemente, die nicht den Filterkriterien entsprechen, werden im Baum nicht mehr dargestellt. Transitive Verknüpfungen werden hinzugefügt.	SF3	Ja
4	In der JIRA Search (WS1.1.1) werden Filterkriterien festgelegt und das Wissen als Graph dargestellt.	Elemente, die nicht den Filterkriterien entsprechen, werden im Graph als Punkte dargestellt.	SF3	Ja
5	In der Graphansicht (WS1.1.2.2) werden Filterkriterien festgelegt und mit dem Filterbutton bestätigt.	Elemente, die nicht den Filterkriterien entsprechen, werden im Graph als Punkte dargestellt.	SF3	Ja
6	In der Graphansicht (WS1.1.2.2) wird das Kontextmenü mit einem Rechtsklick auf einen Knoten geöffnet. Es wird der Punkt „Existierendes Element verlinken“ ausgewählt und aus dem geöffneten Dialogfeld ein Element ausgewählt und bestätigt.	Die Verknüpfung zwischen den Elementen wurde hinzugefügt und wird im Graph angezeigt	SF4	Ja

Nr	Eingabe	Erwartete Ausgabe	Anforderungen	Erfolgreich
7	In der Graphansicht (WS1.1.2.2) wird der Bearbeitungsmodus in der oberen linken Ecke aktiviert und der „Add Edge“ ausgewählt. Danach wird mit gedrückter Maustaste eine Kante von einem Knoten zu einem anderen gezogen.	Die Verknüpfung zwischen den Elementen wurde hinzugefügt und wird im Graph angezeigt	SF4	Ja
8	Eine Kante wird in der Graphansicht (WS1.1.2.2) mit einem linken Mausklick ausgewählt. Danach wird im Bearbeitungsmodus „delete selected“ ausgewählt und die Warnung bestätigt.	Die Verknüpfung wurde gelöscht und wird nicht mehr im Graph angezeigt.	SF5	Ja
9	In der Graphansicht (WS1.1.2.2) wird das Kontextmenü mit einem Rechtsklick auf einen Knoten geöffnet. Es wird der Punkt „Neues Element erstellen“ ausgewählt und im geöffneten Dialogfeld werden Daten für das neue Element eingetragen.	Das neue Element ist erstellt und wird im Graph angezeigt. Eine Verknüpfung zu dem Element, auf dem das Kontextmenü geöffnet wurde, ist erstellt und wird angezeigt.	SF6	Ja
10	In der Graphansicht (WS1.1.2.2) wird das Kontextmenü mit einem Rechtsklick auf einen Knoten geöffnet. Es wird der Punkt „Element bearbeiten“ ausgewählt und die Daten werden im geöffneten Dialogfeld bearbeitet.	Die Daten des Elements wurden geändert und die aktualisierten Daten werden im Graph angezeigt.	SF7	Ja
11	In der Graphansicht (WS1.1.2.2) wird der Dialog zum Bearbeiten mit einem langen Mausklick auf einen Knoten geöffnet und die Daten des Elements werden bearbeitet.	Die Daten des Elements wurden geändert und die aktualisierten Daten werden im Graph angezeigt.	SF7	Ja

Nr	Eingabe	Erwartete Ausgabe	Anforderungen	Erfolgreich
12	Ein Knoten wird in der Graphansicht (WS1.1.2.2) mit einem linken Mausklick ausgewählt. Danach wird im Bearbeitungsmodus „delete selected“ ausgewählt und die Warnung bestätigt.	Das Element ist gelöscht und wird nicht mehr im Graph angezeigt.	SF8	Ja
13	In der Graphansicht (WS1.1.2.2) wird das Kontextmenü mit einem Rechtsklick auf einen Knoten geöffnet. Es wird der Punkt „Element löschen“ ausgewählt.	Das Element ist gelöscht und wird nicht mehr im Graph angezeigt.	SF8	Ja

6.3 Leistungsfähigkeit

Vor allem um der nichtfunktionalen Anforderung Leistungsfähigkeit, aber auch der Nutzbarkeit gerecht zu werden, werden die Ladezeiten der beiden Ansichten gemessen und miteinander verglichen. Es werden sowohl die Wartezeiten auf die Serverantwort, als auch die Zeit vom Auswählen der Visualisierung bis zu ihrer Darstellung gemessen. Die Wartezeit auf die Serverantwort gibt die Zeit an, die der Server braucht um die Daten für die jeweilige Darstellung bereitzustellen. Aus dieser und der Gesamtzeit lassen sich Rückschlüsse auf die Performanz der Serverimplementierung und die Leistungsfähigkeit der verwendeten JavaScript-Bibliotheken für die Visualisierung ziehen. Die Daten werden mit Hilfe der Entwicklertools im Firefox-Browser[3] in dem Projekt, das auch für die Systemtests verwendet wurde, gemessen. Alle Werte sind in Millisekunden angegeben und wurden über fünf Messungen gemittelt.

[3] https://www.mozilla.org/de/firefox/

Tabelle 6.2: Übersicht der Ladezeit für die verschiedenen Darstellungen

Anzahl Elemente	Gesamtzeit	Serverantwort	Gesamtzeit	Serverantwort
	Graph		Baum	
keine Filter angewendet				
1	320	131	300	45
10	560	154	580	108
30	640	372	1400	331
Filter auf Vorgangstypen angewendet				
10	240	143	260	69
30	500	360	650	204

An den ermittelten Werten erkennt man, das die Verarbeitung der Daten auf dem Server für die Baumdarstellung schneller ist. Für die Ladezeit insgesamt ist die Darstellung als Graph bei mehr Elementen und Verknüpfungen unabhängig von angewendeten Filtern schneller. Die Unterschiede sind jedoch gering, so dass beide die Anforderung an Leistungsfähigkeit und Benutzbarkeit erfüllen.

6.4 Statische Codeanalyse

Statische Codeanalyse ist eine weitere Maßnahme, um die Qualität von Software sicherzustellen. Mit Hilfe von Codacy[4], einem in GitHub integrierten Werkzeug, kann geprüft werden, ob der Quelltext die Programmierrichtlinien der jeweiligen Programmiersprache einhält oder es auffällige Metriken gibt.

Eine wichtige Metrik, um die Komplexität einer Klasse darzustellen und damit Aussagen über ihre Wartungs- und Entwicklungsaufwand zu treffen, sind die gewichteten Methoden pro Klasse (MWC) [23]. Diese basieren auf dem Komplexitätsmaß nach McCabe. Die zyklomatische Komplexität basiert auf der Anzahl an Binärverzweigungen. Die MWC berechnet sich aus den Methoden $M_1 ... M_n$ einer Klasse C und ihren zugehörigen Komplexitätswerten $c_1 ... c_n$:

$$WMC = \sum_{i=1}^{n} c_i$$

[4] https://www.codacy.com/

Die in Tabelle 6.3 berechneten Werte für die gewichteten Methoden pro Klasse und der Durchschnitt pro Methode wurden mit dem IntelliJ[5] Plug-In MetricsReloaded[6] berechnet.

Tabelle 6.3: MWC für neue Java-Klassen

Klasse	**MWC**	**∅ MWC**
FilterDataProvider	16	1,45
GraphFiltering	127	3,97
GraphImplFiltered	34	2,83
Vis	36	2,57
VisEdge	9	1
VisNode	14	1,27

Die rot markierten Einträge zeigen zu hohe Werte. Diese führen zu einem schlechter wartbaren Quelltext. Die Werte wurden im Laufe der Entwicklung ständig verbessert, die betreffenden Methoden wegen einer besseren Verständlichkeit und ihrer spezifischen Verwendung jedoch nicht weiter modularisiert.

[5] https://www.jetbrains.com/idea/
[6] https://plugins.jetbrains.com/plugin/93-metricsreloaded

7 Evaluation

Wie in der Einführung schon beschrieben stellten Kleebaum *et al.*[24] fest, dass die Übersichtlichkeit und Filtermöglichkeiten der in der Vorlesung verwendeten Visualisierung von ConDec verbessert werden können. Es wurde evaluiert, ob dies durch die in dieser Arbeit entwickelten Erweiterungen des ConDec JIRA Plug-Ins erreicht werden kann. In Abschnitt 7.1 werden der Aufbau und die Durchührung der Evaluation beschrieben. Die Ergebnisse werden in Abschnitt 7.2 zusammengefasst.

7.1 Aufbau und Durchführung der Evaluation

Um Aussagen über die Nutzung von Softwaresystemen treffen zu können, wird das von Davis *et al.* entwickelte Technology Acceptance Model (TAM)[25] verwendet. In diesem wird postuliert, dass die Bereitschaft, Technologien zu benutzen, von drei wesentlichen Faktoren abhängt:

- **Benutzerfreundlichkeit:** Die Benutzerfreundlichkeit beschreibt, wie einfach es ist, die neu entwickelte Visualisierung als Graph und die in ihr integrierten Funktionen zu verwenden.

- **Nützlichkeit:** Die Nützlichkeit beschreibt, ob Graphdarstellung und Filtermöglichkeiten den Nutzer bei seinen Aufgaben unterstützen.

- **Zukünftiges Nutzungsverhalten**: Das zukünftige Nutzungsverhalten beschreibt, ob Nutzer die Visualisierung als Graph und die Anwendung von Filtern auf die Visualisierung auch in Zukunft verwenden werden.

Um die in dieser Arbeit beschriebene Erweiterung des ConDec JIRA Plug-In auf diese Punkte zu überprüfen, wurden in der Evaluation die Fragen aus Tabelle 7.1 gestellt.

Fragen zur Benutzerfreundlichkeit sind daran zu erkennen, dass sie danach fragen, wie einfach eine bestimmte Funktion zu benutzen ist *(EF1, EF3, EF4, EF5)*. Fragen zur

Nützlichkeit sind daran zu erkennen, dass sie fragen, ob eine bestimmte Funktion für den Benutzer nützlich ist *(EF2,EF6,EF7,EF8,EF9)*. Die abschließende Frage *EF10* bezieht sich auf das zukünftige Nutzungsverhalten.

Die Teilnehmer der Evaluation beantworten die Fragen auf einer fünfstufigen Likert-Skala zwischen starkem Widerspruch und starker Zustimmung. Zusätzlich gibt es die Möglichkeit, die Antwort durch Freitext zu begründen.

Tabelle 7.1: Evaluationsfragen

ID	Evaluationsfrage
EF1	Es ist **einfach** zwischen der Graph- und der Baumdarstellung zu wechseln.
EF2	Der erzeugte Graph **nutzt** dem Erkennen von Zusammenhängen zwischen einzelnen Dokumentationselementen.
EF3	Es ist **einfach**, Elemente und Verknüpfungen im Graph zu bearbeiten, hinzuzufügen und zu löschen.
EF4	Es ist **einfach**, die in JIRA eingebauten Filter auf die Visualisierung anzuwenden.*
EF5	Es ist **einfach**, die Filter innerhalb der Graphdarstellung anzuwenden.*
EF6	Das Filtern von Elementen in der Baumdarstellung **nutzt** dem leichteren Erkennen von Zusammenhängen.
EF7	Das Filtern von Elementen in der Graphdarstellung **nutzt** dem leichteren Erkennen von Zusammenhängen.
EF8	Das Filtern von Elementen in der Graphdarstellung **nutzt** der schnellen Identifizierung, welche Entscheidungen zu Anforderungen getroffen wurden.
EF9	Das Finden bereits getroffener Entscheidungen **nutzt** der eigenen Entscheidungsfindung.
EF10	Ich werde in zukünftigen Projekten das Entscheidungswissen als Graph anschauen.

* EF4 bezieht sich auf die bereits in JIRA vorhandene Suchfunktion und Filtermöglichkeiten, EF5 bezieht sich auf die in der Graphansicht auswählbaren Filterkriterien.

Die Evaluation wurde mit acht Studierenden durchgeführt. Die Erfahrung in der Benutzung des ConDec JIRA Plug-Ins variiert zwischen Nutzern ohne Erfahrung und Nutzern mit etwa zwei Jahren Erfahrung. Im Durchschnitt lag die Zeit, seit der das ConDec JIRA Plug-In verwendet wird, bei etwa acht Monaten.

Während der Evaluation sollen die Teilnehmer jeweils eine eigene Aufgabe erstellen und diese mit einer für diese Evaluation in einem eigenen Projekt angelegten SF verknüpfen. Danach dokumentieren die Nutzer Entscheidungen nach dem bekannten Modell. Dabei

verwenden sie die Möglichkeit Entscheidungswissen entweder in den Kommentaren ihrer Aufgabe oder in separaten JIRA-Vorgängen zu erfassen.

Im Anschluss durchsuchen die Teilnehmer das gemeinsam dokumentierte Entscheidungswissen. Dabei werden die in JIRA vorhandenen Filtermöglichkeiten sowohl auf die Baum- als auch auf die Graphdarstellung angewendet. In der Graphdarstellung wird auch die zusätzliche Filtermöglichkeit für Vorgangstypen und Dokumentationsorte verwendet, sowie die maximale Entfernung für angezeigte Knoten variiert. Da alle Elemente innerhalb eines Tages erstellt wurden, war eine vollständige Evaluation des Filterns nach Erstellungsdatum nicht möglich.

Als Letztes füllen die Teilnehmer den Fragebogen aus. Dabei geben sie an ob sie die Aussagen zu Benutzerfreundlichkeit, Nutzbarkeit und zukünftiger Nutzung unterstützen oder nicht. Die Teilnehmer werden auch darum gebeten, Dinge, die ihnen besonders gut gefallen haben, aber auch Kritik und Verbesserungsvorschläge zu vermerken.

7.2 Ergebnisse

In diesem Abschnitt werden die Ergebnisse der Evaluation zusammengefasst. Abbildung 7.1 zeigt eine Übersicht der Antworten auf die Evaluationsfragen. Die Tabellen 7.2 und 7.3 fassen sowohl positive als auch negative Rückmeldungen sowie Verbesserungsvorschläge zusammen.

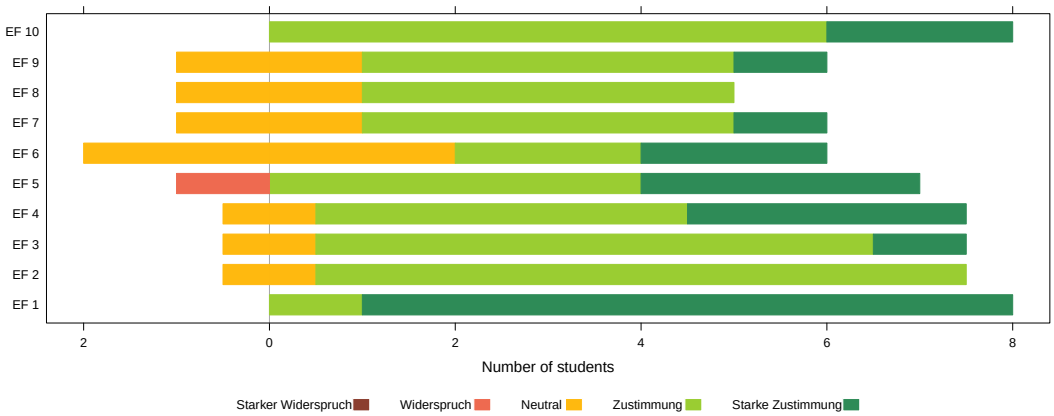

Abbildung 7.1: Ergebnisse der Befragung

Tabelle 7.2: Positives Feedback zu den einzelnen Evaluationsfragen

ID	Feedback
EF1	Das Wechseln zwischen den einzelnen Ansichten ist einfach und intuitiv.
EF2	Zusammenhänge zwischen einzelnen Elementen lassen sich gut erkennen. Das Zoomen und Verschieben des Graphen unterstützt die Übersichtlichkeit.
EF3	Das Bearbeiten und Hinzufügen von Elementen und Verbindungen ist einfach und intuitiv.
EF4	Die Filter werden ohne weiteres Eingreifen des Nutzers auf die Visualisierung angewendet.
EF5	Die Filter sind intuitiv und übersichtlich. Es ist gut, dass nicht die gesamte Seite neu geladen wird.
EF6	Transitive Kanten, die gefilterte Elemente überbrücken, verbessern die Übersichtlichkeit.
EF7	Das Filtern von Elementen in der Graphdarstellung **nutzt** dem leichteren Erkennen von Zusammenhängen.
EF8	Durch Filter lassen sich nur relevante Elemente anzeigen.
EF9	Ein Überblick aller anderen Entscheidungen ist hilfreich.
EF10	Nach erster Benutzung finde ich es besser benutzbar als die Baumansicht.

Tabelle 7.3: Kritik und Verbesserungsvorschläge zu den einzelnen Evaluationsfragen

ID	Feedback
EF1	Die letzte Ansicht des Graphen könnte gespeichert werden, um die Ansicht beim Wechseln konstant zu halten.
EF2	Das Layout des Graphen kann noch verbessert werden.
EF3	Es wäre gut, nach dem Bearbeiten die gleiche Stelle des Graphen zu sehen. Es ist nicht möglich, Kanten zwischen JIRA-Vorgängen und Entscheidungswissen in Kommentaren zu erstellen.
EF4	Nicht alle Filter werden auf Entscheidungswissen in Kommentaren angewendet.
EF5	Das Auswahlmenü für die Elementtypen ist unübersichtlich. Auswahl der Maximaldistanz könnte genauer erklärt werden.
EF6	Durch transitive Kanten können Zusammenhänge zwischen Elementen unklar werden.
EF7	Elemente, die die maximal eingestellte Entfernung übersteigen, sollten ebenso wie gefilterte Elemente als kleiner Knoten angezeigt und nicht in einem gemeinsamen Knoten zusammengefasst werden.
EF8	Eine Filtermöglichkeit für Text wäre gut. Vorgefertigte Filter zum Anzeigen von Entscheidungen zu Anforderungen wären gut.
EF9	Es könnten mehr Kriterien für Entscheidungen visualisiert werden, um die Entscheidungsfindung konsistenter zu machen.
EF10	Der Detailgrad ist etwas zu gering. Die Möglichkeit innerhalb des Graphen zu zoomen, macht ihn unübersichtlich.

7.3 Diskussion

Insgesamt lassen sich die Rückmeldungen als positiv bezeichnen. Die Teilnehmer der Evaluation empfinden die Nutzung von Filtern und die Visualisierung von Entscheidungswissen als Graph als einfach und nützlich. Der hohe Grad der Akzeptanz lässt auf eine hohe Bereitschaft zur Nutzung schließen. Dies stimmt auch mit dem Ergebnis auf die Frage nach der zukünftigen Nutzung überein. Vor allem die positive Rückmeldung auf das Filtern in beiden Ansichten lässt den Schluss zu, dass hierdurch die Bereitschaft ConDec zur Dokumentation von Entscheidungswissen gesteigert werden kann.

Die Tatsache, dass während der Evaluation die Regeln zur Verknüpfung von Elementen nicht eingehalten wurden, ist eine der Ursachen, warum das Layout des visualisierten Graphen nicht ideal war (vergleiche Abbildung 7.2).

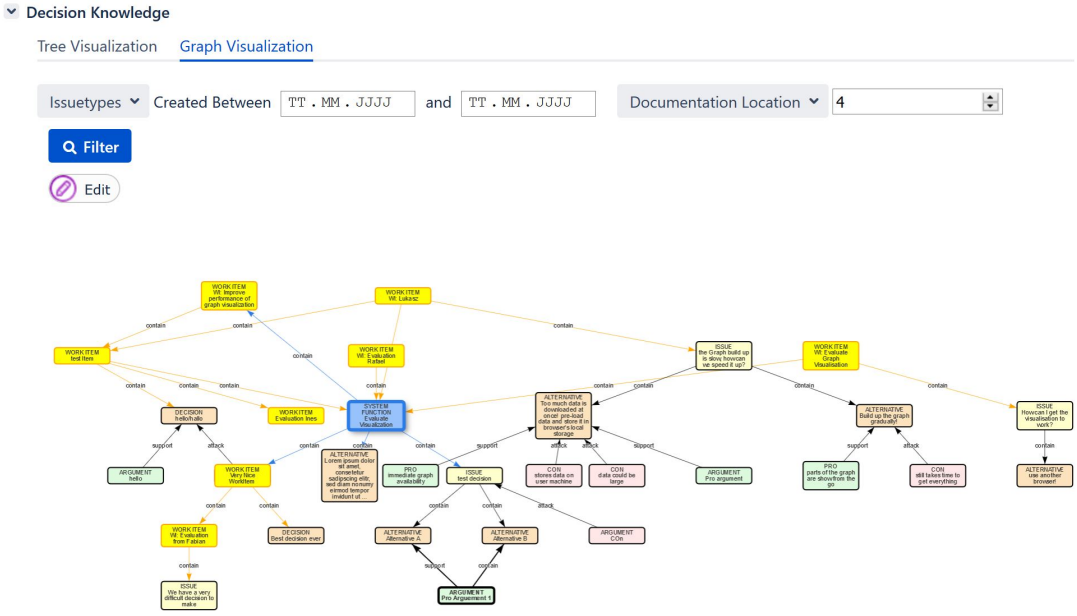

Abbildung 7.2: Graph, der während der Evaluation von den Teilnehmern erstellt wurde

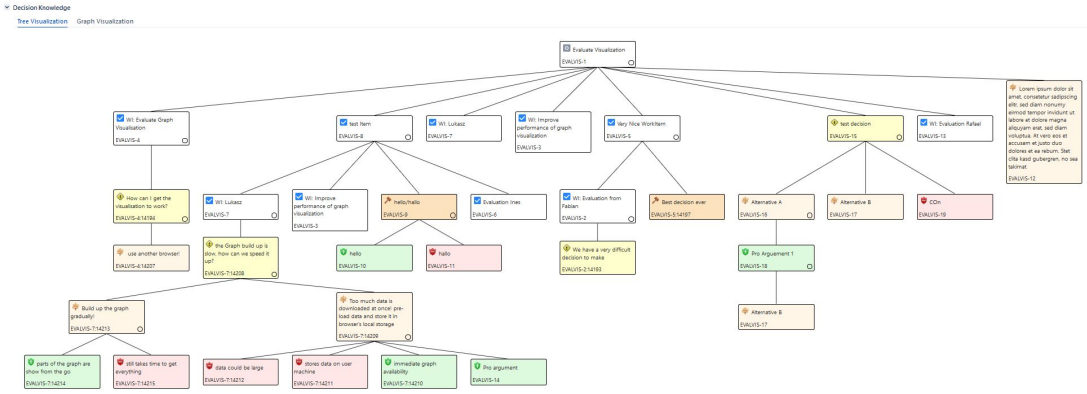

Abbildung 7.3: Graph, der während der Evaluation von den Teilnehmern erstellt wurde

Die Frage, ob beim Filtern nach Entfernungen vom Ursprungselement die Knoten zusammengefasst werden sollen oder wie bei anderen Filterkriterien als Punkt dargestellt werden sollen, lässt sich nicht abschließend beantworten. Der Vorteil, die Knoten in einem Knoten zu vereinen, liegt darin, dass die Größe des Graphen beschränkt wird. Der Nachteil liegt darin, dass dies nicht konsistent zu den anderen Filterkriterien ist.

8 Schlussfolgerung

In diesem Teil der Arbeit werden die wesentlichen Ergebnisse dieser Arbeit zusammengefasst. Anschließend werden Probleme diskutiert und Lösungsansätze sowie Verbesserungsvorschläge beschrieben.

8.1 Zusammenfassung

In dieser Arbeit wurde eine neue Visualisierung von Entscheidungswissen, seine Zusammenhängen und die Beziehungen in Form eines gerichteten Graphen beschrieben. Um einen Überblick über vergleichbare Systeme und ihre Funktionen sowie Ansichten zu erhalten, wurde eine systematische Literaturrecherche durchgeführt. Daraus wurden vor allem die Form als gerichteter Graph und die Filtermöglichkeiten als wichtigste Anforderungen für die Erweiterung des ConDec JIRA Plug-Ins identifiziert. Diese Anforderungen wurden durch Nutzeraufgaben und Systemfunktionen konkretisiert. Dieser Entwurf wurde als Erweiterung des bestehenden ConDec Plug-Ins implementiert. Die Qualität der entstandenen Software wurde gesichert die Nützlichkeit und Benutzerfreundlichkeit mit anderen ConDec-Entwicklern evaluiert.

8.2 Diskussion und Ausblick

Der Prototyp genügt den Anforderungen und stellt Entscheidungswissen, JIRA-Vorgänge und deren Zusammenhänge korrekt dar. Wie auch in der Evaluation angemerkt sollte die Anordnung der Knoten und Kanten im Graph noch verbessert werden. Auch die im Prototyp realisierte Filterfunktion funktioniert korrekt, ist jedoch nicht in der Lage, Elemente, die nicht als JIRA-Vorgänge gespeichert sind, auf alle Kriterien hin zu untersuchen.

Diese Probleme sind zwei Aspekte, die in der Zukunft bearbeitet werden sollten. So ist es zum Beispiel möglich, Positionen von Knoten zu speichern und passende Positionen

für neue Knoten bei der Erstellung des Graphen auf dem Server zu berechnen. Auch sollte die Möglichkeit, Entscheidungswissen, das nicht als JIRA-Vorgang gespeichert ist, zu filtern, über die in dieser Arbeit beschriebenen Kriterien hinaus filterbar sein.

Eine Funktion, um die die Graphdarstellung in Zukunft erweitert werden kann, ist eine Möglichkeit, mit der eine Änderungsauswirkungsanalyse für neue Entscheidungen durchgeführt werden kann. Dies und das Erkennen von widersprüchlichen oder veralteten Entscheidungen können dabei helfen, das Entscheidungswissen aktuell und konsistent zu halten.

Literatur

[1] W. Kunz und H. W. Rittel, *Issues as elements of information systems*. Citeseer, 1970, Bd. 131.

[2] T.-M. Hesse, A. Kuehlwein und T. Roehm, „DecDoc: A Tool for Documenting Design Decisions Collaboratively and Incrementally“, in *Proceedings of the 1st Int. Workshop on Decision Making in Software ARCHitecture*, 2016, S. 30–37. DOI: 10.1109/MARCH.2016.9.

[3] B. Paech, A. Delater und T.-M. Hesse, „Supporting Project Management Through Integrated Management of System and Project Knowledge“, in *Software Project Management in a Changing World*, Springer Berlin Heidelberg, 2014, S. 157–192. DOI: 10.1007/978-3-642-55035-5_7.

[4] A. Kleebaum, J. O. Johanssen, B. Paech, R. Alkadhi und B. Bruegge, „Decision knowledge triggers in continuous software engineering“, in *Proceedings of the 4th International Workshop on Rapid Continuous Software Engineering - RCoSE '18*, ACM Press, 2018. DOI: 10.1145/3194760.3194765.

[5] A. Kleebaum, J. O. Johanssen, B. Paech und B. Bruegge, „Tool Support for Decision and Usage Knowledge in Continuous Software Engineering“, 2018. DOI: 10.11588/heidok.00024186.

[6] H. R. André Krischke, *Graphen und Netzwerktheorie*. Hanser Fachbuchverlag, 6. November 2014, 251 S.

[7] R. A. Burkhard, „Towards a Framework and a Model for Knowledge Visualization: Synergies Between Information and Knowledge Visualization“, in *Lecture Notes in Computer Science*, Springer Berlin Heidelberg, 2005, S. 238–255. DOI: 10.1007/11510154_13.

[8] D. Budgen und P. Brereton, „Performing systematic literature reviews in software engineering“, in *Proceeding of the 28th international conference on Software engineering - ICSE '06*, ACM Press, 2006. DOI: 10.1145/1134285.1134500.

[9] S. Jalali und C. Wohlin, „Systematic literature studies“, in *Proceedings of the ACM-IEEE international symposium on Empirical software engineering and measurement - ESEM '12*, ACM Press, 2012. DOI: 10.1145/2372251.2372257.

[10] Z. S. H. Abad, M. Noaeen und G. Ruhe, „Requirements Engineering Visualization: A Systematic Literature Review“, in *2016 IEEE 24th International Requirements Engineering Conference (RE)*, IEEE, September 2016. DOI: 10.1109/re.2016.61.

[11] M. Shahin, P. Liang und M. R. Khayyambashi, „Improving understandability of architecture design through visualization of architectural design decision“, in *Proceedings of the 2010 ICSE Workshop on Sharing and Reusing Architectural Knowledge - SHARK '10*, ACM Press, 2010. DOI: 10.1145/1833335.1833348.

[12] U. van Heesch, P. Avgeriou und R. Hilliard, „A documentation framework for architecture decisions“, *Journal of Systems and Software*, Bd. 85, Nr. 4, S. 795–820, April 2012. DOI: 10.1016/j.jss.2011.10.017.

[13] L. Lee und P. Kruchten, „A Tool to Visualize Architectural Design Decisions“, in *Quality of Software Architectures. Models and Architectures*, Springer Berlin Heidelberg, 2008, S. 43–54. DOI: 10.1007/978-3-540-87879-7_3.

[14] R. C. de Boer, P. Lago, A. Telea und H. van Vliet, „Ontology-driven visualization of architectural design decisions“, in *2009 Joint Working IEEE/IFIP Conference on Software Architecture & European Conference on Software Architecture*, IEEE, September 2009. DOI: 10.1109/wicsa.2009.5290791.

[15] U. van Heesch, P. Avgeriou und A. Tang, „Does decision documentation help junior designers rationalize their decisions? A comparative multiple-case study“, *Journal of Systems and Software*, Bd. 86, Nr. 6, S. 1545–1565, Juni 2013. DOI: 10.1016/j.jss.2013.01.057.

[16] J. Malloy und J. Burge, „SEURAT_Edu“, in *Proceedings of the 47th ACM Technical Symposium on Computing Science Education - SIGCSE '16*, ACM Press, 2016. DOI: 10.1145/2839509.2844555.

[17] A. Tang, P. Avgeriou, A. Jansen, R. Capilla und M. A. Babar, „A comparative study of architecture knowledge management tools“, *Journal of Systems and Software*, Bd. 83, Nr. 3, S. 352–370, März 2010. DOI: 10.1016/j.jss.2009.08.032.

[18] R. Capilla, F. Nava, S. Pérez und J. C. Dueñas, „A Web-based Tool for Managing Architectural Design Decisions“, *SIGSOFT Softw. Eng. Notes*, Bd. 31, Nr. 5, September 2006. DOI: 10.1145/1163514.1178644.

[19] A. Jansen und J. Bosch, „Software Architecture as a Set of Architectural Design Decisions“, in *5th Working IEEE/IFIP Conference on Software Architecture (WICSA'05)*, IEEE, 2005. DOI: 10.1109/wicsa.2005.61.

[20] A. Tang, Y. Jin und J. Han, „A rationale-based architecture model for design traceability and reasoning“, *Journal of Systems and Software*, Bd. 80, Nr. 6, S. 918–934, Juni 2007. DOI: 10.1016/j.jss.2006.08.040.

[21] P. Liang, A. Jansen und P. Avgeriou, *Knowledge Architect: A Tool Suite for Managing Software Architecture Knowledge*, English. University of Groningen, Johann Bernoulli Institute for Mathematics und Computer Science, 2009, Relation: https://www.rug.nl/informatica/organisatie/overorganisatie/iwi Rights: University of Groningen, Research Institute for Mathematics and Computing Science (IWI).

[22] M. A. Babar und I. Gorton, „A Tool for Managing Software Architecture Knowledge“, in *Second Workshop on Sharing and Reusing Architectural Knowledge - Architecture, Rationale, and Design Intent (SHARK/ADI'07: ICSE Workshops 2007)*, IEEE, Mai 2007. DOI: 10.1109/shark-adi.2007.1.

[23] S. R. Chidamber und C. F. Kemerer, „Towards a metrics suite for object oriented design“, in *Conference proceedings on Object-oriented programming systems, languages, and applications - OOPSLA '91*, ACM Press, 1991. DOI: 10.1145/117954.117970.

[24] A. Kleebaum, J. O. Johanssen, B. Paech und B. Bruegge, „Teaching Rationale Management in Agile Project Courses“, 2019. DOI: 10.11588/heidok.00026358.

[25] F. D. Davis, R. P. Bagozzi und P. R. Warshaw, „User Acceptance of Computer Technology: A Comparison of Two Theoretical Models“, *Management Science*, Bd. 35, Nr. 8, S. 982–1003, August 1989. DOI: 10.1287/mnsc.35.8.982.

Abbildungsverzeichnis

Tabellenverzeichnis

Glossar

JIRA Ein von Atlassian entwickeltes, weit verbreitetes Issue-Tracking-System . 1, 3, 5, 33, 35, 36, 38, 55–57, 61, 62

REST-Schnittstelle Programmierparadigma zur Anfrage von Daten in Web-Anwendungen . 35, 36

Snowballing Methode zur Literatursuche. Dabei wird ausgehend von bekannter Literatur über Zitate und Zitierungen nach weiteren relevanten wissenschaftlichen Arbeiten gesucht . 9