Ruprecht-Karls-Universität
Heidelberg

Institut für Informatik

Lehrstuhl für Software Engineering

Werkzeugunterstützung zur automatisierten Identifikation von gemeinsamen Bestandteilen in User Stories

Bachelorarbeit

Angewandte Informatik

Autor: Maksim Luca Sébastien Schreck

Matrikel-Nr: 3245175

Erstbetreuerin: Prof. Dr. Barbara Paech

Zweitbetreuer: Marcus Seiler

2. November 2018

Eigenständigkeitserklärung

Ich versichere hiermit, dass ich die vorliegende Arbeit selbständig verfasst und keine anderen als die angegebenen Hilfsmittel benutzt habe.
Sowohl inhaltlich als auch wörtlich entnommene Inhalte wurden als solche kenntlich gemacht. Die Arbeit ist in gleicher oder vergleichbarer Form noch bei keiner anderen Prüfungsbehörde eingereicht worden.

Heidelberg, den 2. November 2018

Maksim Schreck

Abstract — Requirements play an important role in the realization of a software project according to customer needs. Crosscutting concerns between these requirements complicate the encapsulation of functionality described by these requirements. If crosscutting concerns stay undiscovered, they later manifest in software that is difficult to maintain and hard to expand. In the scope of this thesis, a literature research is performed to give an overview over existing approaches to identify common components in user stories. These approaches are compared concerning their qualification to be adapted in an automated fashion. Based on predetermined criteria, the most suitable approach is chosen and prototypically implemented as a plugin for the Issue-Tracking-Software Jira. The developed tool is evaluated based on two test-projects. The results show that the ambigous interpretation of textual requirements aggravate the analysis. Badly formulated requirements additionally impair the automatic identification. Hence the developed tool can be used to aid the requirements engineer, however some remaining manual effort to choose the final crosscutting concerns still persists.

Zusammenfassung — Anforderungen sind wichtig um die Umsetzung einer Software gemäß vorgegebener Kundenwünsche zu garantieren. Crosscutting Concerns zwischen diesen Anforderungen erschweren die Kapselung der in den Anforderungen beschriebenen Funktionalität. Bleiben diese Crosscutting Concerns unerkannt, manifestieren sie sich später in schlecht wartbarer und schwer erweiterbarer Software. Im Kontext dieser Arbeit wird eine Literaturrecherche durchgeführt, um einen Überblick über bestehende Ansätze zur Identifikation von gemeinsamen Bestandteilen in User Stories zu erhalten. Diese Ansätze werden auf ihre Eignung zur automatischen Adaption verglichen. Anhand von vorher festgelegten Kriterien wird ein Ansatz ausgewählt und als Plugin für die Vorgangsverfolgungssoftware Jira prototypisch implementiert. Das entwickelte Werkzeug wird anhand von zwei Testprojekten evaluiert. Die Evaluationsergebnisse zeigen, dass die mehrdeutige Auslegung von textuellen Anforderungsdokumenten die Analyse schwierig macht. Zusätzlich beeinträchtigen schlecht formulierte Anforderungen die automatische Identifikation. Somit kann das entwickelte Werkzeug zwar zur Unterstützung des Anforderungsingenieurs genutzt werden, jedoch bleibt ein Restaufwand für den Anforderungsingenieur zur finalen und eindeutigen Auswahl der Crosscutting Concerns bestehen.

Inhaltsverzeichnis

Abkürzungsverzeichnis

AOIG Action-oriented identifier graph. 32, 36

AORE Aspect Oriented Requirements Engineering. 11, 16, 36

AOSD Aspect Oriented Software Development. 11, 33

IDE Integrated development environment. 57

ISO International Organization for Standardization. 7

ITS Issue Tracking Systeme. 7

NFR Non-functional requirement. 12, 29

NLP Natural Language Processing. 13, 25, 28–32, 36–38, 48, 65

POS Part Of Speech. 13, 28, 31, 32, 36, 38, 41, 45, 49

SoC Separation of concerns. 8, 26, 29, 36

SRS Software Requirements Specification. 8

TAFT Tagging Approach to support Feature management. 9, 14

UML Unified Modeling Language. 25, 27

USAS UCREL Semantic Analysis System. 28

WSD word-sense disambiguation. 31, 36

1 Einleitung

Abschnitt 1.1 geht auf die Notwendigkeit und den Nutzen der Identifizierung von gemeinsamen Bestandteilen in User Stories ein. Abschnitt 1.2 beschreibt die weitere Struktur der Arbeit.

1.1 Motivation

Die Definition von Anforderungen stellt auch heute noch für viele Software Unternehmen eine herausfordernde Aufgabe dar. Zu geringes Budget für die Anforderungserhebung und mangelndes Wissen über Anforderungen führen zu Software, die nicht dem Kundenwunsch entspricht sowie zu erhöhten Entwicklungskosten im weiteren Lebenszyklus der Software [32]. Im heutigen agilen Zeitalter setzen viele Unternehmen und vor allem Open Source Projekte auf Issue Tracking Systeme (ITS) und dokumentieren ihre Anforderungen in Form von User Stories.
Wiegers et al. [32] beschreibt den Stellenwert der Anforderungserhebung wie folgt:

> "Of all the software process improvements you could undertake, improved requirements practices are among the most beneficial."[1]

In diesem Zusammenhang spielt die Nachverfolgbarkeit von Softwareartefakten eine entscheidende Rolle im Software-Engineering Prozess.
Laut Gotel et al. [15] beschreibt sie die Fähigkeit, ein Anforderungsdokument über den ganzen Softwareentwicklungszyklus hinweg verfolgen zu können. Sie ermöglicht es, sowohl Design-Entscheidungen nachvollziehen zu können, als auch Auswirkungen von späteren Änderungen korrekt einzuschätzen.
Im Verlauf dieser Arbeit wird der Begriff *Concern* verwendet, um Softwareartefakt in diesem Kontext zu spezifizieren. Es ist wichtig, zwischen den Begrifflichkeiten Concern und Softwareartefakt zu differenzieren. Ein Concern wird von Sutton et al. [30] wie folgt definiert:

> "We will take concern generally to be any matter of interest in a software system."[2]

Nachverfolgbarkeit in sicherheitskritischen Systemen nimmt einen besonders hohen Stellenwert ein, da die Nichteinhaltung teilweise fatale Folgen hat [19]. Der Aufbau dieser Systeme ist durch Standards wie International Organization for Standardization (ISO) 26262 [18] festgelegt. Allerdings steigt bei größerer Komplexität des Systems auch der Aufwand, die Nachverfolgbarkeit über herkömmliche Verlinkungen zu gewährleisten [19].

[1] Karl Wiegers and Joy Beatty, Software requirements, 2013, S.26.

[2] Stanley M. Sutton Jr. and Isabelle Rouvellou, Modeling of Software Concerns in Cosmos, 2002, S.128.

Ein generelles Konzept für eine hohe Nachverfolgbarkeit ist das Prinzip der Separation of concerns (SoC). Hierbei ist das Ziel die Zerlegung des Systems in mehrere Module, um die Komplexität des Gesamtsystems zu verringern [26]. Ein solches Modul, auch Entität [23], realisiert einen Teil der Gesamtmenge der Concerns im System.
Rosenhainer [26] beschreibt weiterhin als *Crosscutting Concern* ein Concern, das Implementierung und Design zweier oder mehrerer Softwarebestandteile schneidet. Somit ist die Modularisierung unter dem Prinzip des separation of concerns bei Crosscutting Concerns nicht trivial möglich. Dies ist damit begründet, dass die Funktionalität nicht in einem Concern gebündelt vorliegt, sondern sich im System verteilt befindet [23]. Außerdem beschreibt Rosenhainer [26] Phänomene in Anforderungsdokumenten, welche auf die Existenz von Crosscutting Concerns schließen lassen. Einzelheiten darüber werden in Abschnitt 2.1 noch näher erläutert.
Die Problematik bei der Etablierung von Nachverfolgbarkeit, die durch die Existenz von Crosscutting Concerns entsteht, ist also die Verteilung von Funktionalität über verschiedene Concerns. Dies macht die Kapselung dieser Funktionalität schwierig.

Die Software Requirements Specification (SRS) (IEEE 830-1998) [14] ist ein Standard, der sich mit Charakteristika guter Anforderungsdokumente befasst. Als erstrebenswerte Eigenschaften werden unter anderem die Eindeutigkeit und die Nachverfolgbarkeit des Anforderungsdokuments angeführt. Die Mehrdeutigkeit eines Anforderungsdokuments ist ein typisches Zeichen, dass hier ein Crosscutting Concern vorliegt, da mehrere Funktionalitäten zusammen beschrieben werden. Diese Mehrdeutigkeit kann durch Ansätze der Crosscutting Concern Identifikation erkannt und behoben werden. Somit ist die Identifikation von Crosscutting Concerns auf Anforderungsebene maßgeblich, um gute Anforderungsdokumente im Sinne dieses Standards zu erhalten.
Weiterhin ist laut Sampaio et al. [28] eine möglichst frühe Identifikation von Crosscutting Concerns sowohl dem Änderungsmanagement als auch der Nachverfolgbarkeit der Anforderungen zuträglich.
Ali et al. [2] argumentieren ebenfalls, dass eine frühe Identifikation auf Anforderungsebene wichtig ist, da sie beispielsweise die Architekturwahl im späteren Verlauf der Entwicklung beeinflusst. Außerdem führt die Vernachlässigung der Identifikation in späteren Entwicklungsphasen beispielsweise zu Duplizierung von Programmcode.

Rosenhainer [26] argumentiert, dass eine hohe Nachverfolgbarkeit im System durch die Identifikation von Crosscutting Concerns auf Anforderungsebene sowohl die Bewertung von Auswirkungen einer Änderung am System, als auch die

Änderung von Anforderungsdokumenten erleichtert. Dadurch kann jeder Faktor des Systems, der von der Änderung betroffen ist, leichter erkannt werden. Findet diese Identifikation nicht statt, führt dies zu nicht erfüllten Anforderungen, die erst spät und im schlimmsten Fall durch Nutzer des Systems aufgedeckt werden können. Dies wiederum verursacht erhöhte Kosten der Behebung, proportional zur verstrichenen Dauer bis zur Aufdeckung. Von monetären Kosten abgesehen, ist es auch dem Ruf der Entwickler abträglich [26].

Allerdings ist die Identifikation von Crosscutting Concerns umso schwerer, je früher im Entwicklungszyklus sie vorgenommen wird. Das ist dadurch begründet, dass Crosscutting Concerns auf späterer Programmcodeebene einfacher zu erkennen sind[2]. Ali et al. [2] argumentieren außerdem, dass die im Folgenden behandelte Identifikation von Crosscutting Concerns auf Anforderungsebene sehr umständlich ist, da zunächst die Gesamtmenge an Anforderungsdokumenten in Betracht gezogen werden muss. Ein weiterer Umstand ist der Mangel an Informationen über das System, die zum Anforderungszeitpunkt vorliegen. Wichtige Designentscheidung sowie Einzelheiten zur Architektur sind noch nicht bekannt und können nicht zur Identifikation herangezogen werden.
Ein weiteres Problem, mit dem sich Entwickler oft konfrontiert sehen, ist Zeitdruck, was sie dazu veranlasst "Abkürzungen" hinsichtlich Strukturierung des Softwareprojekts hinzunehmen und hohen Dokumentationsaufwand zu meiden [28]. Um diesen Nachteil zu beheben und den benötigten Aufwand zu senken , existiert der alternative Tagging Approach to support Feature management (TAFT) [29]. Die Einzelheiten dieses Ansatzes werden in Abschnitt 2.4 noch näher erläutert.

Aus diesen Gründen ist es notwendig, Entwicklern ein Werkzeug an die Hand zu geben, das sowohl Unterstützung bei der Identifikation von Crosscutting Concerns in Anforderungen gibt als auch die Nachverfolgbarkeit über Feature-Tags nach der Identifikation sicherstellt. Diese offensichtlich wichtigen Aufgaben können so in einem möglichst kurzen Zeitrahmen durchgeführt werden.
Ziel der Arbeit ist die Analyse relevanter Literatur, die Auswahl des sinnvollsten Ansatzes anhand von vorher festgelegten Kriterien, die Umsetzung dessen als Jira-Plugin Prototyp (FElICiTy) und die Evaluation des Ansatzes anhand von zwei Fallstudien. Jira wird genutzt, um sämtliche bei Entwurf, Anforderungserhebung und Test eines Softwareprojekts entstehende Artefakte zu administrieren. Dafür ist in Jira die Erstellung einer Reihe von Anforderungsdokumenten möglich, die Design und Struktur des Softwaresystems modellieren. Basierend auf dem gewählten Ansatz werden notwendige Anforderungen erhoben und eine Folgeaktion nach der Identifikation definiert, um oben genannte Nachverfolgbarkeit zu erreichen. Der Entwurf, die Implementierung und die Qualitätssicherung des Prototypen werden

beschrieben und der Prototyp wird anhand von zwei Fallstudien evaluiert.

1.2 Aufbau der Arbeit

Der Rest der Arbeit ist wie folgt aufgebaut: In Kapitel 2 werden die Grundlagen sowie die wichtigen Begriffe im Kontext der Arbeit beschrieben. Kapitel 3 beschreibt das Vorgehen und die Ergebnisse der Literaturrecherche. Kapitel 4 zeigt Vor- und Nachteile der gefundenen Ansätze aus Kapitel 3 auf. Kapitel 5 beschreibt die Anforderungen, den Entwurf, die Implementierung und die Qualitätssicherung des FElICiTy Prototypen. Kapitel 6 evaluiert den FElICiTy Prototypen anhand von zwei Projekten gegen einen selbst erstellten Goldstandard. Die Arbeit schließt mit einer Zusammenfassung und einem Ausblick in Kapitel 7.

2 Grundlagen

Der Abschnitt 2.1 definiert die gemeinsamen Bestandteile in Anforderungsdokumenten. Abschnitt 2.2 stellt den Aufbau und die Verwendung der User Story vor. Die Abschnitte 2.3 und 2.4 geben einen Überblick über technische Aspekte des Themas und konkretisieren für den weiteren Verlauf der Arbeit wichtige Fachtermini. In Abschnitt 2.5 werden die bei der Evaluation verwendeten Metriken definiert.

2.1 Crosscutting Concerns

Ein Crosscutting Concern bezeichnet einen gemeinsamen Bestandteil mehrerer Anforderungsdokumente. Die in der vorliegenden Arbeit behandelten Ansätze zur Identifikation von Crosscutting Concerns sind unter dem Paradigma des Aspect Oriented Software Development (AOSD) zusammenzufassen. Das Hauptziel dieses Ansatzes ist die Identifizierung, Trennung, Darstellung und Zusammenfassung der im System existierenden Crosscutting Concerns [9].
Sampaio et al. [28] führen an, dass eine möglichst frühe Adaption von AOSD im Entwicklungszyklus wichtig ist, um von den Vorteilen maximal profitieren zu können.
Der Teilbereich des oben angesprochenen AOSD, der sich mit der Umsetzung auf Anforderungsebene beschäftigt, ist das Aspect Oriented Requirements Engineering (AORE). Hierbei stehen wiederum Modularisierung und Isolierung der Anforderungen und deren Beziehungen zueinander im Zentrum [21]. Daraus folgt, dass Crosscutting Concerns schon auf Anforderungsebene identifiziert werden müssen.
Außerdem zeigt Rosenhainer [26] zwei Probleme auf, durch welche Crosscutting Concerns entstehen:

- Scattering Problem
- Tangling Problem

Scattering Problem kann frei als *Streuungsproblem* übersetzt werden. Ein Streuungsproblem beschreibt ein Concern, dessen Implementierung im System verteilt ist. Also beispielsweise die Protokoll-Funktion einer Software, die Ereignisse auf gesamter Systembreite aufnehmen muss.
Ein Tangling Problem (*Verwicklungsproblem*) beschreibt hingegen einen Softwarebestandteil, der mehrere Concerns in sich vereint. Hier ist als Beispiel ein Nutzervorgang zu nennen, der durch eine Sicherheitseinschränkung, also beschränkten Zugang, beeinträchtigt wird.
Analog zu dem eingeführten Begriff des Streuungs- und Verwicklungsproblems, machen sich diese Probleme auf Entwicklungebene in zerstreutem und verwickeltem

Programmcode bemerkbar. Auf Anforderungsebene treten sie in den zerstreuten und verwickelten Beschreibungen von Programmfunktionalität zutage [6].
Im Rahmen des Theme/Doc-Ansatzes [6] beschreibt ein *Theme* [5] ein ebensolches Feature. Hierbei unterscheidet man zwischen Themes, welche Grundfunktionalitäten umsetzen und solchen, welche Eigenschaften eines Crosscutting Concerns nach obiger Definition aufweisen. Ein solches crosscutting theme wird auch als *Aspect* bezeichnet [6]. Mithilfe von Aspects ist es möglich, Crosscutting Concerns in einzelne Module zusammenzufassen [9]. Dabei sei zu unterscheiden zwischen nichtfunktionalen Aspects, die durch nichtfunktionale Anforderungen (Non-functional requirement (NFR)) definiert werden und funktionalen Aspects, welche Anforderungen funktionaler Natur umsetzen [6].
Typische funktionale Aspects beinhalten Protokoll, Synchronisierung [26] und Nutzerverwaltung [17], typische nichtfunktionale Aspects Sicherheit [17] und Protokoll [26]. Nichtfunktionale Aspects seien generell leichter zu identifizieren, falls sie explizit in einem Anforderungsdokument erwähnt werden [26]. Auch Jira bietet mit dem Anforderungstyp "Nonfunctional Requirement" diese Funktionalität.

2.2 User Stories

Eine User Story ist eine kurze, zielorientierte Dokumentationsform, Anforderungen darzustellen [12]. Sie beschreibt Funktionalität aus Sicht des Anwenders und folgt nach der *role-feature-reason* Vorlage [1] folgendem Aufbau:

As a — I want — so that —

Ein Beispiel:

As a user, **I want** to search for persons (professors, assistants, secretaries) on the campus **so that** I can view their position on the map.

Mithilfe dieses Aufbaus können die Fragen *wer?*, *was?* und *warum?* im Kontext der Anforderung gezielt beantwortet werden.
In dem angeführten Beispiel ergibt sich somit:
Wer? — *A user.*
Was? — *Search for persons (professors, assistants, secretaries) on the campus.*
Warum? — *So I can view their position on the map.*
Neben dieser am häufigsten verwendeten Vorlage existieren allerdings weitere, welche sich im Aufbau unterschiedlich stark unterscheiden [1]. In einer Variation der gezeigten Vorlage wird die Phrase "so that" durch das äquivalente "in or-

der to" ersetzt, was den Umgang jedoch nicht maßgeblich beeinflusst, also keine Einschränkung bei der Beantwortung der obigen Fragen bringt.
Tauscht man "so that" aus dem obigen Beispiel mit "in order to" erhält man nach geringfügiger Umformung, um grammatikalische Richtigkeit beizubehalten:

As a user, **I want** to search for persons (professors, assistants, secretaries) on the campus **in order to** view their position on the map.

Hier wird ersichtlich, dass die Beantwortung der Fragen ohne Beeinträchtigung, genauso wie oben durchgeführt, erfolgen kann.
In der Praxis kommt es allerdings vor, dass dieser Aufbau einer User Story vernachlässigt oder eine andere Vorlage verwendet wird. Dies ruft eine Reihe von Problemen hervor. Beispielsweise kann so der verwendete Algorithmus das festgelegte Muster nicht erkennen und die Bestandteile der User Stories nicht klar aufteilen. Auf die genauen Auswirkungen wird im weiteren Verlauf der Arbeit in Kapitel 3 und Kapitel 4 eingegangen.

2.3 Analyse von natürlichsprachlichen Anforderungen

User Stories liegen in natürlicher Sprache vor. Bei der Analyse von Texten in natürlicher Sprache liegt es nahe, das Natural Language Processing (NLP) zu verwenden. NLP kann benutzt werden um natürliche Sprache zu interpretieren und diese sowohl semantisch als auch syntaktisch zu verarbeiten [11]. Ein Teilgebiet des NLP ist das Part Of Speech (POS)-Tagging, bei dem ein Eingangsdokument analysiert und jedem Wort seine syntaktische Bedeutung zugewiesen wird.
Die im obigen Beispiel angeführte User Story sieht nach Verarbeitung durch den POS-Tagger wie folgt aus:

As_IN a_DT user_NN ,_, I_PRP want_VBP to_TO search_VB for_IN persons_NNS (_(professors_NNS ,_, assistants_NNS ,_, secretaries_NNS)_) on_IN the_DT campus_NN so_RB that_IN I_PRP can_MD view_VB their_PRP$ position_NN on_IN the_DT map_NN ._.

Bei der Analyse von textuellen Dokumenten steht das Finden von Gemeinsamkeiten auf sprachlicher Ebene im Mittelpunkt. Einige Ansätze verwenden daher den Begriff *action words* [6] und *viewpoints* [28], um relevante und dominante Bestandteile der textuellen Anforderungsdokumente zu beschreiben. Diese Bestandteile bilden dann die Grundlage zur Gliederung der Anforderungsdokumente.

Im Kontext des oben angeführten Beispiels ist ein relevanter Bestandteil der imperative Teil der User Story nach der Phrase “so that” und das dominante Verb “search”.

2.4 Nachverfolgbarkeit über Feature Tags

Im Kontext dieser Arbeit beschreibt ein *Feature* ein Concern auf Anforderungsebene, welches eine Funktionalität spezifiziert.
Um Anforderungsartefakte, die gleiche Funktionalität umsetzen, zu gruppieren, existiert der bereits in Abschnitt 1.1 erwähnte TAFT [29]. Hierbei werden Artefakte, die das gleiche Feature beschreiben, mit demselben *feature tag* versehen. Das feature tag beinhaltet die Bezeichnung des features.
Ein passendes Feature Tag für die oben angeführte User Story ist also beispielsweise *RouteFinding*.

2.5 Metriken

Im weiteren Verlauf der Arbeit wird ein Goldstandard erarbeitet, um die Qualität der von dem Werkzeug erzielten Ergebnisse zu vergleichen. Hierbei wurden manuell alle Crosscutting Concerns zweier Software-Projekte ermittelt. Somit lag ein Soll-Wert für die Menge an Crosscutting Concerns der beiden Systeme vor.
Es wurden die Metriken Precision und Recall verwendet, um die Güte der Ergebnisse beurteilen zu können. Folgende Begriffe müssen definiert werden, um Precision und Recall im Kontext dieser Arbeit zu beschreiben:

- **True Positive** (TP): Menge an Ergebnissen, die korrekterweise erkannt wurde. Im Kontext dieser Arbeit sind dies die vom Werkzeug korrekt erkannten Crosscutting Concerns.
- **False Positive** (FP): Menge an Ergebnissen, die fälschlicherweise erkannt wurde. Im Kontext dieser Arbeit sind dies die vom Werkzeug inkorrekt erkannten Crosscutting Concerns.
- **False Negative** (FN): Menge an Ergebnissen, die korrekt sind, aber nicht erkannt wurden. Im Kontext dieser Arbeit sind dies die laut Goldstandard korrekten Crosscutting Concerns, die jedoch von dem Werkzeug jedoch nicht erkannt wurden.

Somit ergibt sich für die Metriken Precision und Recall:

- **Precision** : $\frac{TP}{TP+FP}$
- **Recall** : $\frac{TP}{TP+FN}$

3 Literaturrecherche

In Abschnitt 3.1 wird die Methodik der durchgeführten Recherche beschrieben. Abschnitt 3.2 geht auf die Ergebnisse und Einzelheiten der Rückwärtssuche ein; Abschnitt 3.3 analog dazu auf die Ergebnisse der Vorwärtssuche. Abschnitt 3.4 listet die Ergebnisse der Recherche auf, Abschnitt 3.5 fast die Recherche zusammen und beantwortet die Fragestellung aus Abschnitt 3.1.1.

3.1 Methodik

Das Ziel der Literaturrecherche wird in Abschnitt 3.1.1 mithilfe der Forschungsfragen präzisiert. Abschnitt 3.1.2 erörtert die Einzelheiten der Durchführung der Recherche.

3.1.1 Forschungsfragen

Im Rahmen der Literaturrecherche soll ein Überblick über bestehende Forschungsbemühungen zum Thema der Arbeit verschafft werden. Das konkrete Ziel der Recherche ist es, einen Katalog über bestehende Ansätze zur Crosscutting Concern Identifikation im Rahmen des AORE zu erarbeiten. Diese Ansätze müssen eine Reihe von Eigenschaften erfüllen, um sich im Kontext dieser Arbeit zur Umsetzung als Jira Plugin zu eignen.
Folgende Forschungsfragen wurden formuliert, um das obige Ziel zu konkretisieren und die Tauglichkeit des Ansatzes zu garantieren:

- Forschungsfrage 1: Welche Ansätze zur automatisierten Identifikation von gemeinsamen Bestandteilen in User Stories existieren?
- Forschungsfrage 2: Welche Aktionen werden im Anschluss an eine automatisierte Identifikation durchgeführt?

Diese Fragen sollen bei der Recherche beantwortet werden. Die gefundenen Ansätze werden anschließend in Kapitel 4 evaluiert. Dabei wird insbesondere der Bestgeeignete zur Adaption und Implementation ausgewählt.

3.1.2 Vorgehen bei der Literaturrecherche

Konform der Richtlinien zur Literaturrecherche [16] wurde ein Katalog an Ausschlusskriterien erarbeitet, um die zur Auswahl stehenden Ergebnisse auf die Artikel einzugrenzen, die einen relevanten Beitrag leisten.
Dieser Katalog mit Ausschlusskriterien ist nachfolgend in Tabelle 1 einsehbar und

wird anschließend näher erläutert.

Ausschlusskriterium	**Beschreibung**
AK1	Der Artikel beschreibt keinen Ansatz zur Identifikation von gemeinsamen Bestandteilen in Anforderungsdokumenten.
AK2	Der im Artikel beschriebene Ansatz ist nicht für eine automatische Adaption geeignet.
AK3	Der im Artikel beschriebene Ansatz erlaubt als Eingabe keine User Stories beziehungsweise textuelle Dokumente.
AK4	Der Artikel ist nicht frei oder durch den Universitätszugang zugänglich.
AK5	Die Glaubwürdigkeit des Artikels ist nicht sichergestellt.
AK6	Der Artikel ist nicht in englischer Sprache verfasst.

Tabelle 1: Ausschlusskriterien

Ausschlusskriterium 1 bescheinigt, dass der Artikel einen Ansatz zur Identifikation von gemeinsamen Bestandteilen in User Stories, wie in RQ1 gefordert, beschreibt.
Ausschlusskriterium 2 garantiert, dass sich der Ansatz für eine automatische Adaption eignet. Im Kontext dieser Arbeit bedeutet dies, dass der Ansatz als JIRA-Plugin und mit den in diesem Zusammenhang verfügbaren Ressourcen umgesetzt werden kann. Ein Ansatz wurde nach Ausschlusskriterium 2 ausgeschlossen, wenn die in ihm beschriebene manuelle Arbeit durch den Anforderungsingenieur Wissen über die Identifikation von Crosscutting Concerns oder extensives Domänenwissen über die vorliegende Software verlangt.
Ausschlusskriterium 3 garantiert, dass der Ansatz auf textuellen Dokumenten arbeiten kann. Das ist zwingend notwendig, da die Themenvorgabe dieser Arbeit die Analyse von User Stories erfordert.
Ausschlusskriterium 4 stellt sicher, dass die Quelle mithilfe des Universitätszugangs zugänglich ist.
Ausschlusskriterium 5 bescheinigt, dass der Artikel von Experten begutachtet als Konferenzband oder im Rahmen eines Journals erschienen ist. Außerdem wird mit

Ausschlusskriterium 5 verlangt, dass der Artikel ohne Referenzen nicht weniger als drei Seiten Umfang hat.
Ausschlusskriterium 6 stellt sicher, dass es sich um einen englischsprachigen Artikel handelt.

Als Einstieg wurden drei Artikel als Ausgangsliteratur vorgegeben. Um neue Artikel zu finden, wurde auf den gegebenen Artikeln Vorwärts- und Rückwärtssuche betrieben. Wenn nicht anders beschrieben, wurden Titel und Zusammenfassung aller gefundenen Artikel zunächst im Hinblick auf die Ausschlusskriterien geprüft. Die dann übrig gebliebenen Kandidaten wurden im Detail gelesen.

Insgesamt wurden zusätzlich zu den drei Ausgangsartikeln durch Vorwärts- und Rückwärtssuche sechs weitere Artikel gefunden.
Eine Auflistung der gefundenen Literatur findet sich in Abschnitt 3.4 in Tabelle 6.
Durch den unterschiedlichen Aufbau der Artikel bot sich eine Einordnung in zwei Kategorien an:

- **Technische Artikel**, die einen neuen Ansatz vorstellen und diesen wahlweise durch eine Fallstudie validieren.

- **Evaluierende Artikel**, die einen Überblick über bestehende Forschungsbemühungen geben und diese in Anbetracht verschiedener Faktoren gegenüberstellen.

Im Folgenden bietet Tabelle 2 einen Überblick über die Verteilung der gefundenen Artikel auf diese beiden Kategorien.
Rosenhainer [26] ist, wegen den sowohl technischen als auch evaluierenden Qualitäten des Artikels, in beiden Spalten der Tabelle vertreten.

Technische Artikel	**Evaluierende Artikel**
Baniassad et al. [6]	Rosenhainer [26]
Sampaio et al. [28]	Herrera et al. [17]
Rosenhainer [26]	Cojocar et al. [13]
Rashid et al. [23]	
Rago et al. [22]	
Ali et al. [2]	
Ali et al. [3]	

Tabelle 2: Kategorisierung der Artikel

Eine auch im weiteren Verlauf der Arbeit wiederholt aufgegriffene Problematik der Literaturrecherche war die inkonsequente Bezeichnung von gemeinsamen Bestandteilen in Anforderungen in den gefundenen Artikeln. Oft wurden ohne ausreichende Differenzierung oder Definition *Aspects* und *Crosscutting Concerns* als Synonyme benutzt. Zur besseren Verständlichkeit wird im weiteren Verlauf der Arbeit ein gemeinsamer Bestandteil von Anforderungen einheitlich als *Crosscutting Concern* bezeichnet.

3.2 Rückwärtssuche

Tabelle 3 gibt einen Überblick über die Anzahl der Referenzen der Ausgangsliteratur.

Autor	**Titel**	**# Referenzen**
Baniassad, Elisa, and Siobhán Clarke	Finding aspects in requirements with theme/doc [6]	11
Rosenhainer, Lars	Identifying crosscutting concerns in requirements specifications [26]	15
Sampaio, Américo, et al.	Mining aspects in requirements [28]	10

Tabelle 3: Referenzen der Rückwärtssuche

Die Referenzen von Baniassad et al. [6] werden in Tabelle 16 als Baniassad 1 bis Baniassad 11 bezeichnet.
Die Referenzen von Rosenhainer [26] werden in Tabelle 16 als Rosenhainer 1 bis Rosenhainer 15 bezeichnet.
Die Referenzen von Sampaio et al. [28] werden in Tabelle 16 als Sampaio 1 bis Sampaio 10 bezeichnet.

Bei der Rückwärtssuche (Backwards-Snowballing) wurden alle Referenzen der drei als Ausgangsliteratur vorgegebenen Artikel inspiziert.
Als Duplikat wurden vier Referenzen, nach AK1 18 Referenzen, nach AK2 zwei Referenzen, nach AK4 sechs Referenzen und nach AK5 zwei Referenzen ausgeschlossen.
Tabelle 16 in Anhang A.1 zeigt die Referenzen der Ausgangsliteratur sowie das Kriterium, das zum Ausschluss eines Artikel führte.
Nach Anwendung aller Ausschlusskriterien wurde schließlich Rashid et al. [23]: "Early Aspects- a Model for Aspect-Oriented Requirements Engineering" ausgewählt.

3.3 Vorwärtssuche

Tabelle 4 gibt einen Überblick über die Anzahl der Zitierungen der Ausgangsliteratur.

Autor	**Titel**	**# Zitierungen**
Baniassad, Elisa, and Siobhán Clarke	Finding aspects in requirements with theme/doc [6]	112
Rosenhainer, Lars	Identifying crosscutting concerns in requirements specifications [26]	80
Sampaio, Américo, et al.	Mining aspects in requirements [28]	96

Tabelle 4: Zitierungen der Vorwärtssuche

Insgesamt konnten bei der Vorwärtssuche (Forward-Snowballing) 288 Artikel identifiziert werden.
Um die Relevanz zu den Forschungsfragen sicherzustellen und eine effiziente Auswahl von Artikeln zu gewährleisten, wurde im Hinblick auf die oben angeführten Ausschlusskriterien ein Schlagwortkatalog erarbeitet. Alle 288 Artikel, die durch die Ausgangsliteratur Baniassad et al. [6], Rosenhainer [26] und Sampaio et al. [28]

gefunden wurden, wurden mithilfe des Schlagwortkatalogs durchsucht. Als erster Suchbegriff wurde unter Berücksichtigung von AK1 und AK3 “aspect mining techniques” gewählt. Um automatische Ansätze zu fokussieren, wurde unter zusätzlicher Berücksichtigung von AK2 “automatic crosscutting concern identification” formuliert. Um die in Abschnitt 3.1.2 angesprochene Problematik der Namenskonvention zu adressieren, wurden bewusst sowohl die Bezeichnungen *aspect* als auch *Crosscutting Concern* in die Begriffsuche aufgenommen. Die Suchbegriffe sind oder-verknüpft, also beliebig untereinander austauschbar.
Dabei ergaben sich mit dem ersten Suchbegriff unter Ausschluss der Duplikate 21 Ergebnisse. Mithilfe des zweiten Suchbegriffes wurden unter Ausschluss der Duplikate zwei Ergebnisse ermittelt.
Der Schlagwortkatalog und die Einzelheiten der Suche sind nachfolgend in Tabelle 5 einzusehen. Das # Zeichen beschreibt die Anzahl der gefundenen Ergebnisse. “Rel.” steht als Abkürzung für *relevant* und “rel. #” beschreibt die Anzahl an relevanten Ergebnissen. In der rechten Spalte beziehungsweise untersten Zeile ist jeweils die Summe der Suchergebnisse aufgeführt.

Suchbegriff	**Suchergebnisse**						**Summe** (ohne Duplikate)	
	Baniassad et al. [6]		Rosenhainer [26]		Sampaio et al. [28]			
	#	rel. #	#	rel. #	#	rel. #	#	rel. #
“aspect mining techniques”[1]	9	2	4	1	18	3	21	3
“automatic crosscutting concern identification“ [2]	2	1	2	1	0	0	2	1
Summe der beiden Suchbegriffe	11	3	6	2	18	3	23	4

Tabelle 5: Verfeinerung der Suchergebnisse anhand eines Schlagwortkatalogs

[1] Suche am 30.06.2018
[2] Suche am 23.07.2018

Die Artikel wurden als nächstes, wie in Abschnitt 3.1.2 beschrieben, hinsichtlich der Ausschlusskriterien durchgesehen. Dabei blieben, wie in obiger Tabelle einsehbar, vier Kandidaten übrig.
In Tabelle 17 werden mit Baniassad 1.1 bis Baniassad 1.9 die Suchergebnisse mithilfe des ersten Suchbegriffs auf Baniassad et al. [6] beschrieben.
Mit Baniassad 2.1 und Baniassad 2.2 werden die Suchergebnisse mithilfe des zweiten Suchbegriffs auf Baniassad et al. [6] beschrieben.
Mit Rosenhainer 1.1 bis Rosenhainer 1.4 werden die Suchergebnisse mithilfe des ersten Suchbegriffs auf Rosenhainer [26] beschrieben.
Mit Rosenhainer 2.1 und Rosenhainer 2.2 werden die Suchergebnisse mithilfe des zweiten Suchbegriffs auf Rosenhainer [26] beschrieben.
Mit Sampaio 1.1 bis Sampaio 1.18 werden die Suchergebnisse mithilfe des ersten Suchbegriffs auf Sampaio et al. [28] beschrieben.
Insgesamt ergaben sich ursprünglich 35 Ergebnisse.
Zwölf Ergebnisse wurde als Duplikat, neun nach AK1, eines nach AK2, drei nach AK3, eines nach AK5 und drei nach AK6 ausgeschlossen.
Die Aufgliederung des oben Erwähnten ist in Tabelle 17 im Anhang A.1 einsehbar.
Der Inhalt des Artikels von Ali et al. [4] entspricht zu großen Teilen dem Inhalt von Ali et al. [3]. Ein Unterschied ist der Anhang eines Umsetzungsbeispiels anhand eines "Campus Messenger Systems". Sollte sich bei der Implementierung herausstellen, dass die beschriebene Vorgehensweise in Ali et al. [3] nicht ausreichend dokumentiert ist, ist es sinnvoll diesen Artikel hinzuzunehmen.
Zu diesem Zeitpunkt bietet dieser Artikel jedoch keine ausreichend einzigartigen Erkenntnisse im Vergleich zu seinem Vorgänger [3], als dass die Hinzunahme in das Literaturportfolio nötig wäre.

3.4 Ergebnisse

Nachfolgend beschreibt Tabelle 6 die final ausgewählte Literatur.

Autor	Name	Fundort
Baniassad, Elisa, and Siobhán Clarke	Finding aspects in requirements with theme/doc	Ausgangsliteratur
Sampaio, Américo, et al.	Mining aspects in requirements	Ausgangsliteratur
Rosenhainer, Lars	Identifying crosscutting concerns in requirements specifications	Ausgangsliteratur
Rashid, Awais, et al.	Early aspects: A model for aspect-oriented requirements engineering	Rückwärtssuche
Ali, Busyairah Syd, and Zarinah Mohd Kasirun	3ci: A tool for crosscutting concern identification	Vorwärtssuche
Ali, Busyairah Syd, and Zarinah Mohd Kasirun	Crosscutting concern identification at requirements level	Vorwärtssuche
Rago, Alejandro et al.	Early aspect identification from use cases using NLP and WSD techniques	Vorwärtssuche
Cojocar, Grigoreta Sofia, and Gabriela Şerban	On some criteria for comparing aspect mining techniques	Vorwärtssuche
Herrera, José, et al.	Revealing crosscutting concerns in textual requirements documents: an exploratory study with industry systems	Vorwärtssuche

Tabelle 6: Ausgewählte Literatur

3.5 Zusammenfassung

Die überwiegende Mehrheit der vorliegenden Artikel betont die große Bedeutung der möglichst frühen Identifikation von Crosscutting Concerns im Software Engineering Prozess [2, 3, 6, 17, 28]. Dabei wird ein besonderer Fokus auf die Identifikation bei der Anforderungserhebung, namentlich auf das Aspect Oriented Requirements Engineering, gelegt. Es wird hervorgehoben, dass das richtige und frühe Erkennen von Crosscutting Concerns den weiteren Entwicklungsprozess, zum Beispiel im Hinblick auf Design- und Architekturwahl, maßgeblich beeinflusst.

Viele der bestehenden Ansätze sind nicht automatisch oder zumindest halbautomatisch und erzielen dadurch zwar eine hohe Treffergenauigkeit (vgl. Rosenhainer [26]: "Maximaler recall bei 99,4% precision in Relation zum erarbeiteten Standard") bei der Identifikation von Crosscutting Concerns, allerdings erfordert dies einen hohen Aufwand seitens des Anforderungsingenieurs.
Die Problematik bei der Formulierung eines vollautomatischen Ansatzes zur Identifikation von Crosscutting Concerns mithilfe von textuellen Dokumenten liegt hauptsächlich bei der Bestimmung der relevanten Verben, auch *Action Words* [28, 3] genannt. Diese Verben stehen bei der Identifikation von Crosscutting Concerns im Mittelpunkt, da durch sie das Verhältnis zwischen den einzelnen Anforderungen ermittelt werden kann. Das stellt durch verschiedene Auslegungsmöglichkeiten der natürlichen Sprache eine zentrale Schwierigkeit dar. Diese Schwierigkeit ist ebenfalls der Grund, warum viele Ansätze zumindest teilweise auf das anwendungsspezifische Wissen des Anforderungsingenieurs angewiesen sind.
Problematisch ist ebenfalls die inkonsistente Definition von Crosscutting Concerns, da dies den Vergleich der einzelnen Ansätze erschwert. Diese Problematik wird auch in einigen evaluierenden Artikeln, wie zum Beispiel [17] angesprochen, in denen versucht wird, eine einheitliche Namenskonvention zu etablieren.

Antwort auf Forschungsfrage 1 (Techniken zur Identifikation):
Circa die Hälfte der relevanten gefundenen Literatur bezüglich Aspect Mining beruft sich bei der automatisierten Extraktion von Crosscutting Concerns auf den Quellcode um entweder mithilfe von Clustering oder Dynamic Analysis Crosscutting Concerns zu ermitteln. Da diese Ressourcen bei der Analyse von User Stories nicht oder nicht sinnvoll zur Verfügung stehen, beschränkt sich die automatische Extraktion von Crosscutting Concerns aus User Stories auf die Erstellung von Graphen beziehungsweise Unified Modeling Language (UML)-ähnlichen Diagrammen [6], die Nutzung von Natural Language Processing [2, 27] und Information Retrieval [26].
Die Nutzung von Graphen erfordert anschließend noch eine manuelle Durchsicht des erstellten Modells, um Crosscutting Concerns zu identifizieren. Bei keinem gefundenen Ansatz konnte dieser Arbeitsschritt automatisiert werden. Damit stellt dies zusätzlichen Aufwand für den Anforderungsingenieur dar, der dabei außerdem über spezifisches Domänenwissen im Kontext der Software verfügen muss.
Für die geplante Umsetzung als automatisiertes Werkzeug bietet sich daher die Nutzung von NLP an. Dieses sollte wenigstens in Kombination mit anderen Techniken zur Extraktion von Crosscutting Concerns aus User Stories genutzt werden.

Antwort auf Forschungsfrage 2 (Folgeaktion):
Das Garantieren und Verbessern der Nachverfolgbarkeit und die Umsetzung der

SoC mithilfe des vorgestellten Ansatzes, wird in Artikeln [2, 23, 28] angesprochen. Dabei bietet sich als zusätzliche Aktion, nach der erfolgreichen Identifikation von gemeinsamen Bestandteilen in User Stories, zum Beispiel das Ausstatten selbiger mit dem passenden Feature Tag an. Dadurch wird die Nachverfolgbarkeit im Projekt sichergestellt. Diese Aktion wurde ebenfalls in der Aufgabenstellung angesprochen.
Beim Umgang mit User Stories ist als zusätzliche Aktion bei der Identifikation von Crosscutting Concerns die Umformulierung und Zusammenlegung mehrerer User Stories nötig.
Bei ungenau formulierten Anforderungen ist es nach erfolgreicher Identifizierung oft notwendig, bereits vorhandene Verlinkungen zwischen den Artefakten entsprechend anzupassen und gegebenenfalls zu löschen. Die Aufbereitung der Ergebnisse für diese Aktion wird vor allem in den Artikeln [28, 6] angeführt.

4 Verwandte Arbeiten

Nachfolgend werden in Abschnitt 4.1 die Ansätze der final ausgewählten Artikel erörtert. Anschließend werden die Artikel in Abschnitt 4.2 miteinander verglichen. Abschnitt 4.3 geht auf den ausgewählten Ansatz und das weitere Vorgehen näher ein. In Abschnitt 4.4 werden zusätzliche Anforderungen an den Prototypen FElICiTy aufgestellt.

4.1 Zusammenfassung der Ansätze

Baniassad et al. [6] beschreiben zunächst über die Signifikanz der Identifikation von Crosscutting Concerns und die Problematik, die selbige darstellt. Dabei sei es schwierig zwischen abhängigem Verhalten und schlecht gekapseltem Verhalten im Bezug auf Software Artefakte zu unterscheiden. Im Kontext ihres Ansatzes bezeichnet ein sogenanntes *Theme* ein Feature. Ziel des vorgestellten Ansatzes ist es *Crosscutting Themes* zu identifizieren.
Bei diesem Ansatz wird aus den Anforderungen mit Hilfe von Keywords ein Graph (Clipped Action Graph, einzusehen in Abbildung 4.1) erstellt, der UML-ähnlich die Abhängigkeit und Hierarchie der einzelnen Anforderungen darstellt. Dabei sind kritische Schritte, wie die Extraktion von Schlüsselwörtern aus den Anforderungen sowie die Evaluation der Graphen zur Identifikation von Crosscutting Concerns per Hand durchzuführen. In diesem Zusammenhang ist Domänenwissen des Anforderungsingenieurs erforderlich. Durch die nur halbautomatische Natur des Ansatzes ist außerdem die Skalierbarkeit ein größeres Problem.
Zusätzlich ist die Eingabe durch die Notwendigkeit von speziell angepassten Anforderungen relativ eingeschränkt. Hierbei muss parallel zu den vorliegenden Anforderungen eine Menge an Schlüsselaktionen definiert werden, die die zentrale Funktionalität der Anforderung repräsentieren. Dieser Vorgang muss manuell durch den Anforderungsingenieur durchgeführt werden.

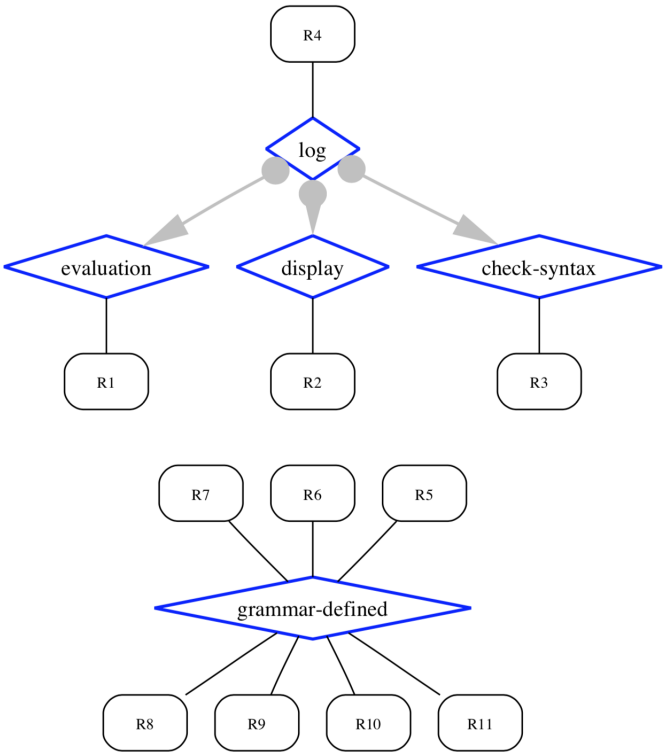

Abbildung 4.1: Clipped Action View [6]

Sampaio et al. [28] gehen zunächst auf die teilweise oben angeführten Nachteile des Theme/Doc Ansatzes [6] bezüglich Skalierbarkeit ein.
Der von ihnen vorgestellte Ansatz beruht auf der kontextsensitiven Analyse von Textdokumenten mithilfe von NLP. Dadurch sei es möglich, beliebig große Textdokumente effizient zu analysieren.
Als POS-Tagger und Natural Language Processor bedient sich der Ansatz des WMATRIX Werkzeugs, welches neben gängigen POS-Tagging Kapazitäten zusätzlich die Möglichkeit bietet, eine semantische Analyse der Eingabedaten durchzuführen. Hierbei wird das UCREL Semantic Analysis System (USAS) [25] genutzt. Dieses benutzt eine Menge an semantischen Bezeichnern, welche auf dem Tom McArthur's Longman Lexicon of Contemporary English[3] basieren.
Die grobe Architektur des Werkzeugs ist nachfolgend in Abbildung 4.2 einsehbar. Als Eingabe für das Mining Werkzeug dienen Anforderungsdokumente in natürlicher Sprache, die mithilfe von WMATRIX analysiert werden.
Es wird ein optionaler Schritt angeführt, bei dem das System aus verschiedenen

[3]McArthur, Tom, and Thomas G. McArthur. Longman lexicon of contemporary English. London: Longman, 1981.

architekturellen Standpunkten beschrieben wird. Dies sei vor allem für den Anforderungsingenieur hilfreich, da dadurch eine übersichtliche Darstellung der Ergebnisse erreicht wird.
Vorteile des Ansatzes sind die gute Skalierbarkeit und die automatische Durchführung von zeitintensiven Aktivitäten wie das Identifizieren von Concerns und *Action Words*. Allerdings bleibt die Notwendigkeit eines Anforderungsingenieurs, um das Werkzeug durch die einzelnen Schritte zu führen. Nur so wird die korrekte Verfeinerung der Ergebnismenge und schlussendlich die korrekte Auswahl von Crosscutting Concerns gewährleistet.

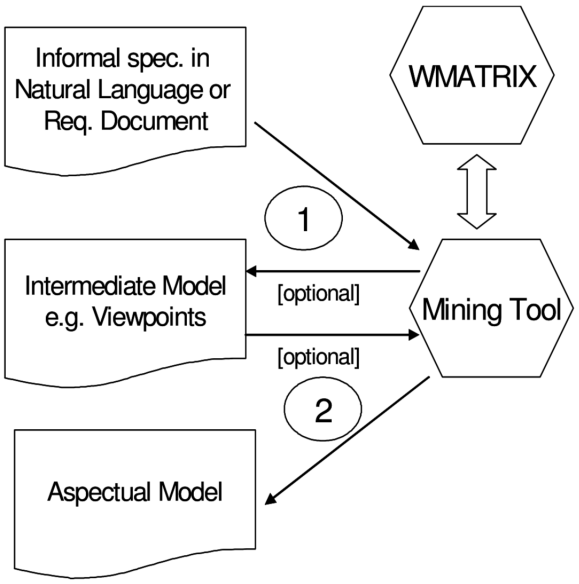

Abbildung 4.2: Aspect Mining Modell [28]

Rosenhainer [26] betont in seinem Artikel zunächst die Wichtigkeit der Identifikation von Crosscutting Concerns, da sie die Nachverfolgbarkeit des Software-Systems maßgeblich positiv beeinflusse. Zusätzlich steigt die Menge an aspektorientierten Ansätzen, was dieser Thematik zusätzliche Relevanz gibt. In diesem Zusammenhang wird außerdem die wichtige Einhaltung des Konzepts der SoC angesprochen.
Nachfolgend wird auf die Problematik von nichtfunktionalen Anforderungen eingegangen, welche oftmals versteckte Einflüsse auf andere Anforderungen hätten. Deshalb sei die explizite Formulierung von NFRs wichtig.
Bei der Änderung und Adaption von Anforderungen oder dem System selbst sei ebenfalls eine Kapselung von Crosscutting Concerns wichtig. Anschließend wird ein NLP-Information Retrieval Ansatz beschrieben, der die Identifikation erleich-

tern soll.

Einen Überblick über vorhergehende Forschungsbemühungen zum Voranbringen des AOSD gibt Rashid et al. [23] und stellt einen Ansatz vor, um die Identifizierung von Crosscutting Concerns zu unterstützen. Außerdem bemängeln die Autoren die mangelnde Definition von Crosscutting Concerns im frühen Stadium des Requirements Engineering.
Beim dem beschriebenen Ansatz werden zunächst manuell mithilfe der Anforderungen Concerns und architekturelle Standpunkte identifiziert, die sich aus bestimmten Schlüsselwörtern der Anforderungen in Verbindung mit Domänenwissen über Teilhaber o.ä. ergeben. Diese werden dann durch das Werkzeug „JPREView“ in Relation zueinander gesetzt. Wenn in diesem Modell architekturelle Standpunkte existieren, die mit mehreren Concerns in Relation stehen, wird das Concern als sogenannter *Candidate Aspect* gekennzeichnet. Die finale Auswahl der Crosscutting Concerns muss dann wiederum durch den Softwareengineer mithilfe von Domänenwissen stattfinden.

Ali et al. [2] stellen ein auf dem Theme/Doc und dem EA-Miner Ansatz aufbauendes Werkzeug vor.
Zunächst wird die fehlende Automatisierung bemängelt, welche bestehende Werkzeuge zur Extraktion von Aspekten bieten. Darunter leide die Skalierbarkeit dieser Ansätze in großem Maße.
Das Werkzeug selbst ist webbasiert und besteht aus drei Modulen, einem Anforderungsverwaltungs-Modul, einem NLP-Modul und einem Modul zur Identifikation von Crosscutting Concerns aus den bearbeiteten Anforderungen.
Die grobe Architektur des Werkzeugs ist in Abbildung 4.3 dargestellt.
Ein klarer Vorteil des beschriebenen Ansatz ist große Bandbreite an Anfordungsformaten, die als Eingabe des Werkzeugs genutzt werden können. Eine besondere Struktur oder Formatierung des Textes der Anforderung ist, durch die Verwendung von NLP, nicht nötig. Dies ist vor allem im Hinblick auf die Analyse von User Stories, die als Fließtext realisiert sind, wichtig. Auch die voll automatisch ablaufende Analyse ist großer Nutzen- und Zeit-Vorteil.
Allerdings räumen die Autoren ein, dass dem Werkzeug noch einige teils wichtige Komfortfeatures fehlen und die Evaluation des Werkzeugs mithilfe einer Fallstudie noch nicht stattgefunden hat. Trotzdem stellt das Werkzeug aus diesem Artikel einen vielversprechenden Ansatz dar.

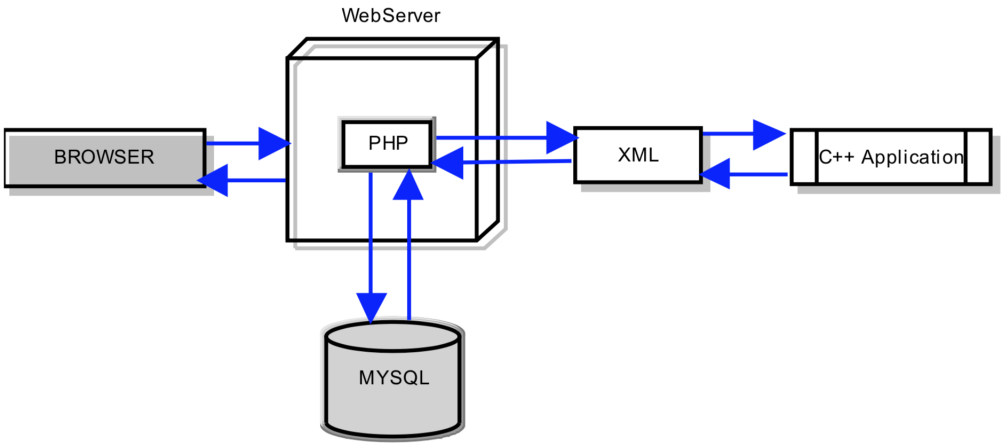

Abbildung 4.3: 3CI-Werkzeug Architektur [2]

Ali et al. [3] gehen in ihrem Artikel neben der Wichtigkeit der Umsetzung von Aspect Oriented Requirements Engineering auch auf die mangelnde Definition von zentralen Begriffen in diesem Umfeld wie *Crosscutting Concern* und *Aspect* ein. Nachfolgend bieten sie eine Definition ebendieser Begriffe.
Anschließend beschreiben sie vorausgehende Forschungsbemühungen; im Einzelnen den Theme/Doc Ansatz [6], den Early-AIM [24] / EA-Miner Ansatz [27] und den Ansatz basierend auf *Information Retrieval* [26].
An all diesen Ansätzen sei deren halbautomatische Natur zu kritisieren, die dem Anforderungsingenieur immer noch viel Arbeit verursache.
Nachfolgend stellen sie einen Ansatz vor, aus dem zu späterem Zeitpunkt das 3CI-Werkzeug [2] hervorgehen wird. Dieser Ansatz vereint NLP-Elemente zur Identifizierung von *Action Words* und dominanten Verben in den Anforderungsdokumenten mit der graphischen Darstellung der Relation zwischen Anforderung und sogenannten *Candidate Aspects*, welche aus den ermittelten dominanten Verben hervorgehen und eine vorläufige Version der finalen Crosscutting Concerns darstellen. Diese Darstellung wurde aus dem Theme/Doc Ansatz [6] übernommen.

Rago et al. [22] greifen die Problematik auf, dass viele der vorliegenden NLP-basierenden Ansätze eine durchweg schlechte Präzision bei der Erkennung von Crosscutting Concerns aufweisen. Diese sei hauptsächlich durch die schwierige Auslegung der natürlichen Sprache verursacht, welche mit konventionellen NLP-Techniken wie POS-Tagging nicht ausreichend und eindeutig genug analysiert werden könne.
Der angeführte Lösungsansatz beinhaltet die zusätzliche Nutzung von word-sense disambiguation (WSD). Hierbei werden auf semantischer Ebene Verben und direkte Objekte (Objekte, die sich syntaktisch direkt auf ein Verb beziehen) mithilfe von

semantischen Datenbanken zu clustern zusammengefasst. Der daraus resultierende Ansatz nutzt eine komplexe Aneinanderreihung von einzelnen syntaktischen und semantischen Analyseschritten, einzusehen in Abbildung 4.4, um einen Action-oriented identifier graph (AOIG) zu erstellen. Innerhalb dieses Graphen sind die analysierten use-cases mit den relevanten Verb-Objekt-Gruppen verknüpft.
Um den Graphen zu erstellen, wird zunächst die Dokumentation der use-cases mithilfe von NLP-Analyse verarbeitet. Ergebnis ist die strukturierte Dokumentation mit korrespondierendem POS- und semantischem Tagging. Danach werden mithilfe von Regeln zum Musterabgleich Verb- und Aktionsgruppen erstellt, die dann zu einem Graph zusammengefügt werden. Davon ausgehend wird beim Durchlaufen des Graphen eine hierarchische Liste an potentiell abhängigen use-cases erstellt. Die finale Auswahl und etwaige Filterung findet dann durch den Anforderungsingenieur statt.
Der Ansatz wird anschließend anhand von zwei Fallstudien evaluiert. Es wird ein Recall von 62% bis 90% bei einer Präzision von 42% bis 67% erreicht.
Der größte Nachteil des Ansatzes manifestiert sich allerdings in der Durchführungszeit, die bei einer Dokumentengröße von 9 use-cases und 15 Anforderungsspezifikationen bei 60 bis 90 Minuten lag.

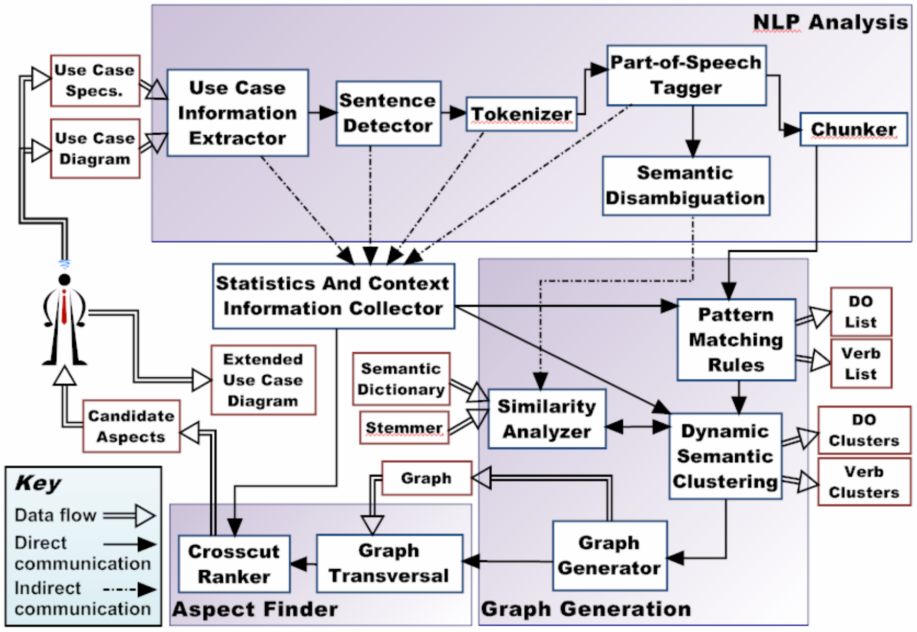

Abbildung 4.4: Schritte zur Aspect Identifizierung [22]

Cojocar et al. [13] präsentieren in ihrem Artikel eine Möglichkeit, verschiedene Aspect Mining Techniken effizient zu vergleichen. Zunächst geben sie einen

Überblick über bestehende Vergleichsansätze und Metriken.
Danach wird ein Modell zum Vergleichen eingeführt, das allgemeine Kriterien definiert, um die verschiedenen Ansätze sinnvoll vergleichen zu können.
Schließlich wird das Modell beispielhaft auf eine Anzahl an bestehenden Techniken angewandt, um die Funktionsweise zu demonstrieren. Dies geschieht allerdings nur auf theoretischer Ebene, da für die meisten der vorgestellten Ansätze keine Ergebnisse zu den tatsächlich identifizierten Crosscutting Concerns vorliegen.

In Herrera et al. [17] referieren die Autoren zunächst über die bestehenden Bemühungen im Forschungsfeld des AOSD und die mangelnde Evaluation der bereits existierenden Ansätze.
Außerdem werden die Ansätze aus Banissad et al. [6] und Sampaio et al. [27] in einer explorativen Studie getestet und verglichen.
Beim Vergleich der beiden Ansätze definieren die Autoren zunächst eine gemeinsame Terminologie und einen festen Satz an Aktivitäten, die zur Bestimmung der Crosscutting Concerns durchführt werden müssen. Mithilfe dieser wird dann die Evaluation durchgeführt, welche maßgeblich die benötigte Zeit zur Durchführung dieser Aktivitäten und die Qualität der entstandenen Ergebnisse mithilfe von Metriken *Precision* und *Recall* wertet.
Anschließend werden beide Ansätze zusätzlich durch die Anwendung auf zwei reale Industrieprojekte im Bankensektor getestet. Bei ausreichender Güte der Eingangsdaten, also ausführlich beschriebenen textuellen Anforderungen, ist die Durchführung des EA-Miner Ansatzes schneller. Die Qualität der Ergebnisse ist hingegen beim Theme/Doc Ansatz fast durchgehend besser.
Die Autoren kommen zu dem Ergebnis, dass bei der Anforderungserhebung am besten EA-Miner unterstützend eingesetzt werden sollte; der Theme/Doc Ansatz sollte genutzt werden um anschließend die Anforderungen während dem Entwicklungsprozess zu prüfen.

4.2 Vergleich der Ansätze

Im Folgenden werden die zuvor zusammengefassten Artikel im Hinblick auf ihre Bedeutung bezüglich der Forschungsfragen aus Abschnitt 3.1.1 verglichen.
Mögliche Ansätze zur Implementierung sind in den technischen Artikeln, entsprechende Evaluationsdaten in den evaluierenden Artikeln zu finden. Artikel 1 bis 7 sind, wie in Abschnitt 3.1.2 erörtert, technische Artikel, in denen ein Ansatz zur Identifikation von Crosscutting Concerns vorgestellt und diskutiert wird.

Die beiden evaluierenden Artikel Cojocar et al. [13] und Herrera et al. [17] vergleichen eine Reihe an Ansätzen, die teilweise ebenfalls in nachfolgende Synthese

aufgenommen werden können.
In Cojocar et al. [13] werden die Ansätze EA-Miner [27], Clone Detection [10], Fan-in [20], Execution relations [7], Execution traces [31] und History Mining [8] verglichen.
Allerdings befasst sich einzig der EA-Miner Ansatz mit der Analyse von Anforderungsdokumenten, wodurch ausschließlich die Hinzunahme dieses Ansatzes in die Synthese sinnvoll ist.
In Herrera et al. [17] werden die Ansätze EA-Miner [27] und Theme/Doc [6] gegenübergestellt.
Somit genügen die Evaluationsdaten aus Cojocar et al. [13] und Herrera et al. [17], um EA-Miner in die Synthese aufzunehmen.

Die Vergleichskriterien wurden unter Zugrundelegung der Forschungsfragen erarbeitet und haben zum Ziel, diese umfassend zu beantworten.
Die Kategorien *Automatisierungsgrad* und *Genutzte Technik* sind elementar, um die Eignung des Ansatzes als automatisiertes Werkzeug zu evaluieren. Diese Kategorien gehen aus Forschungsfrage 1 hervor.
Bei der Beschreibung des Automatisierungsgrads steht *semi* für einen halbautomatischen Ansatz, bei dem Teile des Identifikationsprozesses durch den Anforderungsingenieur durchgeführt werden müssen. Entsprechend beschreibt *voll* eine volle oder so weit wie sinnvoll mögliche Automatisierung.
Im Hinblick auf die automatische Eignung müssen manuelle Bestandteile eines Ansatzes mit dem Automatisierungsgrad *semi* mindestens eines der folgenden Kriterien erfüllen:

- Durchführung der erforderlichen Schritte ist ohne zusätzliches Fachwissen durch den Anforderungsingenieur möglich
- Manuelle Schritte können durch analoge Schritte aus voll-automatisierten Ansätzen, die gleiche Funktionalitäten umsetzen, ersetzt werden

Die Kategorie *Folgeaktion* geht direkt aus Forschungsfrage 2, die anschließende Aktionen nach der automatisierten Identifikation verlangt, hervor.
Die Kategorie *Produziertes Ergebnis* erlaubt die folgende Evaluierung: Ist das Ergebnis des Ansatzes im Kontext eines JIRA-Plugins sinnvoll darzustellen?
Zusätzlich wäre zur Auswahl des effektivsten Ansatzes hilfreich gewesen, die Ansätze anhand von Metriken bezüglich Treffergenauigkeit zu vergleichen.
Einzig in den Ansätzen Rosenhainer [26] und Rago et al. [22] lieferten in ihre Fallstudie eine Evaluation nach Metriken.
In Herrera et al. [17] konnten zusätzlich Metriken zur Perfomanz von EA-Miner [27] und Theme/Doc [6] gefunden werden. Allerdings wurde hier auch ersichtlich, dass die Qualität des Ansatzes sehr stark von Größe und Güte der Eingangsdaten

abhängt. Somit ist es nicht möglich, eine einheitliche Vergleichsbasis zu schaffen, weswegen von der Aufführung von Metriken zum Vergleich abgesehen wurde.
Diese Gegenüberstellung war nicht möglich, da von den ausgewählten Ansätzen nur ein einziger (Rosenhainer [26]) Metriken zur Performanz des Ansatzes anführte.

In nachfolgender Tabelle 7 werden die Gemeinsamkeiten und Unterschiede der in den technischen Artikeln vorgestellten Ansätze aufgezeigt.
In der Spalte "Ansatz" ist neben der Autorenreferenz auch der Name des Ansatzes aufgeführt, falls von den Autoren ein Terminus vergeben wurde.

Ansatz	**Automatisierungsgrad**	**Folgeaktion**	**Eingabe**	**Produziertes Ergebnis**	**Genutzte Technik**
Ansatz 1: Baniassad et al. [6] *Theme/Doc*	semi	Hierarchische Darstellung der Crosscutting Concerns	Speziell formatierte Anforderungen	Graph: *Clipped Action View*	Regelbasierte / manuelle Identifikation
Ansatz 2: Sampaio et al. [28]	voll (unter Ausnahme der Folgeaktion)	Beschreibung des Systems aus architekturellen Standpunkten	Beliebige textuelle Dokumente	Modell der Beziehungen zwischen Anforderungen	NLP
Ansatz 3: Rosenhainer [26]	semi	Verfeinerung der Anforderungen	use-cases, herkömmliche Textdokumente	Von anderen Anforderungen beeinflusste *Requirement Leaves*	Information Retrieval
Ansatz 4: Rashid et al. [23] *Early aspects*	semi	Darstellung des Systems als *aspects*; dadurch: Umsetzung von SoC	*goal-oriented* Anforderungen, use-cases	AORE-Modell	Regelbasiert mit Werkzeug JPREView
Ansatz 5: Ali et al. [2] *3CI*	voll	Strukturierung der Anforderungen	englischsprachige textuelle Anforderungen	POS bzw. semantisch analysierte Anforderungen	NLP
Ansatz 6: Ali et al. [3]	voll	Matrixdarstellung der Beziehungen zwischen Anforderungen	englischsprachige textuelle Anforderungen	Modell der Abhängigkeiten / Identifizierte Crosscutting Concerns	NLP
Ansatz 7: Rago et al. [22]	semi	Generierung des AOIG	use-cases	Aspekt Kandidaten	NLP / WSD
Ansatz 8: Sampaio et al. [27]	voll	Beschreibung des Systems aus architekturellen Standpunkten	englischsprachige textuelle Anforderungen	Crosscutting Concern Kandidaten	NLP / WMATRIX

Tabelle 7: Synthese der relevanten Ansätze

4.3 Weiteres Vorgehen

Im Hinblick auf die in Abschnitt 3.1.2 erarbeitete Kategorisierung in technische und evaluierende Artikel, bietet sich folgende weitere Vorgehensweise an:
Um der Umsetzung als automatisiertes Werkzeug zu genügen, sollte ein vollautomatisierter Ansatz zur Implementierung des FElICiTy Prototypen gewählt werden. Die bei allen halbautomatisierten Ansätzen nötige Interaktion durch den Anforderungsingenieur übersteigt den Grad, der bei Einhaltung der geforderten nichtfunktionalen Anforderung *Benutzbarkeit* zu vertreten wäre.
Auszunehmen hiervon ist Rago et al. [22], deren Ansatz sich doch aus unten angeführten Gründen zur Umsetzung nicht direkt eignet.
Bei der Wahl eines vollautomatisierten Ansatzes beschränkt sich die Auswahl der Technik damit auf NLP, was jedoch im Hinblick auf die in Abschnitt 2.2 angesprochene Strukturierung der User Story als textuelles Dokument keinen Nachteil darstellt.
Rago et al. [22] erkennt und adressiert viele Probleme, unter denen andere NLP-basierte Ansätze leiden. Der dort vorgestellte Ansatz schafft es allerdings nicht, diese im Kontext dieser Arbeit sinnvoll zu bewältigen, da realistische Zeitbeschränkungen bei weitem überschritten werden. Deshalb ist die Adaption des Ansatzes aus Rago et al. [22] als Ganzes nicht sinnvoll. Es können jedoch Teile des Ansatzes wie die semantische Analyse und die Fokussierung auf direkte Objekte anstelle von Verben, wie praktiziert in [2, 28], übernommen werden.
Der in Ali et al. [2] beschriebene 3CI-Ansatz stellt eine gute Basis zur Implementierung des Werkzeugs dar. Die geleistete Vorarbeit für diesen Ansatz ist in Ali et al. [3] einsehbar. Die dort verwendete NLP-Technik ist auf eine beliebige Art von textuellen Anforderungsdokumenten anwendbar. Mit dem in Abschnitt 2.2 dargelegten Aufbau von User Stories kann der Ansatz auf ebendiese zugeschnitten werden, indem die spezifische Formatvorlage ausgenutzt wird.
Der 3CI-Ansatz basiert auf dem Werkzeug EA-Miner, welches in Sampaio et al. [27] beschrieben wird. Die nötigen Evaluationsdaten sind aus Cojocar et al. [13] und Herrera et al. [17] übernommen worden.

4.4 Zusatzanforderungen

Die Übernahme von Teilen des besten Ansatzes und die Erarbeitung von zusätzlichen Anforderungen an den Prototypen, die sich dadurch ergeben, ist für eine erfolgreiche Implementation nötig. Um dieses Ziel zu konkretisieren, wurde eine zusätzliche Frage formuliert:

- Welche weiteren Anforderungen (zusätzlich zu den oben gegebenen Grobanforderungen) ergeben sich an den Prototypen im Kontext dieser Arbeit?

Bei der Verwendung eines NLP-basierten Ansatz ist die Schnittstelle zu dem Natural Language Prozessor zu beachten. Dieser kann womöglich nicht in das Plugin integriert werden und muss extern über PHP Skripte angebunden werden. Dies ist beispielsweise bei Ansätzen [27, 28] der Fall, die sich als Natural Language Processor dem webbasierten WMATRIX bedienen. Dieses ist nicht für die Integration in den Programmcode optimiert und benötigt deshalb eine zusätzliche Anbindung.
Im konkreten Hinblick auf die Implementierung des Werkzeugs als Jira-Plugin muss darauf geachtet werden, dass für den gewählten Natural Language Processor eine passende maven-dependency vorliegt.
Bei der Analyse der Ansätze, insbesondere bei Rago et al. [22] wurde ersichtlich, dass bei der Verwendung von NLP in Kombination mit anderen rechenintensiven Techniken schnell die Ausführungsdauer steigen kann. Deswegen sollte als nichtfunktionale Anforderung die Performanz mit einer sinnvollen Einschränkung der Ausführungsdauer aufgenommen werden. Hierbei ist wichtig, ein lineares Wachstum der Analysedauer proportional zur Anforderungsmenge anzustreben. Das kann durch effiziente Programmierung des Analyse-Algorithmus erreicht werden.
Eine Unterschreitung einer gewissen Zeit pro Anforderung die für das POS-Tagging des Natural Language Processors benötigt wird, ist nicht zu unterschreiten, da dieser Algorithmus nur begrenzt optimiert werden kann.
Es ist anzunehmen, dass die Benutzung des Werkzeugs singulär innerhalb einer Qualitätssicherung stattfindet, um den Nutzen des Werkzeugs zu maximieren. Deshalb ist es sinnvoll eine Zeitgrenze von einer Sekunde pro analysierter Anforderung festzulegen, da diese Zeitspanne je nach Projektgröße eine angemessene Dauer darstellt.
Wie bereits in Abschnitt 3.5 angesprochen, erfordern viele Ansätze eine manuelle Einsicht zur finalen Auswahl der Crosscutting Concerns. Diese Ansätze sind nur begrenzt zur Umsetzung als automatisiertes Werkzeug geeignet, allerdings geht aus der Evaluation der Ansätze hervor, dass in jedem Fall ein finales Auswählen der Crosscutting Concerns durch den Anforderungsingenieur nötig sein wird.
Im Gros der ausgewählten Artikel [17, 3, 2, 6, 28] wird über den hohen Aufwand der Identifikation von Crosscutting Concerns referiert, der dieser für den Anforderungsingenieur bedeutet. Im Hinblick darauf sollten als zusätzliche Anforderungen die einfache Bedienung des Werkzeugs und die übersichtliche Darstellung der Ergebnisse aufgenommen werden, damit diese Auswahl schnell und mit hoher Präzision durchgeführt werden kann. Damit einher geht die weitestmögliche Vermeidung von manueller Arbeit, die erforderlich ist, um die finalen Crosscutting Concerns auszuwählen.

5 FEllCiTy Prototyp

Folgendes Kapitel behandelt die Einzelheiten um den Plugin Prototypen FEllCiTy. Abschnitt 5.1 beschreibt die Anforderungen des Prototypen. In Abschnitt 5.2 wird der Entwurf erläutert. Abschnitt 5.3 behandelt die Implementierung des Werkzeugs im Einzelnen. Schließlich ist in Abschnitt 5.4 die Qualitätssicherung des Plugin Prototypen dokumentiert.

5.1 Anforderungen

Nachdem nunmehr in Abschnitt 4.4 alle nötigen Anforderungen an den Plugin Prototypen erarbeitet wurden, wurden diese, wie in den folgenden Abschnitten beschrieben, modelliert. Hierbei beschreibt Abschnitt 5.1.1 den User Task, Abschnitt 5.1.2 die Systemfunktionen und Abschnitt 5.1.3 die nichtfunktionalen Anforderungen. Abschnitt 5.1.4 gibt einen Überblick über den Aufbau der Benutzeroberfläche und Abschnitt 5.1.5 beschreibt die Domänendaten.

5.1.1 User Task

Der folgende User Task wurde formuliert, um die Funktionalität des Plugin Protypen aus Nutzersicht zu beschreiben:

UT1: Obtain distinct requirements
Eliminate ambiguity in requirement documents.

Refactor association of requirements to features, to better encapsulate crosscutting behaviour.

Der User Task unterteilt sich in die folgenden Subtasks, die die Funktionalität des Plugin Prototypen kleinteiliger beschreiben.

UT1S: Identify requirements
Identify ambigious requirements.
Example of solution: Analyze requirements to detect crosscutting concerns.

UT1S: Relate requirement to feature
Relate requirement to the corresponding feature in order to maintain traceability.
Example of solution: Provide feature-tag for crosscutting concern.

5.1.2 Systemfunktionen

Die folgenden Systemfunktionen setzen die in den obigen Subtasks beschriebene Funktionalität um:

SF: Show user stories
Present all user stories to the user.

SF: Show feature-tags for user story
Show all feature-tags that are attached to a given requirement.

SF: Select user stories
Provide an interface to allow the user to choose user stories for the analysis.
The selected user stories are passed on to the POS-Tagger.

SF: POS-tag user stories
POS-tag all user stories, the user selected to prepare them for the analysis methods.

SF: Identify Crosscutting Concerns by noun-analysis
Analyse chosen and POS-tagged user stories with the noun-analysis-algorithm.

SF: Identify Crosscutting Concerns by verb-analysis
Analyse chosen and POS-tagged user stories with the verb-analysis-algorithm.

SF: Show Crosscutting Concerns
After the analysis, show all identified Crosscutting Concerns to the user in a list.

SF: Remove unfit user stories
Delete unwanted or inapt user stories from the list of crosscutting concerns to aquire a clear overview.

SF: Update feature-tag
Update the feature-tag of a Crosscutting Concern with the feature-tag of its corresponding user story.

5.1.3 Nichtfunktionale Anforderungen

Im Rahmen der Anforderungserhebung wurden drei nichtfunktionale Anforderungen definiert:

NFR: Extensibility
The system provides an interface for other Crosscutting Concern-identification-techniques to be implemented.

Die nichtfunktionale Anforderung *Erweiterbarkeit* garantiert, dass der Plugin Prototyp um eine weitere Identifikationstechnik erweitert werden kann. Das ist sinnvoll, damit in das Werkzeug zu einem späteren Zeitpunkt eine qualitativ bessere oder performantere Technik integriert werden kann.

NFR: Performance
The Process of the identification should be automatic and fast. The automatic identification should not take longer than 1 second for every user story analyzed.

Die nichtfunktionale Anforderung *Performanz* beschränkt die Ausführungsdauer der Analyse. Bereits in Abschnitt 4.4 wurde die Sinnhaftigkeit einer Zeitbegrenzung diskutiert.
Bei der Umsetzung des Identifizierungsalgorithmus hat sich herausgestellt, dass durch das POS-Tagging die meiste Zeit in Anspruch genommen wird. Aus diesem Grund konnte der Aufwand linear abgeschätzt werden und eine sinnvolle Zeitgrenze definiert werden.

NFR: Usability
The operation of the tool should be easy and only require the user to make basic decisions, in terms of which requirements to analyze and what Crosscutting Concerns to accept.
Also the determination of the Crosscutting Concerns should be easy and require no domain-specific knowledge regardless of project size. This means the results should be presented in a clear manner.

Die nichtfunktionale Anforderung *Benutzbarkeit* wurde formuliert, um eine einfache Funktionsweise des Werkzeugs sicherzustellen. Hierbei wurde sowohl Wert auf die einfache Auswahl der Crosscutting Concerns, als auch auf die der User Stories selbst gelegt. Diese Benutzbarkeit ist bei beliebiger Projektgröße zu gewährleisten.

5.1.4 Workspaces

Das Werkzeug umfasst zwei Ansichten, die beide in die Hauptansicht von Jira integriert sind.

WS1: JIRA General View
This is the Jira general view, where all issues are shown.

WS1.1: Crosscutting Concern Analyzing View
In the Crosscutting Concern Analyzing View the user ist guided through the user story analysis.
The user is able to select the desired user stories.

In der ersten Sicht WS1.1 werden alle User Stories angezeigt (Systemfunktionen *Show user stories* und *Show feature-tags for user story*) und es ist dem Benutzer möglich einige oder alle User Stories zur Analyse auszuwählen (Systemfunktion *Select user stories*).

WS1.2: Crosscutting Concern Results View
In the Crosscutting Concern Results View the user is presented with the identified Crosscutting Concerns and has the option to update their feature tags.

Nach der Durchführung der Analyse wird die zweite Sicht WS1.2 gezeigt, in der dem Nutzer alle identifizierten Crosscutting Concerns angezeigt werden (Systemfunktion *Show Crosscutting Concerns*) und die Möglichkeit besteht einzelne Concerns aus einem Crosscutting Concern zu entfernen (Systemfunktion *Remove unfit Concerns*) und die Feature Tags der einzelnen User Stories anzupassen (Systemfunktion *Update feature-tag*).
Die Systemfunktion *POS-tag user stories*, *Identify crosscutting concerns by noun-analysis* und *Identify crosscutting concerns by verb-analysis* sind in den beiden Workspaces nicht umgesetzt, da sie intern bei der Analyse durchgeführt werden und so keine Interaktion mit dem Nutzer stattfindet.
Der Kontext der beiden Sichten ist in nachfolgendem UI-Strukturdiagramm in Abbildung 5.1 visualisiert.

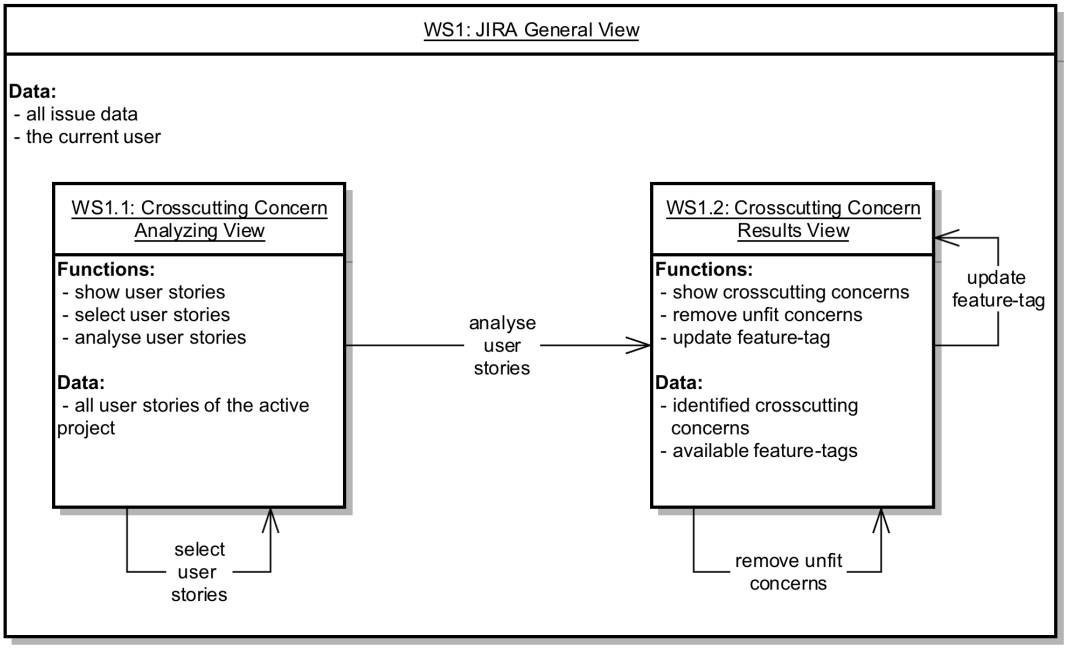

Abbildung 5.1: UI-Strukturdiagramm des FElICiTy Prototypen

5.1.5 Domänendaten

Die Beziehungen zwischen den Domänendaten des Plugin Prototypen sind in folgendem Domänendatendiagramm in Abbildung 5.2 zu entnehmen.

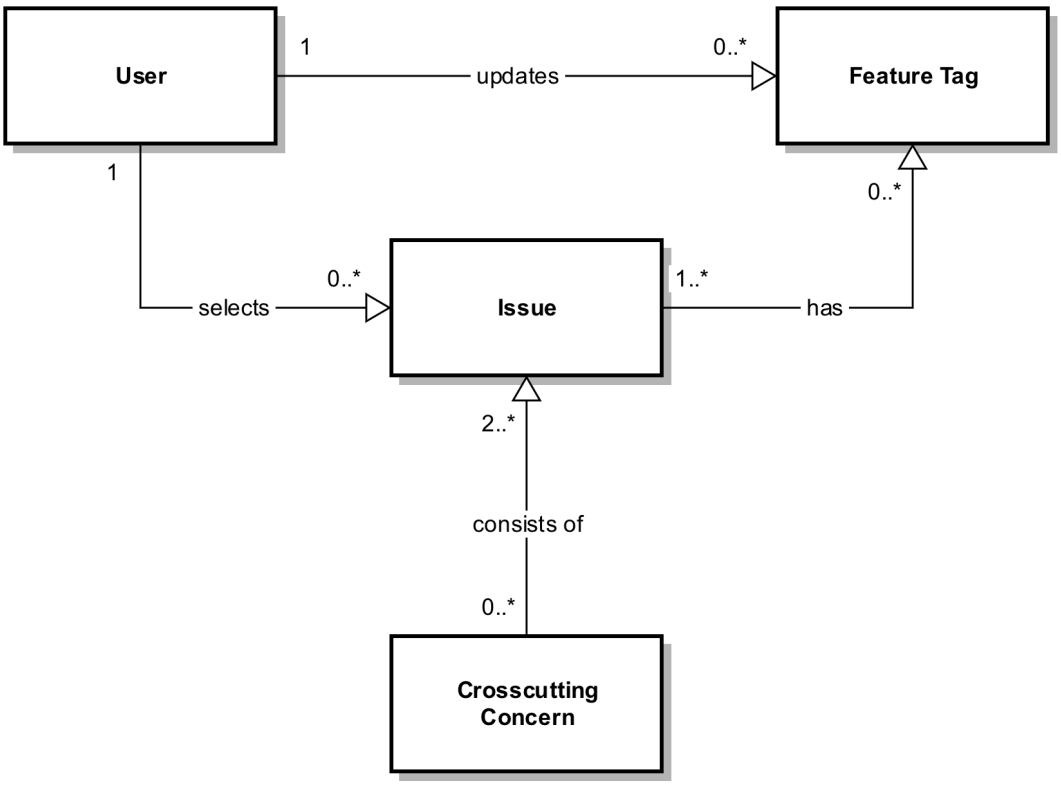

Abbildung 5.2: Domänendatendiagramm des FElICiTy Prototypen

5.2 Entwurf

Abschnitt 5.2.1 beschreibt das Kommandozeilenwerkzeug, welches vor der Implementierung des Plugin Prototypen entwickelt wurde. Abschnitt 5.2.2 beschreibt die Klassen, die zur Realisierung des Werkzeugs nötig waren.

5.2.1 Kommandozeilenwerkzeug

Vor der Implementierung des FElICiTy Prototypen wurde im Rahmen einer Kommandozeilenanwendung der gewählte 3CI-Ansatz testweise umgesetzt.
Hierbei konnten erste Optimierungen des Ansatzes in einem isolierten Umfeld vorgenommen werden.
Zu Implementierung der Kommandozeilenanwendung wurde die Programmierspra-

che Swift gewählt, da diese über einen proprietären Natural Language Processor[4] verfügt. Damit war die problemlose Anbindung der POS-Tagging Funktionalität gewährleistet. An diesem Punkt wurde bereits ersichtlich, dass zur erfolgreichen Identifikation von Crosscutting Concerns in größeren Projekten ein zusätzlicher Algorithmus entwickelt werden muss. Die genaue Funktionsweise dieses Algorithmus wird nachfolgend in Abschnitt 5.3.2 erläutert.

5.2.2 Klassen

Nach der Dokumentation der Anforderungen und der in Abschnitt 5.2.1 geleisteten Vorarbeit, wurde entschieden die Funktionalität des FElICiTy Prototypen in vier Klassen aufzuteilen.

C: AnalysisServlet
This class implements interfaces to the analysis classes and processes the results.

Die Klasse *AnalysisServlet* handhabt die Kommunikation mit der Analysesicht und der Ergebnissicht des Werkzeugs. Außerdem gibt sie die Daten der Analyse sowohl an die Sichten, als auch an die Analysealgorithmen weiter. Damit setzt sie die Systemfunktionen *Show user stories*, *Show feature-tags for user story*, *Select user stories*, *Show Crosscutting Concerns*, *Remove unfit Concerns* und *Update feature-tag* um. Die Handhabung dieser Daten zwischen den beiden Sichten erfordert die Sichtbarkeit der Daten innerhalb des Programms. Dies rechtfertigt die Umsetzung der oben genannten Systemfunktionen in einer Klasse.
Die Klasse *AnalysisServlet* ist eine Subklasse der Klasse *HttpServlet*. Dies ist nötig, da dadurch alle Funktionalitäten, die nötig sind, um Daten auf der Benutzeroberfläche in Jira anzuzeigen, zur Verfügung gestellt werden. Aus diesem Grund besitzt die Klasse eine Reihe von privaten Attributen, die nötig sind, um mit der Issue-Datenbank zu kommunizieren und Änderungen an ebendieser durchzuführen.

C: POSTagger
This class implements the POS-Tagger interface.

Die Klasse POSTagger ist für das POS-Tagging der User Story Beschreibungen zuständig. In ihr wird mit der Methode *tag* die Systemfunktion *POS-tag user stories* umgesetzt. Da hierfür eine externe Bibliothek angebunden wird, wurde diese

[4] `https://developer.apple.com/documentation/foundation/nslinguistictagger`

Funktionalität in einer eigenen Klasse realisiert. Dadurch ist die Kapselung sichergestellt und eine richtige Funktionsweise kann durch gezieltes Testen möglichst leicht garantiert werden.

C: VerbAnalyzer
This class implements the verb analysis algorithm.

C: NounAnalyzer
This class implements the noun analysis algorithm.

Die Klassen *VerbAnalyzer* und *NounAnalyzer* implementieren die beiden Analysealgorithmen. Somit ist hier in der Methode *analyze* jeweils die Systemfunktion *Identify Crosscutting Concerns by noun-analysis* beziehungsweise *Identify Crosscutting Concerns by verb-analysis* umgesetzt. Die genaue Funktionsweise ist in Abschnitt 5.3.2 dargelegt.
Die Abhängigkeiten der einzelnen Klassen sind nachfolgend dem Klassendiagramm in Abbildung 5.3 zu entnehmen.

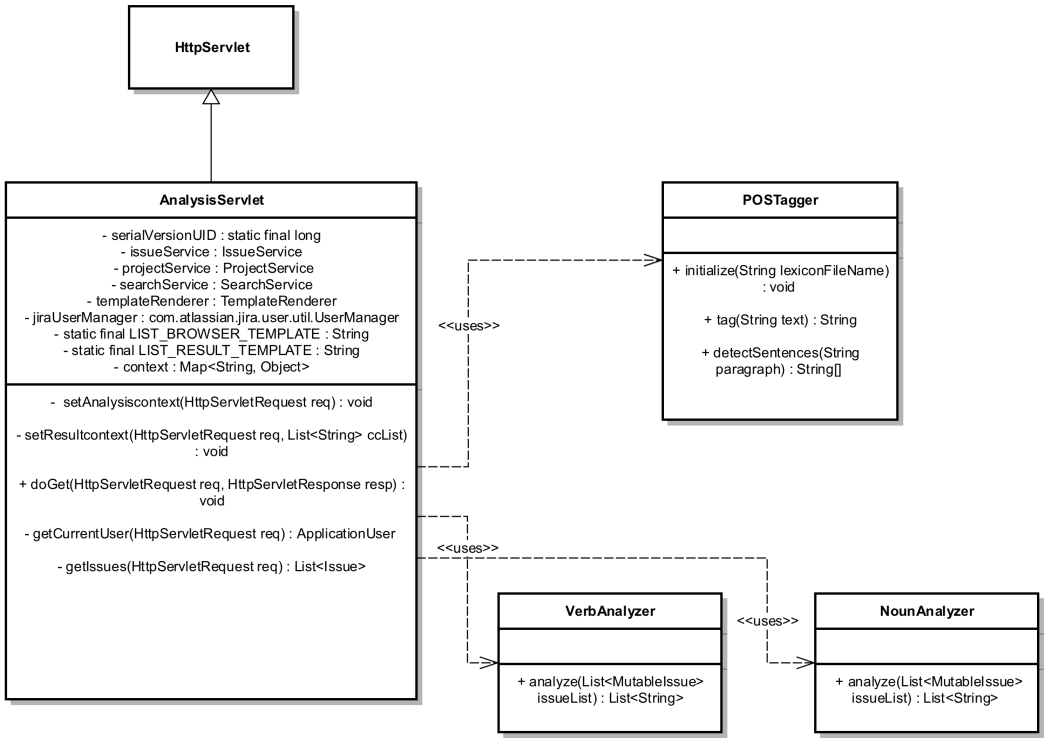

Abbildung 5.3: Klassendiagramm des FElICiTy Prototypen

5.3 Implementation

Der Java-, Apache Velocity-, JavaScript und XML-Quellcode des Plugins wurde in Intellij IDEA[5], Version 2018.2.5, verfasst. Zum Kompilieren wurde Apache Maven[6], Version 3.2.1, beziehungsweise die Atlassian Plugin SDK[7], Version 6.3.10, mit der Java Version 1.8.0_172 verwendet.
Die folgenden Abschnitte gehen auf die Details der Implementierung ein. Abschnitt 5.3.1 befasst sich mit der Analysesicht des Werkzeugs. Abschnitt 5.3.2 geht auf die genaue Funktionsweise der verwendeten Algorithmen ein. Abschnitt 5.3.3 beschreibt den Programmfluss des Plugin Prototypen. Schließlich erläutert Abschnitt 5.3.4 den Aufbau der Ergebnissicht.

[5] `https://www.jetbrains.com/idea/`
[6] `https://maven.apache.org/`
[7] `https://developer.atlassian.com/server/framework/atlassian-sdk/`

5.3.1 Implementierung der Analysesicht

In der Analysesicht werden dem Nutzer alle im Projekt vorhandenen User Stories gezeigt. Um die Darstellung möglichst übersichtlich zu gestalten, wurde entschieden, die User Stories in einer Tabelle darzustellen. Hier werden alle relevanten Daten der User Stories angezeigt. Als relevant im Kontext dieser Arbeit werden die Attribute *Issue Key*, *Summary*, *Description* und *Feature Tag* erachtet. Es besteht die Möglichkeit, mit Klick auf den Issue Key direkt zur ausgewählten User Story zu gelangen.
Der Nutzer kann im rechten Bereich der Tabelle mit Klick auf die entsprechende Checkbox einzelne User Stories zur Analyse an- und abwählen. Um Effizienz und Benutzbarkeit der Auswahl zu gewährleisten, ist es außerdem an gleicher Stelle möglich, alle User Stories auf einmal auszuwählen.
Die Analyse wird mit Klick auf die Schaltfläche *Start Analysis* gestartet. Die Schaltfläche wurde über der Tabelle angebracht, um die Übersichtlichkeit auch bei großen Projekten mit vielen User Stories zu wahren. Ist die Analyse gestartet, zeigt eine Ladeanimation dem Benutzer an, dass das Werkzeug arbeitet.
Abbildung 5.4 zeigt die Analysesicht.

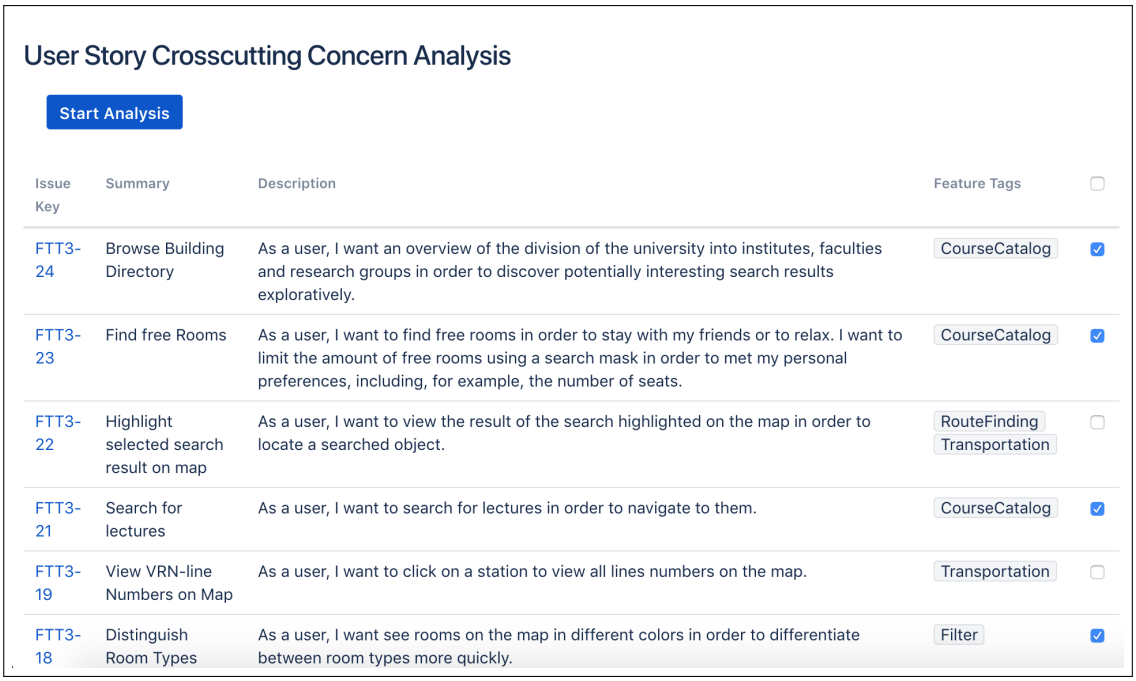

Abbildung 5.4: Analysesicht des FEllCiTy Prototypen

5.3.2 Der Identifizierungsalgorithmus

Bei der Implementierung wurden die bereits lauffähigen Algorithmen aus der Kommandozeilenanwendung entnommen.
Eine Schwierigkeit stellte die Wahl der NLP-Bibliothek dar. Es sollte zunächst der Stanford Natural Language Processor[8] verwendet werden. Dieses Vorhaben musste allerdings wieder verworfen werden, da die Anbindung an das Werkzeug als Maven Dependency nicht möglich war. Als Alternative wurde Apache OpenNLP[9] gewählt, dessen Anbindung keine Probleme darstellte. Beide Prozessoren stellen für das Werkzeug genug Funktionalität zur Verfügung. Die einzelnen Wörter werden in der Form "Wort_Tag" gespeichert, da diese Form eine einfache Weiterverarbeitung ermöglicht.

Der Algorithmus der Verbanalyse arbeitet mit der gleichen Funktionsweise des 3CI-Ansatzes [2]. Zunächst werden alle Anforderungen geordnet eingelesen. Als nächstes werden alle Beschreibungen der User Story mit dem POS-Tagger syntaktisch verarbeitet. Dann werden alle Verben in den Anforderungen identifiziert und eine Liste erstellt. Diese Liste wird mit allen Beschreibungen der User Stories abgeglichen und Verben, die in mehr als einer User Story vorkommen, zusammen mit der korrespondierenden User Story gespeichert. Diese Liste an User Stories wird gespeichert und zur Ausgabe aufbereitet.

Der Algorithmus der Substantivanalyse arbeitet nach einem ähnlichen Prinzip. Zunächst wird wiederum POS-Tagging auf allen User Stories durchgeführt.
Der Abschnitt nach dem Schlüsselwort *in order to* beziehungsweise *so that* der User Story wird als imperativer Teil bezeichnet. Dieser Teil der User Story wird herausgefiltert und zur Analyse herangezogen. Dabei werden paarweise alle User Stories miteinander verglichen und alle Beschreibungen, die mehr als eine festgelegte Zahl an gemeinsamen Substantiven besitzen, in die Ergebnismenge aufgenommen. Diese Anzahl wurde durch eine empirische Studie ermittelt, bei der sich herausstellte, das je nach Projektgröße eine unterschiedliche Anzahl zur besten Ergebnisqualität führt. Auf dieser Grundlage wurde die Anzahl an gemeinsamen Substantiven bei einer Projektgröße von maximal 20 User Stories auf zwei, bei maximal 59 User Stories auf drei und ab 60 User Stories auf vier festgelegt.
Der Vergleichsalgorithmus der Klasse NounAnalyzer ist in Codebeispiel 1 abgebildet:

[8] `https://nlp.stanford.edu/software/`
[9] `https://opennlp.apache.org/`

```
for(Issue issue: issueList){
    String[] referenceArray = issue.getDescription().split(" ");
    List<String> tempCountArray = new ArrayList<>();
    for(Issue issue2 : issueList) {

        List<String> tempWordArray = new ArrayList<>();

        String[] compareArray = issue2.getDescription().split(" ");

        for (String referenceWord : referenceArray) {
            for (String compareWord : compareArray) {
                if (compareWord.equals(referenceWord)
                    && !compareWord.isEmpty()) {
                    tempWordArray.add(compareWord);
                }
            }
        }
        if (tempWordArray.size() >= commonNounCount) {
            tempCountArray.add(issue2.getKey());
        }
        tempWordArray.clear();


    }
    if (tempCountArray.size() > 1) {
        commonNounKeys.add(String.join(" ", tempCountArray));
    }
}
```

Quellcode 1: Vergleichsalgorithmus der Klasse NounAnalyzer

Das ursprüngliche Vorhaben, die Ergebnismenge aus der Schnittmenge der Ergebnisse der beiden Algorithmen zu errechnen, musste wieder verworfen werden, da beide Ergebnismengen oftmals disjunkt sind. Außerdem hat sich eine unterschiedliche Effektivität der beiden Analyseansätze abhängig von der Projektgröße gezeigt. Diese Abhängigkeit wird eingehend in Abschnitt 6.3 behandelt. Aus diesem Grund wurde die Verknüpfung der beiden Analysen nicht umgesetzt.

Sowohl der Vergleichsalgorithmus der Klasse *NounAnalyzer*, als auch der der Klasse *VerbAnalyzer* arbeiten direkt mit dem Dateityp *Issue*. Da mehrere Operationen mit der Beschreibung der User Story durchführt werden, kann so sichergestellt werden, dass die Verlinkungen zwischen *Issue Key* und *Description* der User Story nicht verloren geht.

5.3.3 Programmablauf

Die Identifikation von Crosscutting Concerns in Jira wird durch den Nutzer angestoßen. Dieser hat anschließend die Möglichkeit einzelne User Stories zur Analyse auszuwählen. Diese Daten werden an die Klasse *AnalysisServlet* weitergegeben.

Dort wird die Instanziierung der *POSTagger*, *NounAnalyzer* und *VerbAnalyzer* Klassen vorgenommen. Schließlich werden die ermittelten Ergebnisse via *AnalysisServlet* an die entsprechende Sicht weitergegeben.
Abbildung 5.5 modelliert diesen Ablauf als Sequenzdiagramm.

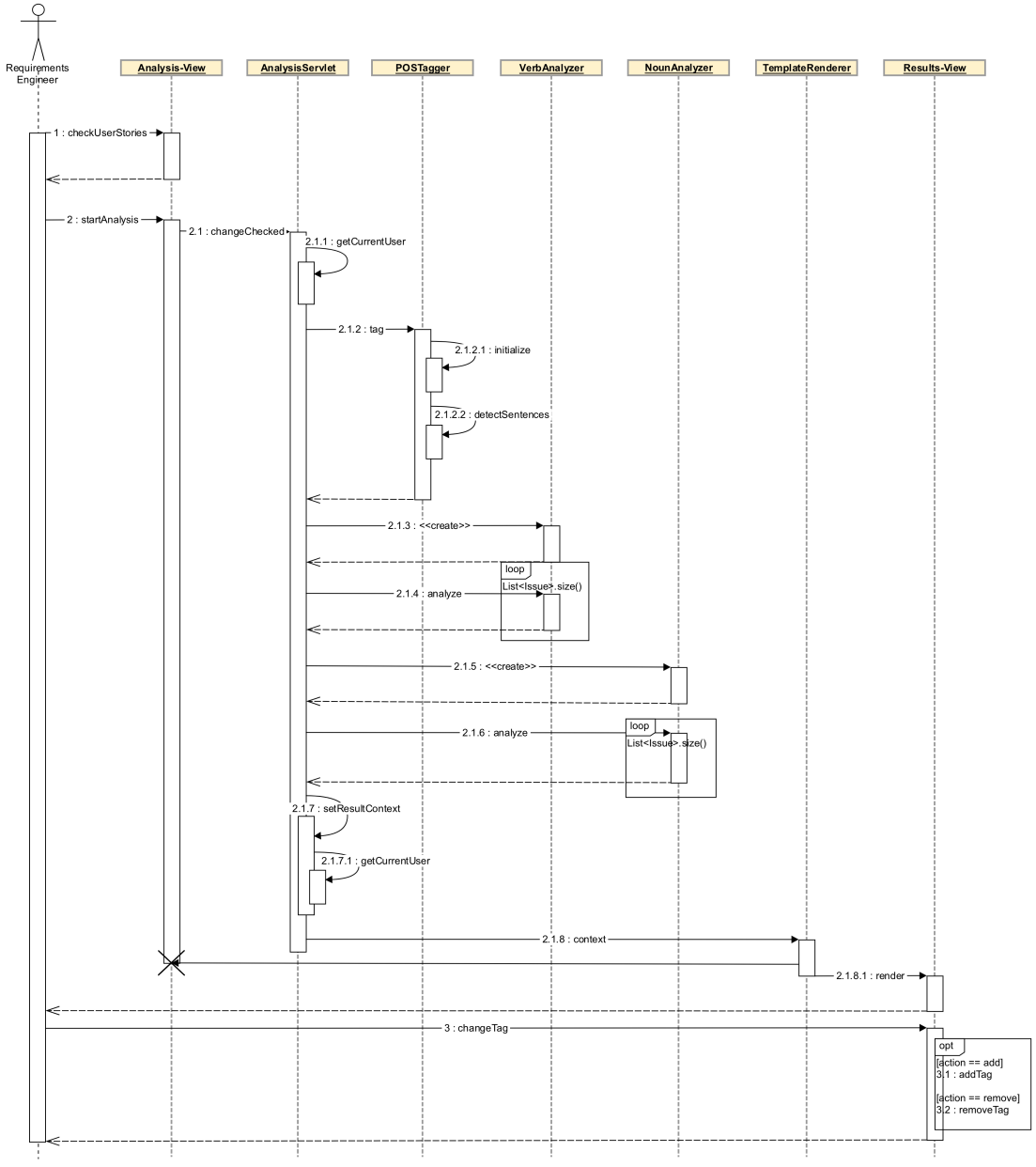

Abbildung 5.5: Sequenzdiagramm des FEllCiTy Prototypen

5.3.4 Implementierung der Ergebnissicht

Die Implementierung der Ergebnissicht folgte den gleichen Entwurfsprinzipien, unter denen auch die Analysesicht konzipiert wurde.
Es wurde wiederum eine Tabelle gewählt, um die Ergebnisse der Analyse übersichtlich darzustellen.
Einzelne Crosscutting Concerns werden in der Tabelle durch einzelne Sektionen abgegrenzt, sodass die übersichtliche Darstellung gewährleistet ist.
Zusätzlich zu den Informationen aus der Analysesicht ist es dem Benutzer möglich, in der linken Spalte einzelne Zeilen und somit einzelne User Stories aus dem entsprechenden Crosscutting Concern zu löschen.
Außerdem hat der Nutzer in der rechten Spalte die Möglichkeit, im Rahmen eines Crosscutting Concerns das Feature Tag einer User Story mit dem Feature Tag einer abhängigen User Story zu versehen. Das erfolgreiche Hinzufügen sowie Entfernen wird durch eine kurze Einblendung, dargestellt in Abbildung 5.7, deutlich gemacht.
Abbildung 5.6 zeigt die allgemeine Ergebnissicht; Abbildung 5.7 zeigt das Anpassen eines Feature Tags einer User Story.

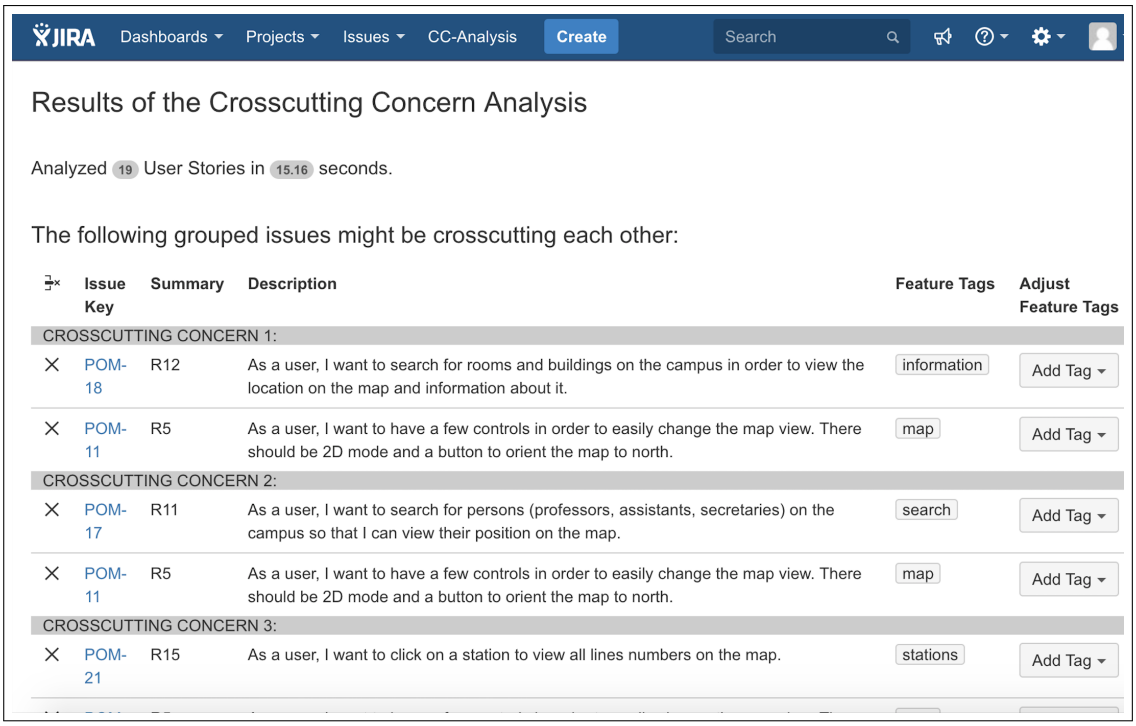

Abbildung 5.6: Ergebnissicht des FEllCiTy Prototypen

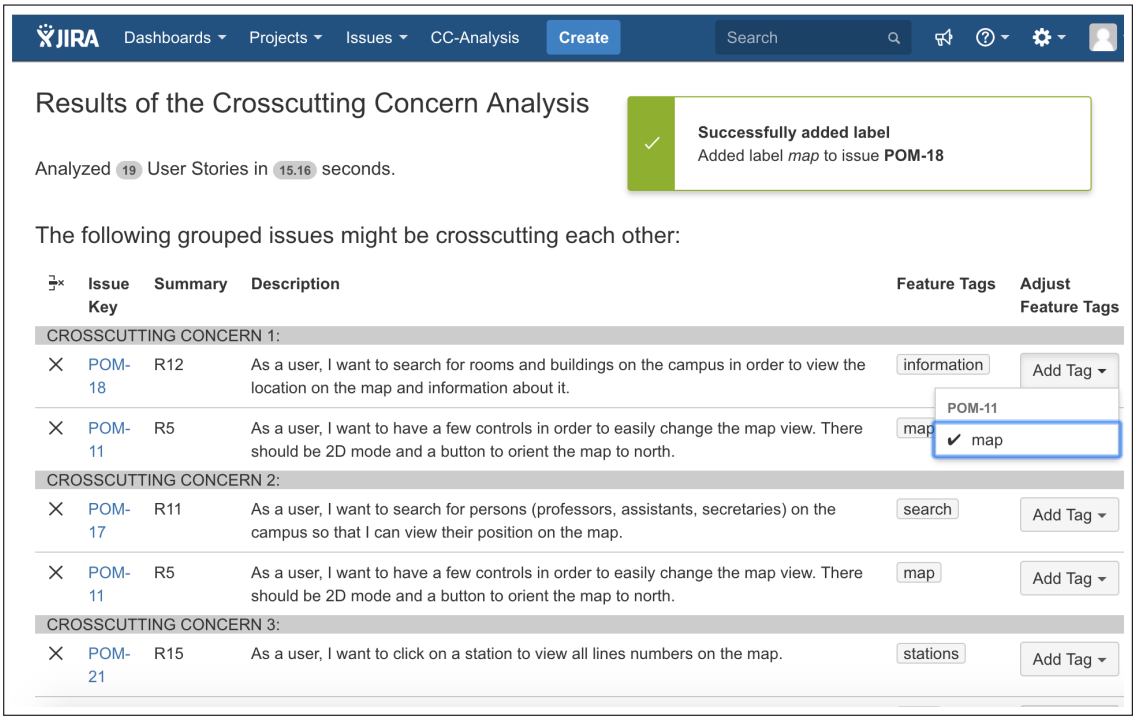

Abbildung 5.7: Hinzufügen eines Feature Tags in der Ergebnissicht

5.4 Qualitätssicherung

Abschnitt 5.4.1 nennt alle bei der Qualitätssicherung verwendeten Werkzeuge. Abschnitt 5.4.2 behandelt die angewandte Teststrategie und beschreibt die Einzelheiten der Testausführung. Abschnitt 5.4.3 beschreibt die Ergebnisse des Testens.

5.4.1 Testwerkzeuge

Zur Abdeckung der logischen Testfälle auf Komponententestebene wurde JUnit 4[10] benutzt, welches mithilfe von Mockito[11] implementiert wurde. Mockito wurde verwendet um instanziierte Issue-Objekte mit Beispieldaten zum Testen auszustatten. Zur statischen Codeanalyse wurden PMD[12] benutzt. PMD wurde vor allem gewählt, da es neben den gängigen Sprachen Java und XML auch Velocity unterstützt. Selenium[13] wurde verwendet, um im Webbrowser automatisierte und wiederholbare Tests durchführen zu können.

[10] `https://junit.org/junit4/`
[11] `https://site.mockito.org/`
[12] `https://pmd.github.io/`
[13] `https://www.seleniumhq.org/`

Für den Performanztest wurde neben der Systemzeitmessung von Java (Methode *System.nanoTime*) Pagespeed[14] verwendet.
Das Werkzeug wurde als Plugin auf dem Jira-Server des Continuous Usage- and Rationale-based Evolution Decision Support (CURES) Projekts installiert, um es unter realen Bedingungen testen zu können.

5.4.2 Teststrategie

Um die korrekte Funktionalität und die Erfüllung aller nichtfunktionalen Anforderungen zu gewährleisten wurde folgende Teststrategie (Tabelle 8) konzipiert:

Nr.	Teststufe	Testobjekt	Testziel	Testart
1	Komponententest	Operation analyze der Klasse NounAnalyzer bzw. VerbAnalyzer	Funktionalität: Richtigkeit	Dynamischer Test mittels JUnit
2	Systemtest	Alle Systemfunktionen in den Workspaces des FElICity Prototyps	Funktionalität: Richtigkeit	Dynamischer Test mit Selenium
3	Systemtest	FElICity Prototyp auf dem CURES Server	Funktionalität: Angemessenheit	Dynamischer Test: Programmablauf durch Persona-Simulation
4	Systemtest	FElICity Prototyp auf dem CURES Server	Effizienz: Performanz	Dynamischer Test: Laufzeitüberprüfung; Ladezeit einer Webanwendung
5	Komponententest	Quellcode des FElICity Prototyp	Funktionalität: Richtigkeit	Statische Analyse mit PMD

Tabelle 8: Teststrategie für den FElICiTy Prototyp

Alle relevanten Komponenten des Plugins, konkret die Funktionsweise des Identifizierungsalgorithmus, wurden mithilfe von Komponententests auf Richtigkeit geprüft. Dafür wurden logische Testfälle abgeleitet und die Algorithmen mit minimalen Eingangsdaten mittels JUnit getestet, um sicherzustellen, dass die Korrelation der einzelnen Satzteile korrekt erkannt und dargestellt wird.
Es wurden Seleniumtests implementiert, um alle Systemfunktionen, die in einem der beiden Workspaces aus Abschnitt 5.1.4 angesiedelt sind, zu testen. Konkret sind das die Systemfunktionen *Show user stories*, *Show feature-tags for user story*, *Select user stories*, *Show Crosscutting Concerns*, *Remove unfit user stories* und *Update feature-tag*.

[14] `https://developers.google.com/speed/pagespeed/insights/`

Es wurde ein Benutzbarkeitstest auf Systemebene durchgeführt, um die Einhaltung der nichtfunktionalen Anforderung Benutzbarkeit zu garantieren. Dazu wurde eine Persona konzipiert, um davon ausgehend mögliche Anwendungsfälle zu definieren. Bei der Erstellung der Persona wurde Wert darauf gelegt, den fachlichen Hintergrund ausreichend zu beschreiben. Außerdem wurde ein realistischer Kontext erstellt, aus dem der bestehende Zeitdruck für den Anforderungsingenieur ersichtlich wird. Die erstellte Persona ist im Folgenden einzusehen:

P: Julia Williams
Name: Julia Williams
Age: 34 years old
Occupation: Requirements Engineer at SAP
– Works at SAP, supervises multiple software projects at once.
– Doesn't always have accurate and up-to-date knowledge about the software system.
– Has to guarantee that all supervised software projects contain distinct and well written requirements.
Use-Case 1: Julia recently had to add a subset of new user stories which describe new functionality to a project she supervises. Now she wants to analyze these newly added user stories to see if a crosscutting concern between these requirements exists.
Use-Case 2: Julia is tasked to check the entirety of user stories of a project she supervises for crosscutting concerns. Due to the scope of her responsibilities, this is the first time she visits it in weeks. After she identifies all crosscutting concerns, she wants to maintain the traceability in the project by adding appropriate feature-tags to some user stories.
Use-Case 3: Julia recently refactored some user stories, which describe old functionality. Now she wants to check if crosscutting concerns exist between these user stories and the rest of the project. For that reason she only is interested in crosscutting concerns caused by her newly edited user stories. After she identifies such a crosscutting concern, she wants to change the feature-tag of another user story. While changing, she makes a mistake and changes the wrong feature-tag instead. She now wants to revert the changes.

Es wurde sich dagegen entschieden, die nichtfunktionale Anforderung Erweiterbarkeit separat zu testen. Bei der Implementierung wurde bereits dafür Sorge getragen, dass die Umsetzung des Ansatzes gekapselt in einzelnen Klassen stattfindet und ein klarer Einstiegpunkt für den Identifizierungalgorithmus gegeben ist. Diese Funktionsweise ist außerdem dem Klassendiagramm in Abbildung 5.3 und dem

Sequenzdiagramm in Abbildung 5.5 des Prototypen zu entnehmen. Somit ist die Erweiterbarkeit des Plugins um einen neuen Ansatz bereits gewährleistet.
Es wurde ein Performanztest durchgeführt, um die gleichnamige NFR zu garantieren. Dafür wurde in Jira eine Laufzeitanalyse des Identifikationsalgorithmus in verschiedenen Konfigurationen durchgeführt. Außerdem wurde die Ladezeit der entsprechenden Website mit Pagespeed gemessen.
Es wurde eine statische Analyse des Quellcodes mithilfe des Werkzeugs PMD durchgeführt, um ungenutzte Imports und Variablen zu erkennen und den Programmfluss zu optimieren.

5.4.3 Testergebnisse

Insgesamt wurden beim Testen 39 Testfälle in 14 Testläufen durchgeführt. Hierbei wurden 6 Fehler entdeckt, die alle nach Verbesserung und erneuter Ausführung des Testlaufs behoben werden konnten.
Beim Komponententest des Identifizierungsalgorithmus mit JUnit wurden 6 Testfälle in 2 Testläufen realisiert. Hierbei wurden keine Fehler entdeckt. Dies ist allerdings auch damit zu begründen, dass der Algorithmus im Kontext des Kommandozeilenwerkzeugs bereits implementiert und optimiert wurde.
Bei den Tests mit Selenium wurden 13 Testfälle in 6 Testläufen durchgeführt. Aus der Systemfunktion *Show user stories* wurden 2 Testfälle abgeleitet. Zunächst wurde der Normalfall getestet, dann ein Fall bei dem keine User Stories im Projekt vorliegen. Aus der Systemfunktion Show *feature-tags for user story* wurden ebenfalls 2 Testfälle abgeleitet. Zunächst liegt eine User Story mit Feature Tag vor, danach eine User Story ohne Feature Tag. Die Systemfunktion *Select user stories* wurde mit 4 Testfällen getestet. Dabei wurde zwischen der Auswahl von einer User Story, mehrerer User Stories, aller User Stories und keiner User Story unterschieden. Aus der Systemfunktion Show CC wurde ein Testfall abgeleitet, der die Anzeige der Crosscutting Concerns prüft. Die Systemfunktion *Remove unfit user stories* wurde mit 2 Testfällen getestet: Das Entfernen einer User Story und das Entfernen aller User Stories aus einem Crosscutting Concern. Die Systemfunktion *Update feature-tag* wurde ebenfalls mit 2 Testfällen auf Richtigkeit geprüft: Hierbei wurde zwischen Hinzufügen und Entfernen eines Feature Tags entschieden. Diese Testfälle sind zunächst fehlgeschlagen, da das Hinzufügen und Löschen eines Feature Tags nicht korrekt an die Datenbank von Jira weitergegeben wurde und so die Persistenz der Änderung nicht gewährleistet war. Durch korrekte Ansteuerung der Datenbank konnte dieser Fehler behoben werden, wodurch der Testfall bei erneuter Durchführung erfolgreich war. Die Implementierung der Selenium-Tests und deren erfolgreiche Durchführung, welche durch die grüne Schrift gekennzeichnet ist, ist in Abbildung 5.8 einsehbar.

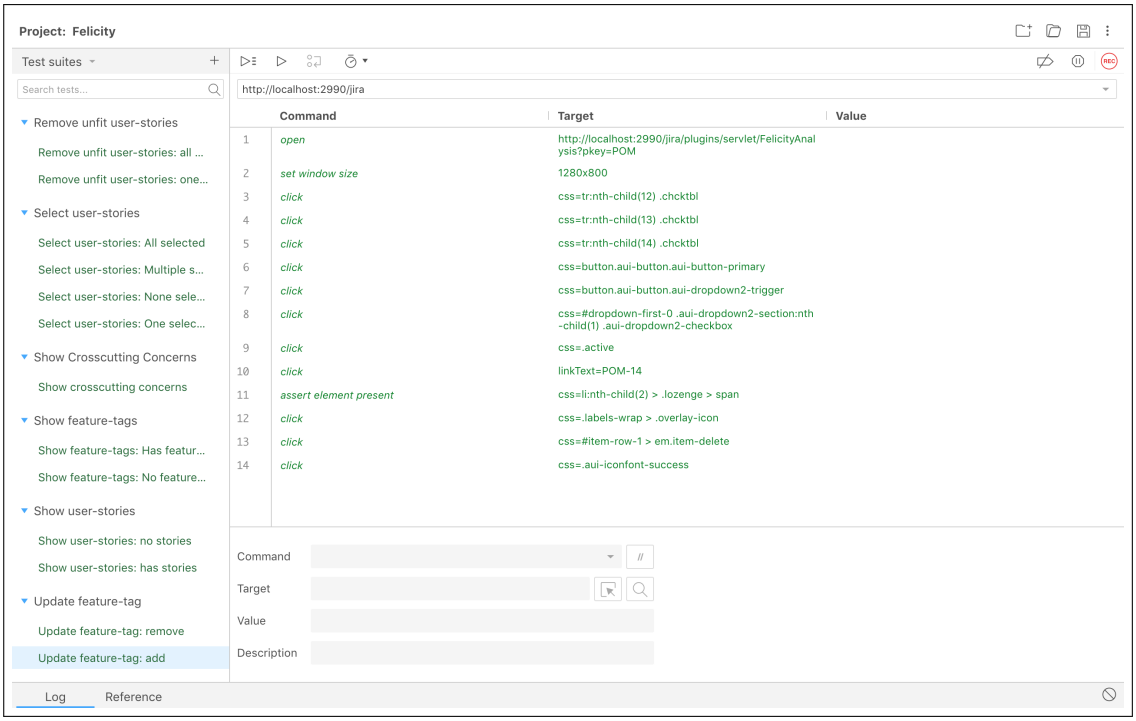

Abbildung 5.8: Selenium: Ansicht der Testläufe

Beim Benutzbarkeitstest wurden 7 Testfälle in 3 Testläufen durchgeführt. Aus Anwendungsfall 1 der Persona wurden 2 Testfälle abgeleitet. Hier wurden die Möglichkeiten betrachtet, ein Crosscutting Concern zwischen den neuen User Stories liegt vor oder es kein Crosscutting Concern liegt vor. Beide dieser Testfälle waren bei erster Durchführung erfolgreich. Aus Anwendungsfall 2 ergaben sich 4 Testfälle: Ein identifizierter Crosscutting Concern besteht aus zwei User Stories, ein identifizierter Crosscutting Concern besteht aus mehr als zwei User Stories, eine im Crosscutting Concern vorliegende User Story besitzt noch kein Feature Tag und eine im Crosscutting Concern vorliegende User Story besitzt bereits mehr als ein Feature Tag. Zwei dieser Testfälle sind zunächst fehlgeschlagen, da die Änderung eines Feature Tags zu umständlich war. Dieser Fehler konnte durch die Verwendung eines Drop-Down-Menüs bei der Auswahl behoben werden. Dadurch konnte die benötigte Anzahl an Klicks, um ein Feature Tag zu ändern, auf 2 gesenkt werden, sodass die Testfälle bei der zweiten Durchführung erfolgreich waren. Aus Anwendungsfall 3 wurde ein Testfall abgeleitet. Dieser Testfall schlug aus gleichem Grund wie die beiden vorher angeführten fehl und war somit ebenfalls, nach der Verbesserung des Auswahlvorgangs, bei erneuter Durchführung erfolgreich.
Beim Performanztest wurden 12 Testfälle in 2 Testläufen durchgeführt. Hierzu

wurde das Plugin in unterschiedlichen Konfigurationen getestet und jeweils die Ausführungsdauer gemessen. Die Ergebnisse dieser Testläufe sind nachfolgend in Tabelle 9 einsehbar. Somit konnte im Durchschnitt eine Ausführungsdauer von 0,69 Sekunden pro Anforderung ermittelt werden, was den in der nichtfunktionalen Anforderung Performanz festgelegten Höchstwert unterschreitet.

Anzahl an Anforderungen	**Dauer in Sekunden**	**Ausführungsort**	**Dauer pro Anforderung in Sekunden**
18	13,44	lokal	0,75
18	11,57	lokal	0,64
19	12,40	lokal	0,65
19	13,99	CURES	0,74
19	13,87	CURES	0,73
19	13,08	CURES	0,69
10	6,35	lokal	0,64
10	7,53	lokal	0,75
10	6,65	lokal	0,67
10	6,76	CURES	0,68
10	6,91	CURES	0,69
10	6,84	CURES	0,68

Tabelle 9: Ergebnisse des Performanztests

Zur statischen Analyse des Quellcodes mit PMD wurde ein Testfall in einem Testlauf realisiert. Die Analyse wurde mit dem Regelset *best practices*[15] durchgeführt. Dieses Regelset wurde verwendet, da es einen guten Kompromiss zwischen der Menge an entdeckten Regelverstößen und deren Schwere darstellt. Wäre ein strengeres Regelset benutzt worden, wäre die Menge an Regelverstößen unübersichtlich hoch gewesen. Hierbei konnte die Sichtbarkeit einiger Variablen optimiert werden. Außerdem konnten mehrere Fehler bei der Variablenreferenzierung in den velocity-templates der beiden Sichten des Plugins identifiziert werden. Dies ist vor allem hilfreich, da in der Integrated development environment (IDE) die Syntaxhervorhebung beim Arbeiten an einer velocity-template mit JavaScript Elementen nicht immer eindeutig ist.

[15] `https://pmd.github.io/latest/pmd_rules_java_bestpractices.html`

6 Evaluation

In Abschnitt 6.1 wird der Goldstandard an Crosscutting Concerns für zwei Projekte erstellt. Dieser bietet die Vergleichsgrundlage für das implementierte Werkzeug, dessen Ergebnisse in Abschnitt 6.2 einzusehen sind. Anschließend werden die Ergebnisse in Abschnitt 6.3 bewertet.

6.1 Goldstandard

Bei der Erstellung des Goldstandards wurden manuell Crosscutting Concerns aus zwei Projekten ermittelt. Dabei wurden Anforderungen so ausgewählt, dass sie der Definition von Crosscutting Concerns laut Abschnitt 1.1 und Abschnitt 2.1 genügen. Dies sind konkret Anforderungen, die entweder ähnliche Funktionalität beinhalten oder in ihrer geforderten Funktionalität aufeinander aufbauen. Dazu wurden alle User Stories paarweise miteinander verglichen und auf Schlüsselwörter und Gemeinsamkeiten untersucht, welche zum Beispiel auf verknüpfte Funktionalität hindeuten.
Die genau Vorgehensweise soll an folgendem Beispiel verdeutlicht werden. Dazu werden Anforderung 2 und 3 von Testprojekt 2 betrachtet:

R2 As a user, I want to restrict the search to certain *categories* of *map objects* in order to get filtered search results.
R3 As a user, I want to apply *categories* to *map objects* in order to highlight filtered objects appropriately.

Hier wird als Schlüsselwort “map objects” in den beiden Anforderungen identifiziert. Zusätzlich werden die in Anforderung 3 erwähnten “categories” in Anforderung 2 benutzt um die “map objects” zu filtern. Daraus ist zu schließen, dass die in Anforderung 3 spezifizierte Funktionalität sich direkt auf Anforderung 2 auswirkt und hier deshalb ein Crosscutting Concern vorliegt.
Im Folgenden beschreibt Tabelle 10 den Goldstandard für Projekt 1, Tabelle 11 analog dazu den Goldstandard für Projekt 2.
Die Anforderungen der beiden Projekte sind dem Anhang in Abschnitt A.2.1 beziehungsweise Abschnitt A.2.2 zu entnehmen.

Anforderungen mit gemeinsamen Bestandteilen	**Funktionale Abhängigkeit**
Anforderung 8, Anforderung 16, Anforderung 28, Anforderung 34	"Create/Manage access levels"
Anforderung 1, Anforderung 43, Anforderung 44, Anforderung 46	"POST/JSON operations"
Anforderung 10, Anforderung 25, Anforderung 50, Anforderung 51, Anforderung 52	"Create/Edit Organizations"
Anforderung 39, Anforderung 41, Anforderung 42	"Search for patients"
Anforderung 18, Anforderung 20, Anforderung 21, Anforderung 24, Anforderung 25	"Delete Profile/User"
Anforderung 2, Anforderung 29, Anforderung 50, Anforderung 57	"Organisation location"
Anforderung 12, Anforderung 13, Anforderung 25, Anforderung 54	"Create/Edit Practitioner"
Anforderung 47, Anforderung 48	"Inserter ID"
Anforderung 5, Anforderung 11, Anforderung 25, Anforderung 53	"Create/Edit Patient"
Anforderung 58, Anforderung 59	"View all patientdata"
Anforderung 39, Anforderung 60	"Relationship between persons"
Anforderung 7, Anforderung 32, Anforderung 33	"Matching Profiles"

Tabelle 10: Projekt 1: Goldstandard

Anforderungen mit gemeinsamen Bestandteilen	**Funktionale Abhängigkeit**
Anforderung 2, Anforderung 3	"Map objects"
Anforderung 4, Anforderung 10, Anforderung 11, Anforderung 12, Anforderung 13, Anforderung 15	"View something on the map"
Anforderung 6, Anforderung 7, Anforderung 8	"Click and see information in infoBox"
Anforderung 6, Anforderung 14	"View information on a person"

Tabelle 11: Projekt 2: Goldstandard

6.2 Ergebnisse der Evaluation

Eine Problematik bei der Berechnung der Metriken ist die Definition eines richtigen Ergebnisses im Kontext der Crosscutting Concern Identifikation. Betrachtet man ein Crosscutting Concern, bestehend aus mehr als zwei Anforderungen, so gibt es zweierlei Möglichkeiten:
Die Berücksichtigung aller Kombinationen aus den Teilelementen des Crosscutting Concerns. Dafür ergibt sich bei einem Crosscutting Concern mit n Elementen folgende Formel:

$$\sum_{k=2}^{n} \binom{n}{k}$$

Hierbei ist zu bemerken, dass das teilweise Finden von Crosscutting Concerns unterbewertet, also ein fehlendes Element zu hoch eingeschätzt wird.
Ein Beispiel hierzu ist: Ein Crosscutting Concern besteht aus vier Elementen, drei davon werden korrekt identifiziert. Dadurch würde sich ein Recall von nur 36,3% ergeben. Zusätzlich steigt die Menge an möglichen Kombination bei der automatischen Analyse großer Projekte sehr schnell inflationär hoch an.
Beispielsweise ergeben sich bei einem aus sieben Elementen bestehenden Crosscutting Concern bereits 120 einzelne Kombinationsmöglichkeiten. Dadurch wird die errechnete Precision deutlich unterschätzt. Allerdings ist dieses Ergebnis nicht für die Güte des Resultats repräsentativ, da ein auf diese Art identifiziertes, mehrere Anforderungen umfassendes Crosscutting Concern durch den Anforderungsingenieur leicht korrigiert werden kann. Aus diesem Grund wurde entschieden, als zweite Möglichkeit gefundene Crosscutting Concerns als Ganzes und nicht alle möglichen Kombinationen zu betrachten. Somit ist ein Ergebnis als richtig zu werten, wenn es wenigstens als Teilmenge ein im Goldstandard identifiziertes Crosscutting Concern enthält.
Im Folgenden werden die durch das FElICiTy Werkzeug identifizierten Crosscutting Concerns und die entsprechenden Metriken in den beiden Projekten beschrieben. Hierbei sind die Ergebnisse von Projekt 1 in Tabelle 12 (Substantivanalyse) und Tabelle 13 (Verbanalyse) einzusehen; die Ergebnisse von Projekt 2 in Tabelle 14 (Substantivanalyse) und Tabelle 15 (Verbanalyse). Korrekt erkannte Anforderungen, die in Crosscutting Concerns aus dem Goldstandard enthalten sind, sind innerhalb der Tabellen **fett** dargestellt.

Anforderungen mit gemeinsamen Bestandteilen
Anforderung 15, Anforderung 23
Anforderung 20, Anforderung 21
Anforderung 47, Anforderung 48
Anforderung 7, Anforderung 32, Anforderung 33
Anforderung 38, **Anforderung 43, Anforderung 44**, Anforderung 57
Anforderung 16, Anforderung 19, **Anforderung 34**

Tabelle 12: Projekt 1: Resultate der Substantivanalyse

Anforderungen mit gemeinsamen Bestandteilen
Anforderung 52, Anforderung 53, Anforderung 54, Anforderung 55, Anforderung 56
Anforderung 20, Anforderung 21, Anforderung 33, Anforderung 38, Anforderung 43, Anforderung 44
Anforderung 7, Anforderung 17, Anforderung 24, Anforderung 34
Anforderung 26, Anforderung 49, Anforderung 50, Anforderung 52, Anforderung 53, Anforderung 55, Anforderung 56
Anforderung 35, **Anforderung 47, Anforderung 48**
Anforderung 17, Anforderung 29
Anforderung 4, Anforderung 10, Anforderung 11, Anforderung 13, Anforderung 14, Anforderung 31, Anforderung 37, Anforderung 42, Anforderung 45
Anforderung 3, Anforderung 7
Anforderung 6, Anforderung 33
Anforderung 1, Anforderung 17
Anforderung 19, **Anforderung 20, Anforderung 21**, Anforderung 22, Anforderung 23
Anforderung 20, Anforderung 21, Anforderung 41
Anforderung 27, Anforderung 39, Anforderung 40, Anforderung 46, Anforderung 59
Anforderung 29, Anforderung 41
Anforderung 20, Anforderung 21
Anforderung 40, Anforderung 46, Anforderung 54
Anforderung 2, Anforderung 41
Anforderung 47, Anforderung 48
Anforderung 37, Anforderung 45, **Anforderung 58, Anforderung 59**, Anforderung 60
Anforderung 2, Anforderung 3, Anforderung 8, Anforderung 20, Anforderung 21, Anforderung 28
Anforderung 5, Anforderung 32, Anforderung 42
Anforderung 30, Anforderung 31
Anforderung 8, **Anforderung 10**, Anforderung 11, Anforderung 12, Anforderung 13, Anforderung 14, Anforderung 15, Anforderung 16, **Anforderung 25, Anforderung 51**
Anforderung 17, Anforderung 24, Anforderung 34
Anforderung 2, Anforderung 6, Anforderung 9, Anforderung 12, Anforderung 15, Anforderung 16, Anforderung 18, Anforderung 20, Anforderung 21, Anforderung 22, Anforderung 23, **Anforderung 25**, Anforderung 26, Anforderung 27, Anforderung 29, Anforderung 30, Anforderung 34, Anforderung 36, Anforderung 39, Anforderung 49, **Anforderung 50, Anforderung 51**
Anforderung 29, Anforderung 37, Anforderung 38, **Anforderung 43, Anforderung 44**, Anforderung 45
Anforderung 57, Anforderung 58, Anforderung 59

Tabelle 13: Projekt 1: Resultate der Verbanalyse

Die kombinierten Metriken der beiden Analysen in Projekt 1 berechnen sich somit wie folgt:

- 8 True Positives (TP)
- 16 False Positives (FP)
- 4 False Negatives (FN)

Precision : $\frac{TP}{TP+FP} = \frac{8}{8+16} = \frac{8}{24} = 33,3\%$

Recall : $\frac{TP}{TP+FN} = \frac{8}{8+4} = \frac{8}{12} = 66,6\%$

Anforderungen mit gemeinsamen Bestandteilen
Anforderung 4, Anforderung 5, Anforderung 8
Anforderung 2, Anforderung 16
Anforderung 6, Anforderung 14
Anforderung 5, Anforderung 8, **Anforderung 11, Anforderung 12**, Anforderung 15

Tabelle 14: Projekt 2: Resultate der Substantivanalyse

Anforderungen mit gemeinsamen Bestandteilen
Anforderung 11, Anforderung 12, Anforderung 18
Anforderung 4, Anforderung 7, **Anforderung 10, Anforderung 11, Anforderung 12, Anforderung 13, Anforderung 14, Anforderung 15**
Anforderung 1, Anforderung 9
Anforderung 2, Anforderung 3
Anforderung 5, Anforderung 18
Anforderung 2, Anforderung 6, Anforderung 9
Anforderung 10, Anforderung 11
Anforderung 6, Anforderung 7, Anforderung 8, Anforderung 15

Tabelle 15: Projekt 2: Resultate der Verbanalyse

Die kombinierten Metriken der beiden Analysen in Projekt 2 berechnen sich somit wie folgt:

- 4 True Positives (TP)
- 5 False Positives (FP)
- 0 False Negatives (FN)

Precision : $\frac{TP}{TP+FP} = \frac{4}{4+5} = \frac{4}{9} = 44,4\%$

Recall : $\frac{TP}{TP+FN} = \frac{4}{4+0} = \frac{4}{4} = 100\%$

6.3 Diskussion

Die Ergebnisse sind sehr stark von der Güte der Eingangsdaten abhängig. Bei User Stories ist hiermit konkret das Einhalten von Formatvorlagen sowie das Vermeiden von Rechtschreibfehlern und Grammatikfehlern gemeint. Diese Problematik tritt beispielsweise in Projekt 1 bei Anforderung 18 zutage, denn hier liegt eine nicht vollständig formulierte Anforderung vor. Bei dieser ist es dem Algorithmus nicht möglich, eine korrekte Korrelation zu anderen Anforderungen herzustellen, da relevante Informationen nicht vorhanden sind. In Projekt 2 sind Anforderung 8 und Anforderung 14 anzusprechen, bei denen gängige Formatvorlagen nicht oder nicht vollständig eingehalten wurden. Zusätzlich erschweren Grammatikfehler die Identifikation von Crosscutting Concerns durch den Algorithmus.
Aus den Ergebnissen aus Abschnitt 6.2 ist zu entnehmen, dass der Ansatz bei Projekt 2, welches 18 Anforderungen enthält, alle Crosscutting Concerns des Goldstandards korrekt identifizieren konnte. Somit erzielt der Algorithmus bei Projekt 2 einen deutlich höheren Recall, als bei Projekt 1, welches 60 Anforderungen beinhaltet.
Der Anstieg der Komplexität bei der Erkennung von Crosscutting Concerns ist stark überproportional zum Anstieg der Anforderungsmenge, da bei größeren Projekten exponentiell mehr Abhängigkeiten zwischen den Anforderungen zu analysieren sind.
Außerdem wird bei Betrachtung der Ergebnisse aus Abschnitt 6.2 ersichtlich, dass die Verbanalyse bei einem kleineren Projekt eine deutlich höhere Trefferquote erzielt als bei einem größeren, wohingegen die Substantivanalyse bei Projekt 1 deutlich effektiver ist. Bei einer entsprechend großen Menge an Anforderungen, nehmen die durch die Verbanalyse aufgedeckten Korrelationen stark zu, da bei der Anforderungserhebung oft gleiche Verben in verschiedenem Kontext benutzt werden (vgl. “view”, “set”). Allerdings ist für die volle Effektivität der Substantivanalyse erst

ein gewisser Stichprobenumfang nötig, um ausreichend Übereinstimmungen finden zu können.
Die manuelle Identifikation der Crosscutting Concerns im Rahmen des Goldstandards erforderte viel Zeit. Durch die Größe von Projekt 1 entstand außerdem ein hoher Aufwand, da hier der paarweise Vergleich der Anforderungen bereits sehr unübersichtlich war. Durch die vorliegenden Ergebnisse des Ansatzes wäre eine deutlich bessere Ausgangslage entstanden, auf deren Basis mögliche Crosscutting Concerns effektiver auszuwählen gewesen wären. Somit hätte der Vorgang insgesamt deutlich beschleunigt werden können.

7 Zusammenfassung und Ausblick

Abschnitt 7.1 fasst die im Rahmen der Arbeit entstandenen Ergebnisse und Erkenntnisse zusammen. Abschnitt 7.2 gibt einen Überblick über mögliche Ansätze zur weiteren Forschungsarbeit.

7.1 Zusammenfassung

Ziel der Arbeit war die Implementierung eines Ansatzes zur Identifikation von gemeinsamen Bestandteilen in User Stories im Kontext eines Jira-Plugins. Dafür musste zunächst eine entsprechende Literaturrecherche durchgeführt werden, um einen Überblick über bestehende Ansätze zur Identifikation von Crosscutting Concerns auf Anforderungsebene zu erhalten. Hierbei wurden auf Basis von vorgegebener Ausgangsliteratur durch Vorwärts- und Rückwärtssuche neue Artikel gesucht, die ebensolche Ansätze beschreiben.
Bereits bei der Literatursynthese wurde ersichtlich, dass es keinen Ansatz gibt, der Crosscutting Concerns auf Anforderungsebene zuverlässig und mit hoher Präzision erkennen kann. Die Qualität der Ergebnisse ist proportional zum Aufwand des Anforderungsingenieurs [6] sowie der Ausführungsdauer des Algorithmus [22]. Es wurde entschieden, einen NLP Ansatz zur Umsetzung zu verwenden, um die Vorteile der Strukturvorlagen von User Stories bestmöglich auszunutzen.
Die aus dem 3CI-Ansatz [2] abgeleitete Verbanalyse hat bei kleinen Projekten bis circa 20 Anforderungen sehr gute Ergebnisse gezeigt. Um auch bei größeren Projekten eine gute Ergebnisleistung zu erzielen, wurde zusätzlich ein alternativer Ansatz entwickelt, der den imperativen Teil von User Stories auf Substantiv-Korrelationen untersucht. Dieser Ansatz zeigt gute Ergebnisse bei dem im Rahmen dieser Arbeit getesteten Projekt mit 60 Anforderungen.
Die eindeutige und zweifelsfreie Identifikation von gemeinsamen Bestandteilen in User Stories im Rahmen des Goldstandards bereitete Schwierigkeiten. Dies ist mit mangelndem Domänenwissen über die beiden Projekte zu begründen.
Die Hauptschwierigkeiten der automatischen Analyse liegen im semantischen Kontext von textuellen Anforderungsdokumenten, der von einem Algorithmus nie in vollem Umfang erkannt werden kann. Die zweideutige Auslegung von Verben in User Stories und deren implizite Bestandteile können oft durch die ausschließliche Textanalyse nicht aufgedeckt werden. Durch die Kombination der beiden angeführten Ansätze wurde versucht, die Nachteile des jeweils anderen zu minimieren und so unabhängig von der Projektgröße gute Ergebnisse zu erzielen. Jedoch kann das entwickelte Werkzeug die Aufgabe des Anforderungsingenieurs nicht vollständig ersetzen, sondern nimmt bei der Identifikation von Crosscutting Concerns eine unterstützende Funktion ein.

7.2 Ausblick

Ein sinnvoller Schritt, um die Auswirkungen des in Abschnitt 7.1 angeführten Mangels an Domänenwissen zu mindern, wäre die Aufnahme weiterer Anforderungs- und Entwurfsdokumente in die Analyse. Dabei kann die Menge an Informationen, die dem Algorithmus zur Verfügung stehen, erhört werden. Dadurch können die nötigen Gemeinsamkeiten, die zwei Anforderungen aufweisen müssen, um als Crosscutting Concern klassifiziert zu werden, erhöht werden, was der Treffergenauigkeit zugute kommt. Voraussetzung dafür ist die vom Anforderungsingenieur vorgenommene korrekte Verlinkung von User Stories mit weiteren Anforderungsdokumenten. Durch diese Verlinkung steigt das Wissen über das System, dass zum Anforderungszeitpunkt vorliegt.
Eine alternative Möglichkeit ist die Entwicklung eines Ansatzes aus dem Bereich des maschinellen Lernens. Hierbei können als Trainingsdaten vollständige Projekte mit korrekt identifizierten und implementierten Crosscutting Concerns dienen. Damit kann das Wissen über typische Crosscutting Concerns auf Anforderungsebene, das sich beim heutigen Stand fast ausschließlich auf nichtfunktionale Anforderungen beschränkt [17, 26], vergrößert werden.
Eine Frage, die noch weiterer empirischer Untersuchungen bedarf, ist die Qualität des entwickelten Ansatzes bei der Anwendung auf entsprechend großen Projekten. In diesem Zusammenhang wäre es möglich, strengere Regeln für die Klassifikation von Crosscutting Concerns zu verwenden, da die Menge an Informationen über das System größer ist. Konkret bedeutet das, die Einstufung als Crosscutting Concern nur vorzunehmen, wenn eine größere Anzahl an Gemeinsamkeiten zwischen den entsprechenden Anforderungen vorliegt. Durch den modularen Aufbau des Plugin Prototypen wäre, bei Existenz eines ausreichend großen Projekts, eine Adaption des Werkzeugs zur Unterstützung dieser Regeln ohne Weiteres möglich.

A Anhang

A.1 Ergebnisse der Vorwärts- und Rückwärtssuche

Kürzel	Titel	Ausschluss-kriterium
Baniassad 1	Theme - An Approach for Aspect-Oriented Analysis and Design	AK1
Baniassad 2	Towards requirements-driven information systems engineering- the Tropos project	AK1
Baniassad 3	Composition patterns- An approach to designing reusable aspects	AK1
Baniassad 4	Formal refinement patterns for goal-driven requirements elaboration.	AK1
Baniassad 5	The viewpoints faq. BCS/IEE Software Engineering Journal	AK1
Baniassad 6	Aspect-oriented requirements engineering for component based software systems	AK1
Baniassad 7	Architectural views of aspects	AK1
Baniassad 8 Rosenhainer 10 Sampaio 2	Modularisation and composition of aspectual requirements	AK2
Baniassad 9	Early stage concern modeling	AK1
Baniassad 10	N degrees of separation-Multi-dimensional separation of concerns	AK1
Baniassad 11	Agent-oriented requirements engineering using congolog and i*	AK1
Rosenhainer 1 Sampaio 1 Baniassad et al. [6]	Finding Aspects in Requirements with Theme/Doc	Duplikat
Rosenhainer 2	Standards, Guidelines, and Examples on System and Software Requirements Engineering	AK4
Rosenhainer 3	The Case for Aspects	AK4
Rosenhainer 4	Mining Aspects	AK1
Rosenhainer 5	Crosscutting Quality Attributes for Requirements Engineering	AK1

Rosenhainer 6	Crosscutting Requirements	AK5
Rosenhainer 7	On the Criteria To Be Used in Decomposing Systems into Modules	AK1
Rosenhainer 8	A Process Centered Requirements Engineering Environment	AK4
Rosenhainer 9	Toward Reference Models for Requirements Traceability	AK1
Rosenhainer 11 Sampaio 3	Early Aspects- a Model for Aspect-Oriented Requirements Engineering	
Rosenhainer 12	Mastering the Requirements Process	AK4
Rosenhainer 13	Requirements Interaction Management	AK2
Rosenhainer 14	Concerns in a Requirements Model— A Small Case Study	AK1
Rosenhainer 15	Modeling of Software Concerns in Cosmos	AK1
Sampaio 4 Rosenhainer [26]	Identifying Crosscutting Concerns in Requirements Specifications	Duplikat
Sampaio 5	REVERE- Support for Requirements Synthesis from Documents	AK1
Sampaio 6	Agile Alliance Website. Available at: http://www.agilealliance.org/home.	AK5
Sampaio 7	AbstFinder, A Prototype Natural Language Text Abstraction Finder for Use in Requirements Elicitation	AK1
Sampaio 8	Object-Oriented Software Engineering: A Use Case Driven Approach	AK4
Sampaio 9	Requirements Engineering – A Good Practice Guide	AK4
Sampaio 10	An Introduction to Formal Specification and Z. Prentice Hall	AK4

Tabelle 16: Rückwärtssuche: Ausschluss nach Ausschlusskriterien

Kürzel	Titel	Ausschluss-kriterium
Baniassad 1.1	Early aspect identification from use cases using NLP and WSD techniques	
Baniassad 1.2	A comparative study of aspect-oriented requirements engineering approaches	AK1
Baniassad 1.3	Aspect Mining. Past, Present, Future	AK1
Baniassad 1.4	3ci: A tool for crosscutting concern identification	
Baniassad 1.5	Non-functional requirements to architectural concerns: ML and NLP at crossroads	AK1
Baniassad 1.6	Revealing crosscutting concerns in textual requirements documents: an exploratory study with industry systems	
Baniassad 1.7	Guidelines for the incremental identification of aspects in requirements specifications	AK2
Baniassad 1.8	在跨版本的史更中辨面向	AK6
Baniassad 1.9	Identifying architectural concerns from non-functional requirements using support vector machine	AK1
Baniassad 2.1	Crosscutting concern identification at requirements level	
Baniassad 2.2	Developing tool for crosscutting concern identification using nlp	
Rosenhainer 1.1	Early aspect identification from use cases using NLP and WSD techniques	Duplikat
Rosenhainer 1.2	3ci: A tool for crosscutting concern identification	Duplikat
Rosenhainer 1.3	Guidelines for the incremental identification of aspects in requirements specifications	Duplikat
Rosenhainer 1.4	Utilisation des services et des aspects pour la réutilisabilité du logiciel d'un automate pour l'analyse de plasma	AK6

Rosenhainer 2.1	Crosscutting concern identification at requirements level	Duplikat
Rosenhainer 2.2	Developing tool for crosscutting concern identification using nlp	Duplikat
Sampaio 1.1	On some criteria for comparing aspect mining techniques	
Sampaio 1.2	Evaluating ea-miner: Are early aspect mining techniques effective?	AK1
Sampaio 1.3	On evaluating aspect mining techniques	AK1
Sampaio 1.4	A comparative study of aspect-oriented requirements engineering approaches	Duplikat
Sampaio 1.5	Evaluation measures for partitioning based aspect mining techniques	AK1
Sampaio 1.6	A partitional clustering algorithm for crosscutting concerns identification	AK3
Sampaio 1.7	Aspect Mining. Past, Present, Future	Duplikat
Sampaio 1.8	Hierarchical clustering for identifying crosscutting concerns in object oriented software systems	AK1
Sampaio 1.9	Analysing source code: looking for useful verb–direct object pairs in all the right places	AK3
Sampaio 1.10	3ci: A tool for crosscutting concern identification	Duplikat
Sampaio 1.11	Identifying crosscutting concerns using partitional clustering	AK3
Sampaio 1.12	Non-functional requirements to architectural concerns: ML and NLP at crossroads	Duplikat
Sampaio 1.13	Revealing crosscutting concerns in textual requirements documents: an exploratory study with industry systems	Duplikat
Sampaio 1.14	Guidelines for the incremental identification of aspects in requirements specifications	Duplikat

Sampaio 1.15	Legacy Systems towards Aspect-Oriented Systems	AK1
Sampaio 1.16	Considerações acera da mineração de aspectos considerations about aspect mining	AK6
Sampaio 1.17	Identifying architectural concerns from non-functional requirements using support vector machine	Duplikat
Sampaio 1.18	Natural language program analysis: Combining natural language processing with program analysis to improve software maintenance tools	AK5

Tabelle 17: Vorwärtssuche: Ausschluss nach Ausschlusskriterien

A.2 Anforderungen der Evaluationsprojekte

A.2.1 Anforderungen des Projekts 1

R1 As an Administrative User I want to change the JSON schemas by validating diffrent types of Input Standarts, in order to adapt to changing requirements.
R2 As a technical user with certain access level, I want to use a function in order to insert the geolocation (lat., long.) of Organisations into their profiles, to set their location.
R3 As an Administrative User with certain access level, I want to set the percentage of matched values so that the treshold for the matching algorithms is set to the provided percentage.
R4 As an administrative User I want to to ensure that only reliable sources enter data to the Database in order to approve data sources (Organizations/Practitioners) that send information to the MasterPatientIndex.
R5 As a Administrative User with certain access level, I want to see the "created byëntry of a profile in order to check which user (administrator, organisation, practitioner) has created this patient profile.
R6 As an Administrative User I want to use the GUI in order to configure the internals of the MPI matching algorithm (weightings) and various other configurations (e.g. schemata validation -¿ required fields, database settings,).
R7 As an Administrative User I want to move patient profiles between Persons, to correct mismatched entries.
R8 As a Administrative User with certain access level, I want set one or more

accessrights in order to create a access level.
R9 As an technical user I want use a create
R10 As a Administrative User with certain access level, I want to enter attributes of a Organisationprofile in order to create a profile of that Organisation.
R11 As a Administrative User with certain access level, I want to enter attributes of a Patientprofile in order to create a profile of that Patient.
R12 As an technical user I want use a function in order to create a Practitioner.
R13 As a Administrative User with certain access level, I want to enter attributes of a Practitionerprofile in order to create a profile of that practitioner.
R14 As a Administrative User with certain access level, I want to enter an Email-Adress, a Username and a Password in order to create a Useraccount for the MPI.
R15 As an Administrative User I want to use the GUI to create a new user (of any type) containing an email-address, first/last/username and password
R16 As an Administrative User I want to use the GUI to create a new access level group.
R17 As a Administrative User with certain access level, I want to change given rights in order to remove a previous created access level."
R18 As an technical user I want use a delete
R19 As a Administrative User with certain access level, I want to be able to delete an Organisationprofile using GUI.
R20 As a technical User with certain access level, I want to use a delete function in order to delete the corresponding profile. The profile should be deleted, but a "deprecated"flag should be set.
R21 As a Administrative User with certain access level, I want to use a delete button in order to delete the corresponding profile. The profile should be deleted, but a "deprecated"flag should be set.
R22 As an technical user I want use a function in order to delete a Practitioner.
R23 As an Administrative User I want to use the GUI to delete an existing user (of any type)
R24 As a Administrative User with certain access level, I want klick a delete button of a Useraccount in order to remove this user.
R25 As an technical user I want use a edit function in order to create a Organisation.
R26 As an technical user I want use a function in order to edit the values of a Practitioner.
R27 As a technical User, i want to use a filter function in order get a list of all unmatched Patients in the Database.
R28 As an Administrator I want to set diffrent rights for diffrent users in order to grant Access of diffrent Levels to the Systems Endpoints.
R29 As a technical user with certain access level, I want to use a function with

an Organisation and a radius as input, witch returns all near located Organisations and their locations to the given Organisation and radius in order to find and display all near located Organisations.
R30 As an technical User I want to use function on all routes in order to log all actions made via these endpoints.
R31 As a Human User, i want to enter Username and Password in order to log into Useraccount corresponding to the entered data.
R32 As a Administrative User I want to see a list of matches in order to manually process matching of unsure matched profiles.
R33 As a administrative User I want to match new profiles with existing profiles, to connect profiles for the same Person. Profiles which have a matching factor in a certain range (below matching treshold but above a 2nd treshold) should be suggested for this Person.
R34 As an Administrative User I want to use the GUI to add/remove access levels to/from an existing access level group.
R35 As a technical user with certain access level, I want to reassign a patient to an other person profile. The patient to reassign must be identifiable by either his masterID or his external ID + inserting organization / practitioner.
R36 As an Administrative User with certain access level, I want use a relate function (not sure how it will be implemented) in order to link the relationships between different persons (not profiles).
R37 As a Administrative User with certain access level, I want to enter keywords of a Organisationprofile in order to find and view a profile a Organisation.
R38 As a technical user with certain access level, I want to POST keywords of an Organisation in order to search and find Organisations. The output should be JSON.
R39 As a technical User I want to use a search function in order to search a patient by his ID, to get all stored patient data
R40 As a Technical User with certain access level (organisation / practitioner), I want to send the internal ID of my organisation in order to get a profiles Master ID.
R41 As a Administrative User I want to insert search parameters regarding patients (PatientID, Firstname, Lastname, Origin [Organization/Practitioner], Birthdate, deprecated [bool]). in order to display an (_web_) interface for viewing search results.
R42 As a Human User I want to enter an ID into a searchfield in order to see all information to a specific Patient.
R43 As a technical user with certain access level, I want to POST keywords of a Patient in order to search and find Patients. The output should be JSON.
R44 As a technical user with certain access level, I want to POST keywords of a

Practitioner in order to search and find Practitioners. The output should be JSON.
R45 As a Administrative User with certain access level, I want to enter keywords of a Practitionerprofile in order to find and view a profile a Practitioner.
R46 As a Technical User with certain access level, I want to send a persons ID by POST to the Service in order to get the relationships of that person to other persons. (in JSON)
R47 As a technical user with certain access level, I want to store a ïnserted by"value, that stores an ID, in the Patientprofile in order to keep track of the Organisation that inserted the Patient. The inserter ID must be saved.
R48 As a technical user with certain access level, I want to store a ïnserted by"value, that stores an ID, in the Patientprofile in order to keep track of the Practitioner that inserted the Patient. The inserter ID must be saved.
R49 As a Administrative User with certain access level, I want to use a function in order to edit a previous created access level.
R50 As a technical user with certain access level, I want to use a function in order to edit the geolocation (lat., long.) values of the corresponding Organisation.
R51 As an technical user I want use a update function in order to create a Organisation.
R52 As a Administrative User with certain access level, I want to edit attributes of a Organisationprofile in order to update a profile of that Organisation.
R53 As a Administrative User with certain access level, I want to edit attributes of a patientprofile in order to update a profile of that person.
R54 As a technical user with certain access level, I want to send the internal profile ID (in the organisation) OR the master ID in order to update a person's profile.
R55 As a Administrative User with certain access level, I want to edit attributes of a Practitionerprofile in order to update a profile of that Practitioner.
R56 As a Administrative User with certain access level, I want to edit attributes of a Useraccount in order to update this user.
R57 As a Human User, i want to click a ßearch for near located OrganisationsButton on a profile of an Organisation in order to view near located Organisations.
R58 As a Human User with certain rights, I want to click a "Tree-Button" in order to view the retrieved patientdata as Tree in the User Interface
R59 As an administrative User I want to click a "view patients"Button in order to view a list of all patientdata in the Database to get an overview of the stored data.
R60 As a Human User with certain access level, I want to select a Person in order to view the relationships of that person to other persons.

A.2.2 Anforderungen des Projekts 2

R1 As a user, I want see rooms on the map in different colors in order to differentiate between room types more quickly.
R2 As a user, I want to restrict the search to certain categories of map objects in order to get filtered search results.
R3 As a user, I want to apply categories to map objects in order to highlight filtered objects appropriately.
R4 As a user, I want to view the result of the search highlighted on the map in order to locate a searched object.
R5 As a user, I want to have a few controls in order to easily change the map view. There should be 2D mode and a button to orient the map to north.
R6 As a user, I want to click on people in the info box of a room in order to get information about this person.
R7 As a user, I want to click on a station in order to view the departure schedule. The departure schedule is displayed in the InfoBox.
R8 As a user, I want to click on an object (building, room, stop) in order view the information of an object.
R9 As a user, I want to see the map in the main view of the app in order to get an overview of the campus.
R10 As a user, I want to enter two points in order to view the route between them. Points can be, e.g., rooms, buildings, people, stations, actual position.
R11 As a user, I want to search for persons (professors, assistants, secretaries) on the campus so that I can view their position on the map.
R12 As a user, I want to search for rooms and buildings on the campus in order to view the location on the map and information about it.
R13 As a user, I want to set a start and end point with a long click on the map in order to view the route between these points.
R14 As a user, I want to view the person's tweets when I view information of a person.
R15 As a user, I want to click on a station to view all lines numbers on the map.
R16 As a user, I want an overview of the division of the university into institutes, faculties and research groups in order to discover potentially interesting search results exploratively.
R17 As a user, I want to find free rooms in order to stay with my friends or to relax. I want to limit the amount of free rooms using a search mask in order to met my personal preferences, including, for example, the number of seats.
R18 As a user, I want to have the latest LSF-data in order to search for correct lectures.

Tabellenverzeichnis

Quellcodeverzeichnis

Abbildungsverzeichnis

Literaturverzeichnis

[1] agilealliance. *What is Role-Feature-Reason*. 2018. URL: `https://www.agilealliance.org/glossary/role-feature/` (besucht am 12. 10. 2018).

[2] Busyairah Syd Ali und Zarinah Mohd Kasirun. "3ci: A tool for crosscutting concern identification". In: *CIMCA 2008, IAWTIC 2008, and ISE 2008*. IEEE. 2008, S. 351–355.

[3] Busyairah Syd Ali und Zarinah Mohd Kasirun. "Crosscutting concern identification at requirements level". In: *Malaysian Journal of Computer Science* 21.2 (2008), S. 78–87.

[4] Busyairah Syd Ali und Zarinah Mohd Kasirun. "Developing tool for crosscutting concern identification using nlp". In: *Information Technology, 2008. ITSim 2008. International Symposium on*. Bd. 3. IEEE. 2008, S. 1–8.

[5] Elisa Baniassad und Siobhan Clarke. "Theme: An approach for aspect-oriented analysis and design". In: *Proceedings of the 26th International Conference on Software Engineering*. IEEE Computer Society. 2004, S. 158–167.

[6] Elisa Baniassad und Siobhán Clarke. "Finding aspects in requirements with theme/doc". In: *Early Aspects: Aspect-Oriented Requirements Engineering and Architecture Design* (2004), S. 15–22.

[7] Silvia Breu und Jens Krinke. "Aspect mining using event traces". In: *Automated Software Engineering, 2004. Proceedings. 19th International Conference on*. IEEE. 2004, S. 310–315.

[8] Silvia Breu und Thomas Zimmermann. "Mining aspects from version history". In: *Automated Software Engineering, 2006. ASE'06. 21st IEEE/ACM International Conference on*. IEEE. 2006, S. 221–230.

[9] Isabel Brito. "Aspect-Oriented Requirements Engineering". In: *Proceeding of the7th International Conference on Unified Modelling Language (UML)*. 2004.

[10] Magiel Bruntink u. a. "On the use of clone detection for identifying crosscutting concern code". In: *IEEE Transactions on Software Engineering* 31.10 (2005), S. 804–818.

[11] Gobinda G Chowdhury. "Natural language processing". In: *Annual review of information science and technology* 37.1 (2003), S. 51–89.

[12] Mike Cohn. *User stories applied: For agile software development*. Addison-Wesley Professional, 2004.

[13] Grigoreta Sofia Cojocar und Gabriela Şerban. “On some criteria for comparing aspect mining techniques”. In: *Proceedings of the 3rd workshop on Linking aspect technology and evolution*. ACM. 2007, S. 7.

[14] IEEE Computer Society. Software Engineering Standards Committee und IEEE-SA Standards Board. “IEEE recommended practice for software requirements specifications”. In: *Software Requirements Specification*. Institute of Electrical und Electronics Engineers. 1998.

[15] Orlena CZ Gotel und CW Finkelstein. “An analysis of the requirements traceability problem”. In: *Requirements Engineering, 1994., Proceedings of the First International Conference on*. IEEE. 1994, S. 94–101.

[16] Lehrstuhl Software-Engineering Heidelberg. *Richtlinien für Literaturrecherche*. 2017. URL: `http://confluence-se.ifi.uni-heidelberg.de/x/hgWc` (besucht am 27. 10. 2018).

[17] José Herrera u. a. “Revealing crosscutting concerns in textual requirements documents: an exploratory study with industry systems”. In: *2012 26th Brazilian Symposium on Software Engineering*. IEEE. 2012, S. 111–120.

[18] ISO ISO. “26262–1: 2011 road vehicles functional safety. ISO”. In: *International Organization for Standardization, Geneva, Switzerland* (2011).

[19] Michael Käßmeyer, Michael Schulze und Markus Schurius. “A process to support a systematic change impact analysis of variability and safety in automotive functions”. In: *Proceedings of the 19th International Conference on Software Product Line*. ACM. 2015, S. 235–244.

[20] Marius Marin, Arie Van Deursen und Leon Moonen. *Identifying aspects using fan-in analysis*. IEEE, 2004.

[21] Ana Moreira u. a. *Aspect-oriented requirements engineering*. Springer, 2013.

[22] Alejandro Rago u. a. “Early aspect identification from use cases using NLP and WSD techniques”. In: *Proceedings of the 15th workshop on Early aspects*. ACM. 2009, S. 19–24.

[23] Awais Rashid u. a. “Early aspects: A model for aspect-oriented requirements engineering”. In: *Requirements Engineering, 2002. Proceedings. IEEE Joint International Conference on*. IEEE. 2002, S. 199–202.

[24] Awais Rashid, Paul Rayson u. a. “Early-aim: An approach for identifying aspects in requirements”. In: *null*. IEEE. 2005, S. 487–488.

[25] Paul Rayson u. a. “The UCREL semantic analysis system.” In: (2004).

[26] Lars Rosenhainer. “Identifying crosscutting concerns in requirements specifications”. In: *Proceedings of OOPSLA Early Aspects*. 2004.

[27] Américo Sampaio u. a. “EA-Miner: a tool for automating aspect-oriented requirements identification”. In: *Proceedings of the 20th IEEE/ACM international Conference on Automated software engineering*. ACM. 2005, S. 352–355.

[28] Américo Sampaio u. a. “Mining aspects in requirements”. In: *Early Aspects 2005: Aspect-Oriented Requirements Engineering and Architecture Design Workshop*. 2005.

[29] Marcus Seiler und Barbara Paech. “Using tags to support feature management across issue tracking systems and version control systems”. In: *International Working Conference on Requirements Engineering: Foundation for Software Quality*. Springer, Cham. 2017, S. 174–180.

[30] Stanley M Sutton Jr und Isabelle Rouvellou. “Modeling of software concerns in cosmos”. In: *Proceedings of the 1st international conference on Aspect-oriented software development*. ACM. 2002, S. 127–133.

[31] Paolo Tonella und Mariano Ceccato. “Aspect mining through the formal concept analysis of execution traces”. In: *null*. IEEE. 2004, S. 112–121.

[32] Karl Wiegers und Joy Beatty. *Software requirements*. Pearson Education, 2013.