HOCHSCHULE
MITTWEIDA
UNIVERSITY OF
APPLIED SCIENCES

DIPLOMARBEIT

Herr
Markus Misselwitz

Optimierung der Rückverfolgbarkeit von Anforderungen im Software-Entwicklungsprozess

Entwicklung eines Lösungsansatzes zum Verknüpfen von Anforderungen mit Komponenten in der Benutzerschnittstelle

2012

Fakultät **Elektro- und Informationstechnik**

DIPLOMARBEIT

Optimierung der Rückverfolgbarkeit von Anforderungen im Software-Entwicklungsprozess

Entwicklung eines Lösungsansatzes zum Verknüpfen von Anforderungen mit Komponenten in der Benutzerschnittstelle

Autor:
Markus Misselwitz

Studiengang:
Multimediatechnik

Seminargruppe:
MK08

Erstprüfer (Hochschule Mittweida):
Prof. Dr.-Ing. Wilfried Schubert

Zweitprüfer (intecsoft GmbH & Co. KG):
Dipl.-Inf. Claus Jungmann

Mittweida, November 2012

Bibliografische Angaben

Misselwitz, Markus: Optimierung der Rückverfolgbarkeit von Anforderungen im Software-Entwicklungsprozess, 77 Seiten, 33 Abbildungen, Hochschule Mittweida (FH), Fakultät Elektro- und Informationstechnik

Diplomarbeit, 2012

Abstract

Traceability is a prerequisite for important tasks in project management. If requirements are traceable, than e.g. the progress of the project or the completeness of implementation can efficiently be monitored or profound cost estimations can be conducted.

This thesis deals with the optimization of traceability in the software development process at intecsoft GmbH & Co. KG. A software tool was developed to display requirements at their place of implementation in the user interface of the application during runtime.

The major interest was a practicable solution for the daily work in software projects. No extra personal costs should be necessary to establish the traceability solution. The obvious capture method would have been manual linking of components of the user interface with requirements. Instead, there were used traces between source code and requirements, which already existed in the code versioning system of the software project.

Referat

Die Rückverfolgbarkeit von Anforderungen ist Voraussetzung für wichtige Aufgaben im Projektmanagement. Sind Anforderungen im Entwicklungsprozess nachvollziehbar, können z. B. der Projektfortschritt effektiv überwacht, die Vollständigkeit der Umsetzung festgestellt oder fundierte Aufwandsschätzungen durchgeführt werden.

Diese Arbeit beschäftigt sich mit der Optimierung der Rückverfolgbarkeit im Software-Entwicklungsprozess des Unternehmens intecsoft GmbH & Co. KG. Es wurde eine Software entwickelt, die Anforderungen an der Stelle ihrer Umsetzung in der Benutzeroberfläche des Softwareprodukts darstellt.

Das Hauptaugenmerk lag dabei auf einer praktikablen Lösung für den Projektalltag. Es sollte kein weiterer personeller Aufwand zur Etablierung der Rückverfolgbarkeit entstehen. Anstelle einer Erfassungsmethode, die auf manuelles Markieren von Komponenten der Benutzerschnittstelle basiert, wurde auf Beziehungen zwischen Quellcode und Anforderungen zurückgegriffen, die bereits im Versionsverwaltungssystem des Softwareprojekts erfasst waren.

I. Inhaltsverzeichnis

Inhaltsverzeichnis I
Abbildungsverzeichnis II
Tabellenverzeichnis III
Abkürzungsverzeichnis IV
1 Einleitung 1
1.1 Ziele und Abgrenzung 2
1.2 Motivation 3
1.2.1 Hochschule Mittweida 3
1.2.2 Praxispartner 3
1.2.3 Autor 3
1.3 Kapitelübersicht 4
2 Rückverfolgbarkeit im Bereich des Anforderungsmanagement 5
2.1 Einordnung 5
2.2 Aktivitäten 5
2.2.1 Herstellung von Beziehungen 6
2.2.2 Wartung von Beziehungen 7
2.2.3 Nutzung von Beziehungen 8
2.3 Aktuelle Herausforderungen in Bezug auf diese Diplomarbeit 9
2.3.1 Heterogene Artefakte 9
2.3.2 Kosten, Nutzen und Akzeptanz 9
2.3.3 Granularität von Beziehungen 10
2.3.4 The Grand Challenge of Traceability (v1.0) 11
3 Ist-Analyse 13
3.1 Kenndaten des Unternehmens 13
3.2 Prozesse und Rollen 13
3.3 Softwareunterstützung der Prozesse 15
3.3.1 Redmine 15
3.3.2 Git 17

3.3.3 Etablierte Rückverfolgbarkeit ... 17
3.4 Defizite in der Rückverfolgbarkeit von Anforderungen im Unternehmen ... 19
4 Soll-Zustand ... 21
4.1 Anwendungsfälle ... 22
4.2 Anforderungen ... 24
4.2.1 Funktionale Anforderungen ... 24
4.2.2 Nicht-Funktionale Anforderungen ... 25
4.3 Technische Randbedingungen ... 26
5 Lösungsansätze ... 29
5.1 Facharbeit E. Gorning ... 29
5.2 BugHerd ... 31
5.3 Nutzung vorhandener Beziehungen zwischen Quellcode-Versionsständen und Redmine Tickets ... 34
5.4 Evaluation ... 36
5.5 Favorisierung eines Lösungsansatzes ... 39
6 Konzeption ... 41
6.1 Systemkomponenten ... 41
6.1.1 WicketTracer Module ... 41
6.1.2 WicketTracer Client ... 42
6.1.3 WicketTracer Webservice ... 43
6.2 Benutzerschnittstelle ... 43
6.3 Programmierschnittstellen ... 44
6.3.1 WicketTracer Client - WicketTracer Webservice ... 45
6.3.2 WicketTracer Webservice - Git ... 45
6.3.3 WicketTracer Webservice - Redmine ... 45
7 Realisierung ... 47
7.1 WicketTracer Module ... 47
7.2 WicketTracer Client ... 48
7.2.1 Benutzerschnittstelle ... 48
7.2.2 Codierung ... 50
7.3 WicketTracer Webservice ... 51

8 Fazit 55
8.1 Zusammenfassung und Bewertung 55
8.2 Ausblick 58
8.2.1 Manuelle Markierung auf Benutzeroberfläche 58
8.2.2 Vorwärts gerichtete vertikale Rückverfolgung 59
8.2.3 Verbesserungen des WicketTracer Clients 60
A Interviewfragen Ist-Zustand 61
B Frühere Versionen des Systementwurfs 65
C Frühere Entwürfe der Benutzerschnittstelle 67
Literaturverzeichnis 69
Glossar 73

II. Abbildungsverzeichnis

1.1 Liste von Anforderungen in Redmine 2
2.1 Rückverfolgbarkeit von Anforderungen vor und nach Spezifikation 6
2.2 Horizontale und Vertikale Rückverfolgbarkeit 8
3.1 Überblick über Prozesse und Rollen der intecsoft 14
3.2 Bildschirmfoto eines in Redmine angelegten Tickets 16
3.3 Ticketstatus dokumentiert die Station im Prozess 17
3.4 Verlinkungen der bereits etablierten Rückverfolgbarkeit 18
4.1 Anwendungsfälle anhand der Prozesse der Anforderungsanalyse und Konzeption 22
4.2 Anwendungsfälle anhand der Prozesse der Implementierung, Test und Auslieferung . 23
4.3 Ausgangslage der Nachbarsysteme 26
5.1 Erstellen einer Beziehung zwischen Benutzerschnittstelle und Redmine-Ticket in der Facharbeit von E. Gorning 30
5.2 Anzeige von Beziehungen zwischen Benutzerschnittstelle und Redmine-Ticket in der Facharbeit von E. Gorning 30
5.3 Übersicht des Datenflusses zwischen den Systemen des Lösungsansatz der Facharbeit von E. Gorning 31
5.4 Melden eines Fehlers in einer Webseite mit Bughherd 32
5.5 Anzeige von Markierungen auf einer Webseite mit BugHerd 32
5.6 Übersicht des Datenflusses zwischen den Systemen der Lösung von BugHerd 33
5.7 Vergleich der Verlinkungen der Lösungsansätze 34
5.8 Log Befehl mit Pfad als Parameter an das Versionierungssystem Git 35
5.9 Vergleich der Attribute von Wicket-Path und Wicket-Source anhand der schematischen Darstellung einer mit Apache Wicket generierten HTML-Seite 37
6.1 Komponenten und Datenfluss der Ziellösung 42
6.2 Finaler Entwurf der Benutzerschnittstelle des WicketTracer Client 44
7.1 WicketTracer-Klasse wird in der Wicket-Applikation aufgerufen 47
7.2 Quellcode der Klasse „WicketTracer“ 48
7.3 Bildschirmfoto des WicketTracer Clients 49

7.4 UML-Klassendiagramm des WicketTracer Client 50
7.5 Quellcode der Haupt-Klasse des Webservice (vereinfacht) 52
7.6 Vorgänge im Konstruktor der Klasse SourceIssueResponse 53
7.7 Bildschirmfoto der Log-Ausgabe des WicketTracer Webservice 54
8.1 Entwurf der Benutzerschnittstelle der Weiterentwicklung zur manuellen Markierung 59
8.2 Anforderungen erscheinen direkt bei einer Komponente der Benutzerschnittstelle ... 60
B.1 Version 1 „Starke Integration in Wicket-Applikation, geringe Entkoppelung, keine clientseitige Logik“ 65
B.2 Version 2 „Geringe Integration in Wicket-Applikation, hohe Entkoppelung, hauptsächliche Logik im Client“ 65
C.1 Frühere Entwürfe der Benutzerschnittstelle 67

III. Tabellenverzeichnis

4.1 Anwendungsfälle 22

4.2 Funktionale Anforderungen 24

4.3 Nichtfunktionale Anforderungen 25

5.1 Erfüllung der funktionalen Anforderungen 38

8.1 Bewertung der Ergebnisse anhand „The Grand Challenge of Traceability (v1.0)“ 57

IV. Abkürzungsverzeichnis

CSS	**C**ascading **S**tyle **S**heets
HTML	**H**yper**t**ext **M**arkup **L**anguage
JS	**J**ava**S**cript
JSON	**J**ava**S**cript **O**bject **N**otation
PHP	**P**HP: **H**ypertext **P**reprocessor
PL	**P**rojekt**l**eiter
REST	**Re**presentational **S**tate **T**ransfer
URL	**U**niform **R**esource **L**ocator
XML	E**x**tensible **M**arkup **L**anguage
XPath	**X**ML **Path** Language

1 Einleitung

Rückverfolgbarkeit – als eine Unterdisziplin des Anforderungsmanagements in der Softwaretechnik – ist die Grundlage für wichtige Aufgaben im Software-Entwicklungsprozess. Sie wird ermöglicht, sobald Anforderungen, deren Werdegang und zugehörige Artefakte miteinander vernetzt sind. Eine Umfrage von Bouillon et al. unter 71 Unternehmen hat folgende Nutzungsszenarien für die Rückverfolgbarkeit von Anforderungen ergeben [BP09]:

1. Zustandsverfolgung von Anforderungen bzw. Aufträgen,
2. Nachweis, dass alle beschriebenen Anforderungen umgesetzt sind,
3. Entwickeln von Testfällen (direkt aus den Anforderungen),
4. Nachvollziehen, wie einzelne Anforderungen entstanden sind (z. B. Regelwerke, Stakeholder),
5. Planung von Releases,
6. Verfeinern von Anforderungen,
7. Fehlersuche nach Tests,
8. Aufwandsschätzung für Änderungswünsche.

Der dafür notwendige hohe Grad an Verlinkungen zwischen Anforderungen kann in der Praxis nur mithilfe spezieller Software gewährleistet werden.

Jedoch sind solche Systeme zur Verwaltung von Anforderungen fast ausschließlich textbasiert (Abbildung 1.1). Anforderungen werden verbal beschrieben und bestenfalls mit einer Skizze versehen. Der Titel einer Anforderung, der für das erneute Auffinden über Jahre hinweg entscheidend ist, muss klug gewählt sein. Bei der intecsoft GmbH & Co. KG (im Folgenden intecsoft) sind im Anforderungsverzeichnis mehrere hundert Einträge pro Projektjahr die Regel. So entstehen große Anforderungslisten, die für die meisten Projektbeteiligten mit fortschreitender Projektdauer immer unüberschaubarer werden.

An dieser Stelle setzt diese Diplomarbeit an. Es soll eine alternative Sichtweise auf die in der Projektmanagementsoftware Redmine hinterlegten Anforderungen geschaffen werden, indem Anforderungen in der Benutzeroberfläche des Softwareprodukts angezeigt werden. Weiterhin muss eine geeignete Lösung entwickelt werden, um die dafür nötigen Beziehungen zwischen Anforderungen und Benutzerschnittstelle zu erfassen, zu warten und darzustellen.

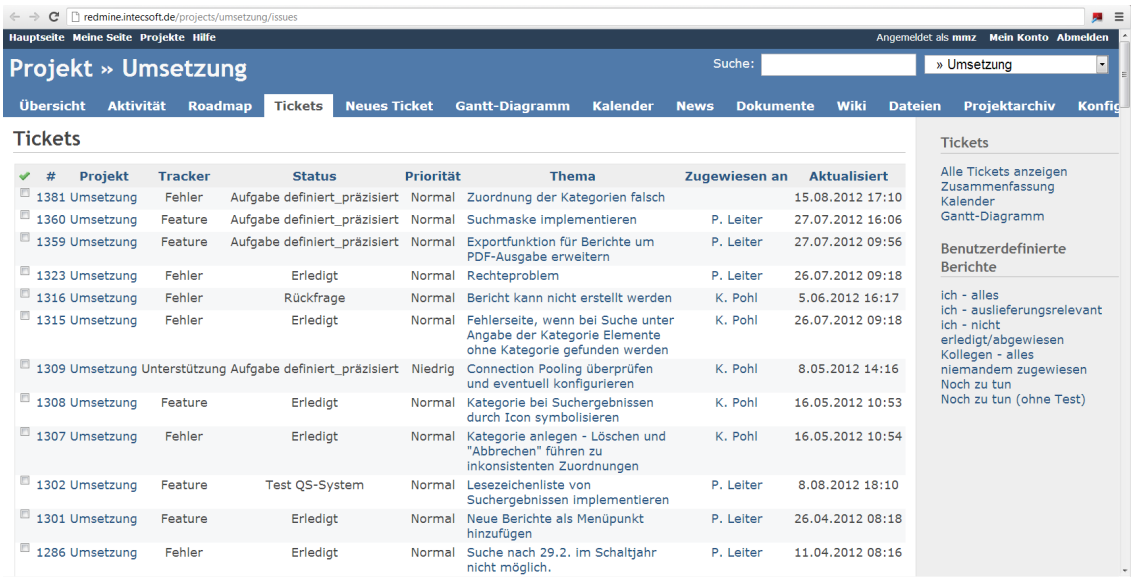


Abbildung 1.1: Liste von Anforderungen in Redmine

1.1 Ziele und Abgrenzung

Die Arbeit soll zunächst theoretische Grundlagen in der aktuellen Literatur zur Rückverfolgbarkeit von Anforderungen erheben, die in Bezug mit dem Diplomthema stehen. Defizite im Bereich Rückverfolgbarkeit bei der intecsoft werden im Rahmen der Ist-Analyse untersucht und mit in der Literatur dokumentierten Forschungsergebnissen verglichen.

Es soll ein Lösungsansatz zum Verknüpfen von Anforderungen mit Komponenten in der Benutzerschnittstelle entwickelt werden. Insbesondere soll auf die Herausforderungen einer bedarfsgerechten Erfassungsmethode und einer geeigneten Visualisierung eingegangen werden.

Die Lösung soll auf den bei der intecsoft verwendeten Technologien und Entwicklungswerkzeugen im Rahmen der Webentwicklung basieren:

- Apache Wicket als Framework für die Benutzerschnittstelle (Java) [Wic12]
- Versionsverwaltungssystem mit Git [Git12]
- Redmine als Projektmanagementsoftware [Red12a]

Durch diese Entwicklung soll die Rückverfolgbarkeit von Anforderungen im Unternehmen optimiert werden. Verbesserte Auffindbarkeit von Anforderungen und eine bessere Sichtbarkeit von Anforderungsbeziehungen sollen die Durchführung nachfolgender Aufgaben im Projektmanagement schneller und effektiver ermöglichen, wie: Ermittlung des aktuellen Projektstatus, Vollständigkeitsanalysen, Risikoanalysen bei Changes, Suche nach Spezifikation einzelner Komponenten.

Diese Arbeit kann als Grundlage für nachfolgende Forschungen dienen. Im jungen Feld der Rückverfolgbarkeit existieren aktuell noch keine dokumentierten Forschungsergebnisse zur Verlinkung von Anforderungen an Elemente der Benutzerschnittstelle zur Laufzeit einer Applikation [vgl. EGHB12].

1.2 Motivation

1.2.1 Hochschule Mittweida

In der praktischen Arbeit während des Multimediatechnikstudiums sind Projekte, die über ein Semester hinausgehen eher die Ausnahme. Da der Fokus auf Neuentwicklung von Software liegt, werden Aspekte der Wartung meist vernachlässigt. Dennoch haben Anforderungsmanagement und Software Engineering durch eigene Lehrveranstaltungen eine hohe Bedeutung im Diplomstudiengang Multimediatechnik.

Diese Diplomarbeit ist für die Hochschule interessant, da Methoden und Werkzeuge entwickelt werden, die langfristige Projekte unterstützen. Gegenüber auf kurze Dauer angelegten Vorhaben, gewinnen hier Lösungen zur Rückverfolgbarkeit von Anforderungen eine größere Bedeutung. Ein modernes Verfahren, welches durch den Praxispartner an die reale Projektarbeit angepasst ist, bildet daher eine Bereicherung für Hochschule und Studenten.

1.2.2 Praxispartner

Die intecsoft ist als Beratungsunternehmen in den Bereichen Prozess- und Softwarearchitekturberatung sowie als Entwickler kundenspezifischer Software tätig. Zu den Kunden in Deutschland und der Schweiz zählen Unternehmen aus der Energie-, Finanzwirtschafts- und Medizinbranche [nach JBG12].

Ein erfolgreiches Projektmanagement gemeinsam mit dem Kunden, sowie effektive Arbeitsprozesse sind auch bei der intecsoft maßgeblich dafür entscheidend, dass Projekte im Zeit- und Budgetplan bleiben. Die Optimierung der Rückverfolgbarkeit von Anforderungen im Software-Entwicklungsprozess ist daher ein wichtiges Anliegen des Unternehmens.

1.2.3 Autor

Der Autor hat sich bereits in der Schulzeit mit programmiertechnischen Implementierungen auseinander gesetzt – beispielsweise beim Erstellen von Webseiten oder dem Entwickeln eines Algorithmus zur Bilderkennung in C++. Dieser Werdegang setzte sich

im Studium an der Hochschule Mittweida (FH) im Studiengang Multimediatechnik fort. In praxisnahen Projekten kamen hier hauptsächlich Hochsprachen (PHP, JS, Java, C#) zum Einsatz. Das bisher größte Projekt war die Entwicklung einer produktiven Anwendung „App“ für Smartphones des Android-Betriebssystems während des Praxissemesters.

Mit der Größe der Projekte wuchs auch der Bedarf an geeigneter Methodik zur Bewältigung umfangreicher Anforderungen und vieler Projektetappen. Der Autor legte daher den Schwerpunkt der fachlichen Weiterentwicklung neben der Implementierung in Themengebiete des Projekts- und insbesondere des Anforderungsmanagements. Die Anforderungsanalyse bildet nach Meinung des Autors den ersten Schritt zum Projekterfolg.

Die Diplomarbeit, welche sich in einem Untergebiet des Anforderungsmanagements wiederfindet, ist daher eine logische Fortführung des eingeschlagenen Weges.

1.3 Kapitelübersicht

Nachdem im ersten Kapitel Motivation, Aufgabenstellung und Ziele der Diplomarbeit erläutert wurden, bietet das zweite Kapitel einen Überblick über den aktuellen Stand der Wissenschaft im Bereich der Rückverfolgbarkeit als Disziplin des Anforderungsmanagements. Hierbei werden die nötigen wissenschaftlichen Grundbegriffe definiert, Modelle und Konzepte beschrieben, sowie über aktuelle Herausforderungen im Fachbereich informiert.

Das dritte Kapitel beschreibt die aktuelle Situation im Unternehmen. Mithilfe von Mitarbeiterinterviews wurden existierende Arbeitsprozesse und Rollen, sowie Werkzeuge zur Prozessunterstützung erhoben. Die bereits etablierte Rückverfolgbarkeit wurde ermittelt und darin bestehende Defizite aufgedeckt.

Während im vierten Kapitel der Soll-Zustand mithilfe von Anwendungsfällen, Anforderungen und technischen Randbedingungen dokumentiert wird, zeigt Kapitel fünf Lösungsmöglichkeiten auf, die anschließend bewertet werden.

Im sechsten Kapitel wird die Konzeption der favorisierten Lösungsvariante beschrieben. Das siebte Kapitel beinhaltet Details der Implementierungsarbeit.

Im abschließenden achten Kapitel wird Bilanz gezogen und es werden als Ausblick mögliche zukünftige Ausbauschritte aufgezeigt.

2 Rückverfolgbarkeit im Bereich des Anforderungsmanagement

Rückverfolgbarkeit wird in der heutigen Zeit in vielen Industriebereichen angewandt. Beispielsweise kann in der Lebensmittelindustrie der Weg von Lebensmitteln vom Produktionsort, über die Verarbeitung bis hin in einzelne Verkaufsfilialen nachvollzogen werden. Rückverfolgbarkeit wird nach GCHH12, S. 9 ermöglicht, wenn Beziehungen zwischen Artefakten (z. B. der Objekte „Lebensmittel“) hergestellt und genutzt werden können.

In der Softwaretechnik findet Rückverfolgbarkeit u. a. im Anforderungsmanagement seine Anwendung. Hier handelt es sich bei den *Artefakten* um Anforderungen und alle mit ihr in Beziehung stehenden Attribute, Dokumente oder Ereignisse. Gotel und Finkelstein empfahlen 1994 die Rückverfolgbarkeit von Anforderungen als *Fähigkeit* zu definieren, den Werdegang einer Anforderung bidirektional zu beschreiben und nachzuvollziehen – beispielsweise von den Ursprüngen, die in die Anforderungserhebung einfließen, über die Konzeption bis hin zur Implementierung und den Tests [nach GF94, S. 4].

2.1 Einordnung

Da unter den betrachteten Artefakten die Anforderungen im Mittelpunkt stehen, kann das Fachgebiet grundsätzlich in zwei Gebiete unterteilt werden – je nachdem ob Beziehungen zwischen Anforderungen und anderen Artefakten vor oder nach der Anforderungsspezifikation nachverfolgt werden. Abbildung 2.1 stellt diesen Sachverhalt dar. Während die frühen Überlegungen, die überhaupt erst zu einer dokumentierten Anforderung führten zur ersten Gruppe zählen, gehören Beziehungen zwischen Anforderungen und Konzeption, Quellcode oder anderen nachgelagerten Artefakten zur zweiten Gruppe. Diese Diplomarbeit mit Anforderungen und Quellcodeversionen als Artefakte ist dem zweiten Gebiet zuzuordnen.

2.2 Aktivitäten

Die drei Hauptaktivitäten sind das Herstellen, Warten und Nutzen von Beziehungen.[1] Während die Nutzung von Beziehungsinformationen das gewünschte Ziel der Rückverfolgbarkeit von Anforderungen darstellt, umfassen die anderen beiden Aktivitäten lediglich die Vorbereitung bzw. Voraussetzung dazu.

[1] In der hauptsächlich englischsprachig vorhanden Literatur lauten die Fachbegriffe “Creation, maintenance and use of traces”

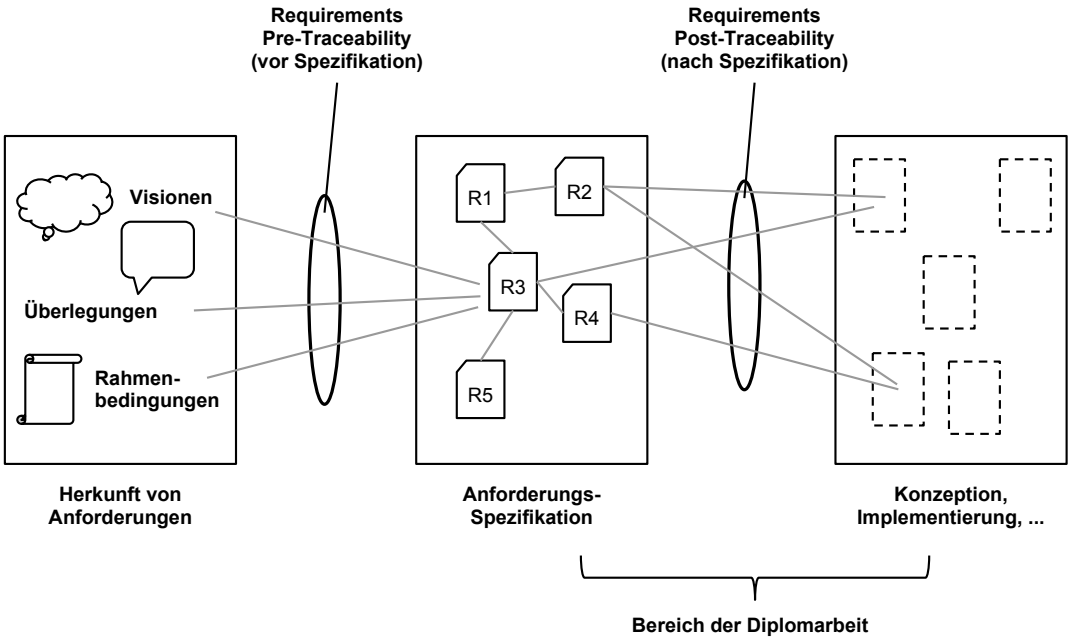


Abbildung 2.1: Rückverfolgbarkeit von Anforderungen vor und nach Spezifikation [eigene Darstellung in Anlehnung an Poh96]

Da zum Zeitpunkt der Herstellung und Wartung oft nicht bekannt ist, ob und wann die Beziehungen *genutzt* werden, ergeben sich häufig Akzeptanzprobleme bei der Einführung von Lösungen, die die Rückverfolgbarkeit ermöglichen. Hier handelt es sich um eine Herausforderung die bereits oft in der Literatur diskutiert wurde (vgl. 2.3.2 Kosten, Nutzen und Akzeptanz, Seite 9). Die Entwicklung zur Optimierung der Rückverfolgbarkeit dieser Diplomarbeit bildete dabei keine Ausnahme, wie in Kapitel 2.3.2 weiter ausgeführt wird.

Während der Konzeption der Implementierungsaufgabe dieser Diplomarbeit hat es sich ergeben, dass bereits vorhandene Ideen zur Herstellungen und Wartung von Beziehungen beiseite geschoben wurden, um zunächst die Anforderungen der Nutzung zu definieren, welche ihrerseits Anforderungen an die Erfassungsmethode stellten.

2.2.1 Herstellung von Beziehungen

Beim Herstellen einer Beziehung werden zwei oder mehr Artefakte miteinander verlinkt, um diese Beziehung für die spätere Nutzung bereitzustellen. Hierbei existieren manuelle, automatische und halbautomatische Erfassungsmethoden. [Nach GCHH12, S. 15]

Manuelle Erfassung wird durch den Nutzer erledigt, beispielsweise durch manuelles Markieren oder das Eintragen von eindeutigen Bezeichnern (IDs) der Artefakte in eine Datenbank.

Von *automatischer Erfassung* wird gesprochen, „wenn Rückverfolgbarkeit durch automatische Techniken, Methoden und Tools erreicht wird“ [GCHH12, S. 10][2]. Ein konkretes Beispiel ist das Projekt von A. Lucia et al [LMOP12]. Hier werden Anforderungsdokumente mit Data-Mining-Methoden durchsucht um Beziehungen zwischen Anforderungen herzustellen.

Oft kämpfen rein automatische Erfassungsmethoden mit sog. „False positives“ (falsche Beziehung erfasst) und „False negatives“ (tatsächliche Beziehung wurde nicht erkannt). In solchen Fällen eignen sich *halbautomatische Erfassungsmethoden*, die den Menschen teilweise in die Erfassung von Beziehung involvieren. Beispielsweise werden dem Nutzer Kandidaten angezeigt, die er bestätigen muss. [Vgl. GCHH12, S. 15] Der Prototyp „traceMAINTAINER“ in der Dissertation von P. Mäder verfolgt den Werdegang von Beziehungen zwischen Artefakten in UML-Dokumenten. Neben der automatischen Funktionsweise wird dem Nutzer die Möglichkeit geboten, die Erfassung von angebotenen Beziehungen zu bestätigen oder abzulehnen. [Vgl. Mä09, S. 136]

2.2.2 Wartung von Beziehungen

Im Projektverlauf passiert es oft, dass neue Anforderungen hinzukommen, andere fallen gelassen oder ersetzt werden. Anforderungsdokumente werden aufgeteilt, vereinigt oder gelöscht. Das gleiche kann mit Artefakten passieren, die mit Anforderungen in Beziehung stehen. Beispielsweise unterliegt der Quellcode in Softwareprojekten während Entwicklungs- und Wartungsarbeiten einer permanenten Veränderung.

Dies geschieht nicht ohne Auswirkung auf die Beziehungen zwischen den Anforderungsdokumenten. Die Qualität der späteren *Nutzung* hängt maßgeblich von der Korrektheit und Vollständigkeit der vorhandenen Beziehungsinformationen ab. Daher müssen nach Änderungen von Artefakten die erfassten Beziehungen nachgeführt werden. [Vgl. MRP06]

„Die Wartung von Beziehungen umfasst Aktivitäten, die existierende Beziehungen aktualisieren oder – sofern nötig – neue Beziehungen herstellen, um die Rückverfolgbarkeit relevant und aktuell zu halten.“ [GCHH12, S. 15][3]

In diese Diplomarbeit flossen mehrere Überlegungen ein, um den Aufwand für die Pflege von Beziehungen gering zu halten. Insbesondere bei der Auswahl einer geeigneten Datenquelle für Beziehungsinformationen spielte dieser Faktor eine große Rolle (vgl. 5 Lösungsansätze, Seite 29).

[2] Vom Autor ins Deutsche übersetzt
[3] Vom Autor ins Deutsche übersetzt

2.2.3 Nutzung von Beziehungen

Sind die Beziehungen erfasst, können Aufgaben im Bereich der Softwaretechnik unterstützt werden, wie z. B. Verifizierung und Validierung von Anforderungen, Einflussanalyse und Änderungsmanagement [nach GCHH12, S. 17]. In dieser Diplomarbeit zählten Aufgaben der Projektüberwachung, Risikoabschätzung und Abdeckungsanalyse zu den Anwendungsfällen (vgl. 4.1 Anwendungsfälle, Seite 22).

Neben der Einordnung nach dem Zeitpunkt, vor oder nach der Anforderungsspezifikation (vgl. 2.1 Einordnung), lässt sich die Nutzung der Rückverfolgung nach der Richtung einteilen. *Vertikale Verfolgung* findet zwischen Artefakten „unterschiedlicher Abstraktionsebenen“ statt – etwa um den Softwareprojekt-Lebenszyklus oder eine Rückverfolgung von einem zum anderen Ende abzubilden. Die Richtung gibt dabei die Ausrichtung aus Sicht der Quelle an – also vorwärts von der Anforderung zu späteren Artefakten im Software-Entwicklungsprozess (z. B. Konzeption, Quellcode) oder beispielsweise vom Quellcode zurück zur Anforderung. Findet die Verfolgung zwischen Artfakten einer Abstraktionsebene statt, wird von *horizontaler Verfolgung* gesprochen [nach GCHH12, S. 19].

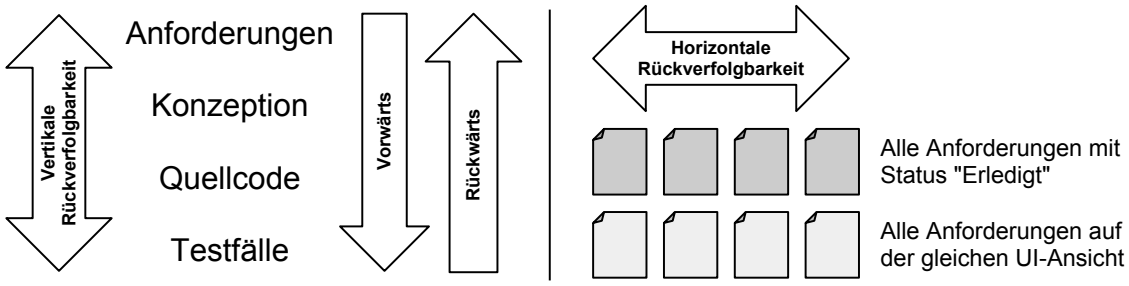


Abbildung 2.2: Horizontale und Vertikale Rückverfolgbarkeit

In dieser Diplomarbeit findet sowohl vertikale als auch horizontale Rückverfolgung statt. Die Implementierung dieser Diplomarbeit ermöglicht das Nachverfolgen von Komponenten der Benutzerschnittstelle *zurück* zu Anforderungen. Horizontale Rückverfolgbarkeit wird durch die Selektion von Anforderungen erreicht, die sich auf der gleichen Maske der grafischen Benutzeroberfläche befinden. Eine im Prototyp implementierte Funktion, die zu jeder Anforderung eine Historie von Ereignissen anzeigt, ist ebenfalls eine horizontale Rückverfolgung.

2.3 Aktuelle Herausforderungen in Bezug auf diese Diplomarbeit

In den folgenden Abschnitten identifiziert der Autor Herausforderungen der Rückverfolgbarkeit von Anforderungen, die in dieser Diplomarbeit eine große Rolle spielen. Zu dieser Auswahl führten Erkenntnisse aus der Ist- und Soll-Zustand-Analyse, Literaturquellen und die Evaluation existierender Projekte oder Produkte, die Rückverfolgbarkeit in einer bestimmten Weise implementieren. Das letzte Unterkapitel 2.3.4 bildet eine Zusammenfassung der aktuellen Herausforderungen auf dem Gebiet, indem eine hypothetische Betrachtung des Zustands der Rückverfolgbarkeit im Jahr 2035 von Gotel et al. wiedergegeben wird.

2.3.1 Heterogene Artefakte

In dieser Diplomarbeit sollen Beziehungen zwischen zwei Artefakttypen hergestellt und genutzt werden: Komponenten in der Benutzerschnittstelle einer Webapplikation und durch eine Projektmanagementsoftware verwaltete Anforderungen. Mit dem Versionierungssystem „Git“ stand zudem eine weitere Datenquelle zur Verfügung, die Quellcode-Versionsstände als dritten Artefakttyp anbot.

Jede der drei Artefakttypen stammte aus einem anderen System und war über eine andere Art referenzier- und damit suchbar. Die Herausforderung bestand darin, heterogene Artefakte über Systemgrenzen hinweg nachzuverfolgen. Während die Anforderungen durch eindeutige Bezeichner (IDs) leicht referenziert und durch eine von Redmine angebotene Schnittstelle[4] abgerufen werden konnten, mussten zusätzliche Wege und Kompromisse gefunden werden, um zur Laufzeit instanziierte Komponenten der Benutzerschnittstelle ebenfalls zu referenzieren und mit den Anforderungen in Verbindung zu bringen.

2.3.2 Kosten, Nutzen und Akzeptanz

Steigt der Aufwand zur Herstellung und Wartung von Beziehungen, steigen auch die Erwartungen, dass die Nutzung der erfassten Beziehungen bedeutend Kosten im Software-Projekt einspart – zumindest aber sollte die Investition wieder eingespielt werden, um die Akzeptanz für eine Rückverfolgbarkeitslösung zu wahren.

Leider lässt sich der tatsächliche Gewinn nicht ohne zusätzlichen Aufwand akkurat messen [vgl. IR12, S. 29-32], während die Kosten (Zeitaufwand) für die Herstellung und

[4] Redmine bietet eine REST-Schnittstelle an, die Zugriff auf den Datenbestand von Außen ermöglicht [Red12b].

Wartung von Beziehungen sofort ersichtlich sind. Durch die nicht unmittelbare Messbarkeit des Nutzens, mehren sich leicht Zweifel, ob sich der Aufwand tatsächlich lohnt. Die wahrgenommene Akzeptanz einer zu etablierenden Rückverfolgbarkeitslösung kann gefährdet sein.

Nach Aizenbud-Reshef et al. und Arkley et al. werden die Kosten für das manuelle Erstellen und Warten von Rückverfolgbarkeitsinformationen als allgemeiner Grund für den geringen Einsatz von Rückverfolgbarkeit in der Industrie genannt, obwohl die Vorteile bekannt sind [vgl. ARNR06, S. 516 und AMR01].

Arkley et al. heben hervor, dass es wichtiger ist „die Art und Weise zu unterstützen wie Software- und Systementwickler arbeiten, als neue Prozesse zu etablieren." [AMR01][5]

Es zeigt sich deutlich, dass (a) die Kosten für das Erfassen und Warten von Beziehungen gering gehalten werden und (b) diese Aktivitäten möglichst passend im Software-Entwicklungsprozess integriert sein müssen.

Ein weiteres Akzeptanzproblem liegt nach Aizenbud-Reshef et al. darin, dass „Rückverfolgbarkeit meist nicht von den gleichen Personen genutzt wird, die die Beziehungen erfassen, was die Motivation derer verringert, die Rückverfolgbarkeitsinformationen erfassen und warten." [ARNR06][6]

2.3.3 Granularität von Beziehungen

Die Granularität von Beziehungen bezeichnet nach Gotel et al. den „Detailgrad, wonach eine Beziehung aufgenommen und verarbeitet wird" [GCHH12, S. 20][7]. Beispielsweise kann die Granularität bei der Verlinkung von Anforderungen mit Quellcode unterschiedlich fein ausfallen, indem entweder Packages, Klassen oder Methoden referenziert werden.

Da Beziehungen immer vorbereitend erfasst werden müssen, ist es möglich, dass es nie zur Nutzung kommt und umsonst Aufwand für die Erfassung entstanden ist. Hier gilt es die richtige Balance zu wahren und die Qualität, wie auch Quantität der erfassten Beziehungen entsprechend den Anforderungen zu variieren. Egyed et al. veröffentlichte hierzu seine Erkenntnisse aus drei Studien. Demnach konnten insbesondere durch das selektive Verringern der Granularität von Beziehungen Kosten verringert werden (Vgl. [EGHB09, S. 6]).

Ziel dieser Diplomarbeit ist u. a. die Etablierung einer bedarfsgerechten Lösung, die einen guten Kompromiss in der Granularität von Beziehungen zu erreicht.

[5] Vom Autor ins Deutsche übersetzt
[6] Vom Autor ins Deutsche übersetzt
[7] Vom Autor ins Deutsche übersetzt

2.3.4 The Grand Challenge of Traceability (v1.0)

Die vom Autor ausgewählten Themen spielen auch in den folgenden von O. Gotel et al. in [GCHH+11, S. 349] definierten Herausforderungen eine Rolle. Es handelt sich hier um eine all-umfassende Liste von hypothetischen Annahmen, die sich auf Eigenschaften der Rückverfolgbarkeit von Anforderungen im Jahr 2035 beziehen. Sie adressiert aktuelle Probleme und ist zudem als Empfehlung für zukünftige Forschungen bestimmt.[8]

1. *Bedarfsgerecht (Purposed).* Die Rückverfolgbarkeit unterstützt die Anforderungen der Stakeholder.
2. *Kosteneffizient (Cost-effective).* Kosten und Nutzen stehen in einem adequaten Verhältnis.
3. *Konfigurierbar (Configurable).* Rückverfolgbarkeit kann je nach Bedarf etabliert und sich verändernden Anforderungen angepasst werden.
4. *Zuverlässig (Trusted).* Alle Stakeholder vertrauen der etablierten Rückverfolgbarkeit, die angesichts von Inkonsistenzen, Auslassungen und Änderungen aufgebaut und gewartet wird. Sie können und werden sich auf die bereitgestellte Rückverfolgbarkeit verlassen.
5. *Skalierbar (Scalable).* Unterschiedliche Arten von Artefakten können unter Anwendung variabler Feinheitsgrade (Granularität) in verschiedener Quantität verfolgt werden, während sich die Rückverfolgbarkeit über organisatorische und geschäftliche Grenzen hinweg ausbreitet.
6. *Portabel (Portable).* Rückverfolgbarkeit kann ausgetauscht, vereinigt und wiederverwendet werden – in und zwischen Projekten, Organisationen, Domänen, Produktlinien und unterstützenden Werkzeugen.
7. *Geschätzt (Valued).* Rückverfolgbarkeit ist eine strategische Priorität und wird von von allen geschätzt. Jeder Stakeholder hat eine Rolle zu erfüllen und verrichtet aktiv seine Verantwortlichkeiten.
8. *Omnipräsent (Ubiquitous).* Rückverfolgbarkeit ist jederzeit vorhanden, ohne dass dessen Präsenz aktiv wahrgenommen wird, da es in den Software-Engineering-Prozess eingebettet ist.

[8] Liste vom Autor ins Deutsche übersetzt

3 Ist-Analyse

Zur Ermittlung des Ist-Zustands wurden beim Praxispartner Interviews mit den Mitarbeitern geführt, sodass jede Rolle mindestens einmal befragt wurde.[9] Zusätzlich floss die persönliche Projekterfahrung des Autors im Unternehmen mit ein.

Während die in Kapitel 3.1 beschriebenen Kenndaten das Unternehmen näher vorstellen, werden in Kapitel 3.2 die Rollen und Prozesse skizziert, die durch diese Diplomarbeit unterstützt werden sollen. Der für die Implementierungsarbeit relevante Ist-Zustand wird in 3.3 Softwareunterstützung der Prozesse vorgestellt.

3.1 Kenndaten des Unternehmens

Mit 24 Beschäftigten im Jahr 2012 gehört die intecsoft zu den mittelständischen Unternehmen der Branche.

Die intecsoft berät Unternehmen der Energie-, Finanzwirtschafts- und Medizinbranche in Fragen der Modellierung von Prozessen, um diese anschließend in der IT-Infrastruktur abzubilden. Im Regelfall möchte der Kunde seine bestehenden Geschäftsprozesse, die bisher noch per Hand oder mit unterschiedlichen Softwaresystemen erledigt wurden, konsolidieren, aufeinander abstimmen und effizienter gestalten.

Neben der Prozessberatung werden Entwicklungsdienstleistungen angeboten. Die Umsetzung von Geschäftsprozessen erfolgt dann entweder als Integration in bestehenden IT-Infrastrukturen (Hard- und Software) oder als eigenständige, kundenspezifische IT-Lösung. Zum Einsatz kommen je nach Anforderungen und Rahmenbedingungen Standardsoftware von Partnern (SAP, Oracle) oder Open Source Software (Linux, Apache Wicket, Hibernate).

Die Art des Projektmodells wird in der Regel vom Kunden vorgegeben. Die meisten Projekte werden nach dem Wasserfall- oder V-Modell durchgeführt, wobei auch agile Entwicklungsmethoden angeboten werden.

3.2 Prozesse und Rollen

Wie bereits in 3.1 Kenndaten des Unternehmens beschrieben, werden die meisten Projekte nach dem Wasserfall- oder V-Modell durchgeführt. Im Folgenden werden die internen

[9] Die Interviews wurden durch einen Fragebogen unterstützt, siehe A Interviewfragen Ist-Zustand, Seite 61

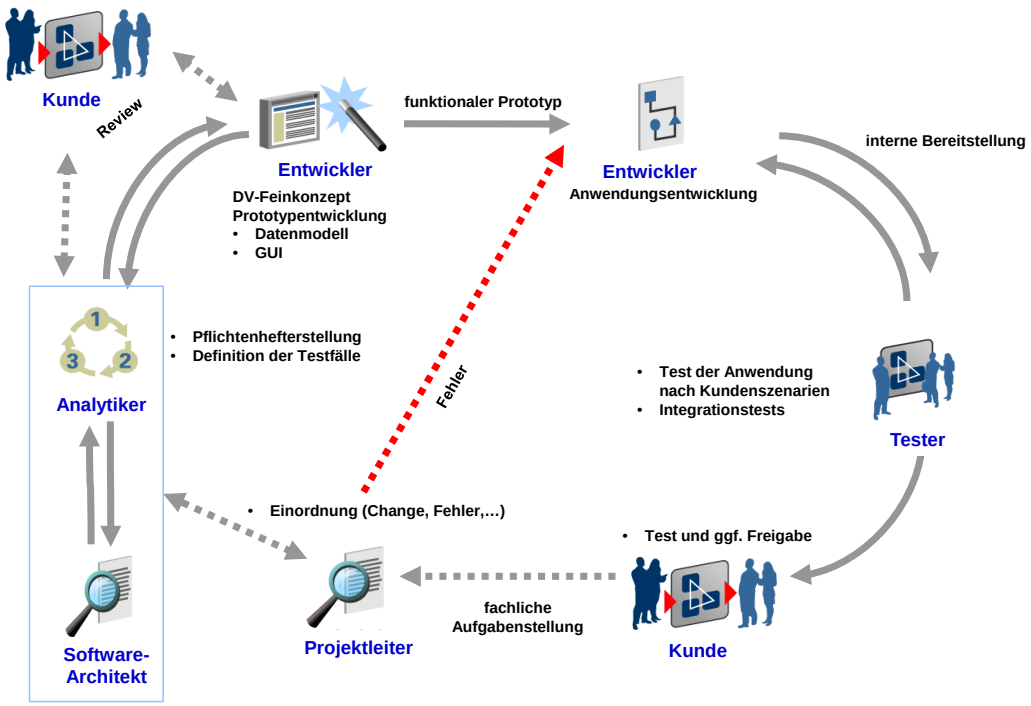


Abbildung 3.1: Überblick über Prozesse und Rollen der intecsoft [Nach Jun11, S. 22]

Prozesse und Rollen der intecsoft zur Bewältigung dieser Modelle beschrieben und in Abbildung 3.1 veranschaulicht.

Zu Beginn des Projekts äußert der Kunde dem Projektleiter, der oft von einem Analytiker für Geschäftsprozesse unterstützt wird, die fachliche Aufgabenstellung des zu lösenden Problems. Nur im seltenen Fall hat der Kunde schon ein Lastenheft vorliegen, daher wird er auch bei der Anforderungserhebung begleitet. Meist existiert zu diesem Zeitpunkt schon eine Beziehung zum Auftraggeber, sodass der Projektleiter bereits Einblick in dessen Geschäftsprozesse hat. Beispielsweise wurde bereits ein Workshop zur Prozessdefinition durchgeführt oder es handelt sich um ein Folgeprojekt.

Analytiker und Software-Architekt definieren im nächsten Schritt, wie die Anforderungen aus Sicht des Auftragnehmers umgesetzt werden. Während der Analytiker das Datenmodell entwirft, um die Prozesse später abbilden zu können, legt der Software-Architekt die technische Umsetzung des Systems fest – beispielsweise welche Technologien für die Persistenzschicht, Geschäftslogik oder die Benutzeroberfläche verwendet werden. Gemeinsam mit dem Projektleiter arbeiten sie das Pflichtenheft aus, das dem Kunden zum Review gegeben wird.

Basierend auf dem Pflichtenheft wird ein Prototyp angefertigt, der visualisiert, wie der Auftragnehmer das System umsetzen möchte. Der Prototyp, der sich meist auf die

Eingabemasken beschränkt, wird anschließend dem Kunden zur Abnahme vorgestellt. Gelegentlich deckt der Prototyp abweichende Annahmen zwischen Auftragnehmer und Auftraggeber auf, die zu unterschiedlichen Projekterwartungen und -ergebnissen geführt hätten. Werden solche Abweichungen erst später entdeckt, sind unvorhergesehene zusätzliche Kosten die Folge.

Wurden Risiken (z. B. Missverständnisse, unklare Anforderungen, unklare Prozesse) durch regelmäßige Reviews des Pflichtenhefts und Prototyps minimiert, erstellt ein Entwickler ein technisches Feinkonzept (DV-Feinkonzept), welches er in detaillierte Implementierungsaufgaben unterteilt.

Diese Aufgaben arbeiten die Entwickler in der Implementierungsphase ab und weisen sie anschließend der Qualitätssicherung zum Testen zu. Nachdem intern festgestellt wurde, dass die Funktionalitäten den Anforderungen entsprechen, wird die Anwendung dem Kunden mit dem Ziel der Freigabe bereitgestellt.

Sollte der Kunde Fehler feststellen, muss ermittelt werden, ob es sich hier um neue Anforderungen oder tatsächlich um einen Fehler handelt. Dies wirkt sich darauf aus, wer die Kosten für den Mehraufwand trägt.

3.3 Softwareunterstützung der Prozesse

Um die Geschäftsprozesse im Projektverlauf zu dokumentieren und somit *nachverfolgbar* zu machen, ist spezielle Software nötig. Zur Dokumentation auf Ebene der Anforderungen kommt die Projektverwaltungssoftware Redmine zum Einsatz. Auf Ebene des Quellcodes wird das Versionsverwaltungssystem Git eingesetzt. Im Folgenden wird beschrieben *wie* diese Werkzeuge im Projektalltag verwendet werden, um die in Kapitel 3.2 skizzierten Prozesse zu unterstützen.

3.3.1 Redmine

Die Anforderungen werden in der Projektmanagementsoftware Redmine – herunter gebrochen in feingranulare Implementierungsaufgaben – als „Tickets“ abgelegt. Ein Ticket repräsentiert einen Vorgang im Projekt. Es enthält Felder, um die zu implementierende Anforderung und den Werdegang der Umsetzung zu dokumentieren (Vgl. Abbildung 3.2). So kann es einem Mitarbeiter zugewiesen werden, der es bearbeitet und anschließend weiterleitet. Am Ticketstatus lässt sich die aktuelle Station im Prozess ablesen. Wie in anderen Projektmanagementwerkzeugen auch (z. B. Microsoft Project) können Beziehungen zwischen Tickets spezifiziert werden, etwa Nachfolger- und Vorgängerbeziehungen. Am häufigsten zum Einsatz kommen die Beziehungstypen „blockiert von“ um eine nötige Zuarbeit oder „in Beziehung mit“ um eine thematische Beziehung anzuzeigen.

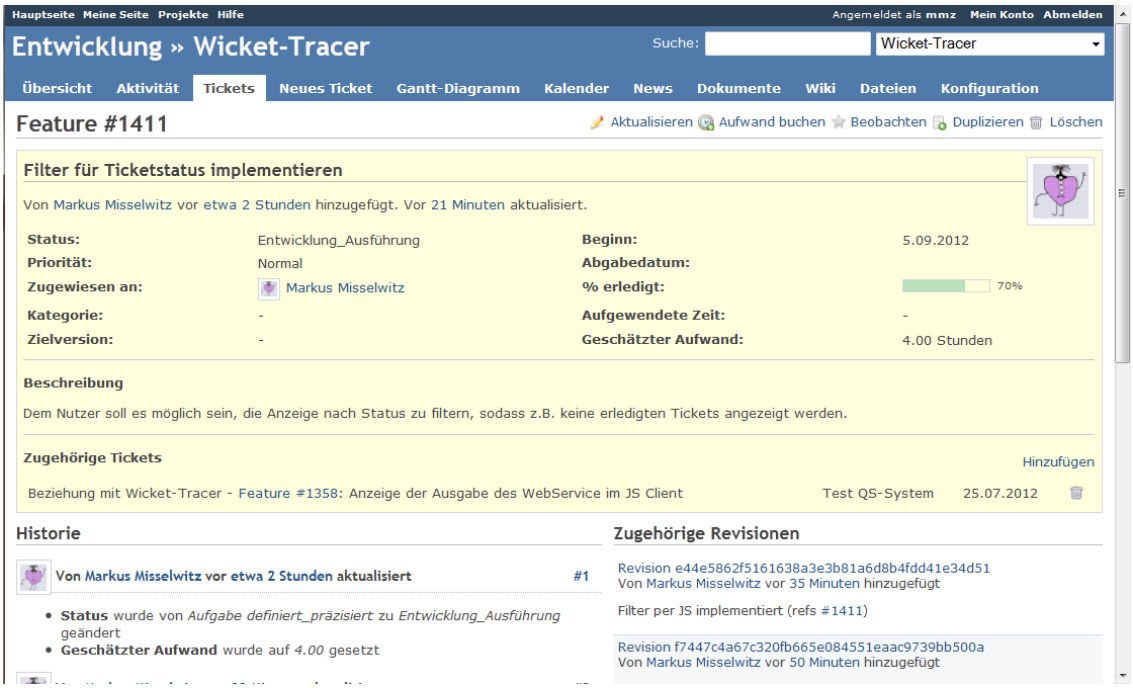


Abbildung 3.2: Bildschirmfoto eines in Redmine angelegten Tickets

Folgendes Beispielszenario verdeutlicht den Einsatz des Systems im Projektalltag (Vgl. Abbildung 3.3): Der Projektleiter erstellt ein neues Ticket, beschreibt eine Funktion, die es zu implementieren gilt und weist es einem Entwickler zu. Sobald der Entwickler sich der Aufgabe annimmt, setzt der den Status auf „Entwicklung_Ausführung“. Nach Fertigstellung der Implementierung spielt der Entwickler seine Arbeit auf das intern vorhandene Abbild des Softwaresystems des Kunden (Integrationssystem) ein und weist das Ticket dem Tester zu, sodass dieser überprüfen kann, ob die neue Funktionalität fehlerfrei ist und den Anforderungen genügt. Gibt es noch etwas nachzubessern, verweist der Tester das Ticket zurück an den Entwickler und setzt den Status erneut auf „Entwicklung_Ausführung“, andernfalls wird der Status als „QS-Stand bereitstellen“ markiert. Dies signalisiert der Rolle des Systemintegrators, dass die Funktionalität nun auf das Testsystem beim Kunden gespielt werden kann. Sobald dies erfolgt, steht das Ticket so lange im Status „Test QS-System“ bis die Tests des Kunden bestanden sind. Wenn auch dieser Test bestanden ist, kann die Funktionalität in das Produktivsystem überführt werden (Status „In Produktion überführen“). Im negativen Fall muss der Projektleiter, wie in 3.2 Prozesse und Rollen beschrieben, entscheiden, ob es sich um einen Fehler oder Change handelt und das Ticket entsprechen weiterleiten. Ist die im Ticket beschriebene Funktionalität im Produktivsystem aktiv, wird das Ticket auf „Erledigt“ gesetzt.

In Redmine können nicht nur die Vorgänge, sondern auch statische Dokumente in Form einer Wiki-Seite oder Datei hinterlegt werden. Diese Funktion wird genutzt, um das Pflichtenheft und andere globale Dokumente wie Installationsanleitungen abzuspeichern. Im Wiki gepflegte Dokumente haben den Vorteil, dass hier Links zu Tickets gesetzt

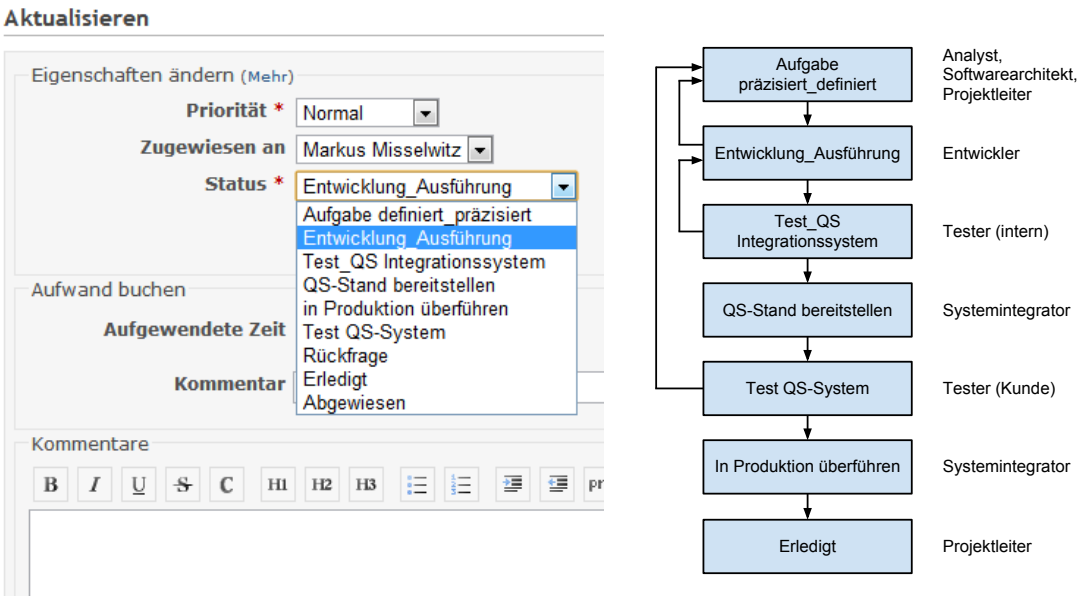


Abbildung 3.3: Ticketstatus dokumentiert die Station im Prozess

werden können (Vgl. 3.3.3 Etablierte Rückverfolgbarkeit).

Als Projektmanagementsoftware unterstützt Redmine selbstverständlich weitere Funktionen wie ein Gantt-Diagramm, Meilensteine oder Auslieferungsversionen denen Tickets zugewiesen werden können. Dies wird in dieser Diplomarbeit jedoch nicht weiter ausgeführt.

3.3.2 Git

Das verteilte Versionsverwaltungssystem Git wird eingesetzt, um den Quellcode der Softwareprojekte zu verwalten. Der im Beispielszenario in Kapitel 3.3 genannte Entwickler spielt seine neu entwickelte Funktionalität *mithilfe von Git* in den Quellcode des Integrationssystem ein. Dazu erstellt der Entwickler einen neuen Versionsstand (Commit) in dem die Dateiänderungen und ein Kommentar erfasst werden. Anschließend wird der Commit an das zentrale Quellcodeverzeichnis auf dem Server übermittelt (Push).

Im Unternehmen gilt die Regelung, dass in den Kommentaren von Versionsständen die Ticketnummer zu erfassen ist, auf die sich die Quellcodeänderung bezieht. Von einem Ticket in Redmine können dann die zugehörigen Änderungen im Quellcode nachvollzogen werden (Vgl. Abbildung 3.2, „Zugehörige Revisionen“).

3.3.3 Etablierte Rückverfolgbarkeit

In der Analyse des Ist-Zustands wurde ermittelt, dass Rückverfolgbarkeit bereits auf Ebene der Anforderungsdokumente und des Quellcodes etabliert ist, indem zwischen

den Artefaktetypen „Lastenheft“, „Pflichtenheft“, „Tickets“ und „Versionsstände“ Beziehungen bestehen (vgl. 3.4).

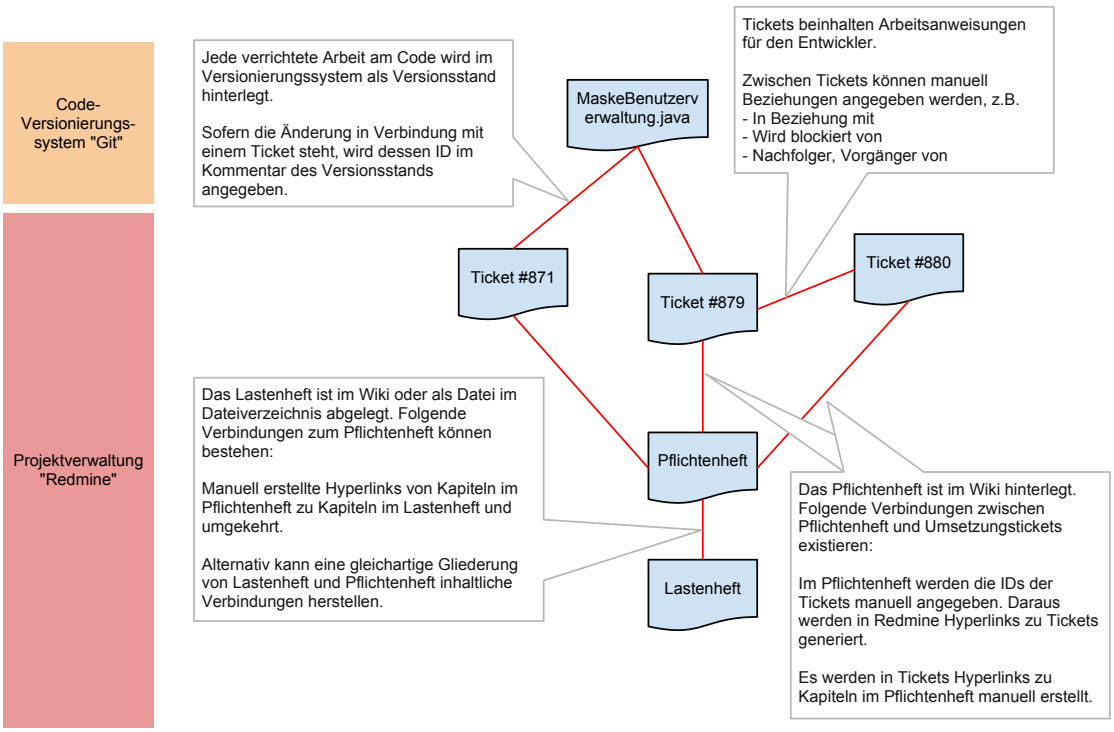


Abbildung 3.4: Verlinkungen der bereits etablierten Rückverfolgbarkeit

Die Herstellung von Beziehungen zwischen Quellcode-Artefakten und Tickets ist im Software-Entwicklungsprozess verankert, indem eindeutige Bezeichner von Tickets (IDs) in Kommentaren von Versionsständen erfasst werden (Vgl. 3.3.2 Git).

Verlinkungen zwischen Tickets und Anforderungsdokumenten (Pflichtenheft) werden manuell erstellt, sodass die Ursprünge von Anforderungen während der Implementierungsphase noch ersichtlich sind (Rückverfolgung rückwärts) – gleichzeitig aber auch überprüft werden kann, welche im Pflichtenheft definierten Funktionalitäten durch welche Implementierungsaufgaben realisiert wurden (Rückverfolgung vorwärts).

Beziehungen zwischen Tickets werden verwendet, um die Entwicklungsarbeit zu strukturieren und zugehörige Informationen zu bündeln.

3.4 Defizite in der Rückverfolgbarkeit von Anforderungen im Unternehmen

In Gesprächen mit Mitarbeitern wurden folgende Defizite identifiziert:

- Die Masse an Tickets mit durchschnittlich 500 Tickets pro Projektjahr ist mit fortschreitender Projektdauer immer schwieriger zu überblicken (Vgl. Abbildung 1.1 Liste von Anforderungen in Redmine, Seite 2).
- Der Bezug von einem Ticket zur implementierten Funktionalität in der Anwendung wird zwar verbal beschrieben und mit Stellen im Quellcode der Applikation verlinkt, jedoch ist keine Beziehung zur Funktionalität in der Benutzerschnittstelle vorhanden. Allerdings spielt die Benutzerschnittstelle in der Diskussion über Funktionalitäten eine große Rolle – insbesondere mit dem Kunden bildet sie oft die einzige Diskussionsgrundlage.
- Aufgrund der textuellen Notation von Anforderungen, muss auch die Suche danach textbasiert erfolgen. Da die Sprache jedoch vielfältige Ausdrucksweisen für den gleichen Sachverhalt bietet, ist es möglich, dass trotz einer inhaltlich gleichen Suchanfrage keine Ergebnisse geliefert werden. Somit ist die Wiederauffindbarkeit von Tickets erschwert.
- Es existiert keine befriedigende Möglichkeit, Tickets zu selektieren, die aufgrund der Benutzerschnittstelle miteinander in Beziehung stehen.
- Später hinzu gekommene Einzelanforderungen berücksichtigen nicht immer alle vorherigen Überlegungen. Eine engere Darstellung der Einzel-Informationen zusammen könnte ein besseres Bild über frühere Anforderungen ergeben.

4 Soll-Zustand

In diesem Kapitel wird der Soll-Zustand der Implementierungsaufgabe dieser Diplomarbeit beschrieben. Treibende Kräfte für dieses Vorhaben, an denen sich die Anforderungen dieser Diplomarbeit ableiten, sind der Wunsch nach:

- Steigerung der Wahrnehmung von Anforderungen
- Steigerung der Verfügbarkeit von Anforderungen
- Steigerung der Auffindbarkeit von Anforderungen
- Steigerung der Rückverfolgbarkeit von Anforderungen, ohne zusätzlichen Zeitaufwand für die Herstellung und Pflege von Rückverfolgbarkeitsbeziehungen zu investieren
- Integration in den bereits etablierten Software-Entwicklungsprozess

In Kapitel 4.1 werden die Anwendungsfälle anhand des Software-Entwicklungsprozess definiert. Daraus leiten sich die in Kapitel 4.2 aufgestellten Anforderungen ab, deren Umsetzung innerhalb der in Kapitel 4.3 festgelegten Technische Randbedingungen stattfinden muss.

4.1 Anwendungsfälle

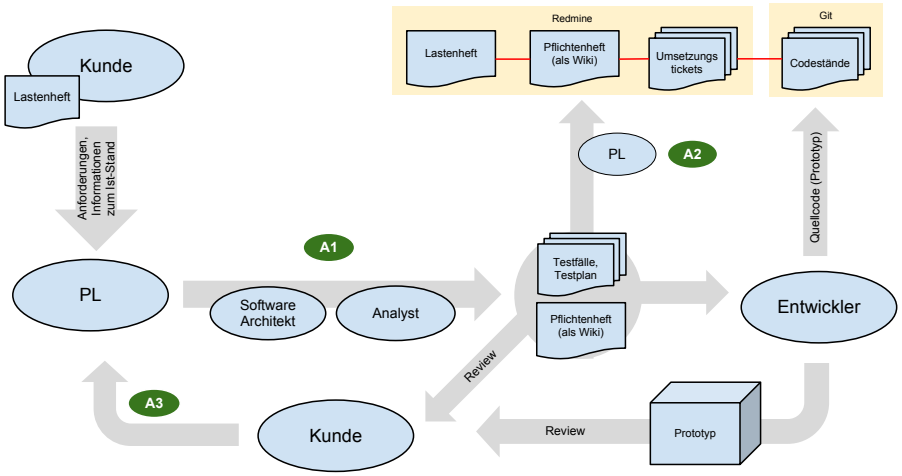


Abbildung 4.1: Anwendungsfälle anhand der Prozesse der Anforderungsanalyse und Konzeption

Nr.	Anwendungsfall
A1	Bisherige Anforderungen, Überlegungen, Entscheidungen und verrichtete Arbeiten werden gesichtet, um Kosten und Risiken einzuschätzen, sowie Konflikte und unnötige Neuentwicklungen zu vermeiden.
A2	Projektleiter ermittelt, ob alle Anforderungen der Maske vorhanden sind. (Vollständigkeit der Anforderungen)
A3	Projektleiter hat im Gespräch mit dem Kunden schnellen Zugriff auf die Anforderungsdokumentation, indem er die Tickets auf der Benutzerschnittstelle sieht.
A4	Projektleiter überwacht permanent den Status der Tickets einer Maske und erkennt u. a. zeitnah den kritischen Pfad.
A5	Entwickler sucht Informationen in anderen Tickets und deren zugehörigen Artefakten, die mit seiner Implementierung in Beziehung stehen.
A6	Tester prüft, ob alle Anforderungen korrekt und vollständig umgesetzt wurden. (Vollständigkeit der Umsetzung)
A7	Tester sucht Tickets, die mit einem Fehler in Verbindung stehen können, um nicht unnötig ein neues Ticket zu öffnen.
A8	Projektleiter prüft, ob alle Tickets/Anforderungen für einen Meilenstein getestet wurden. (Vollständigkeit der Tests)
A9	Projektleiter ermittelt die Kosten eines Change unter Einbeziehung aller betroffenen Tickets einer Maske.

Tabelle 4.1: Anwendungsfälle

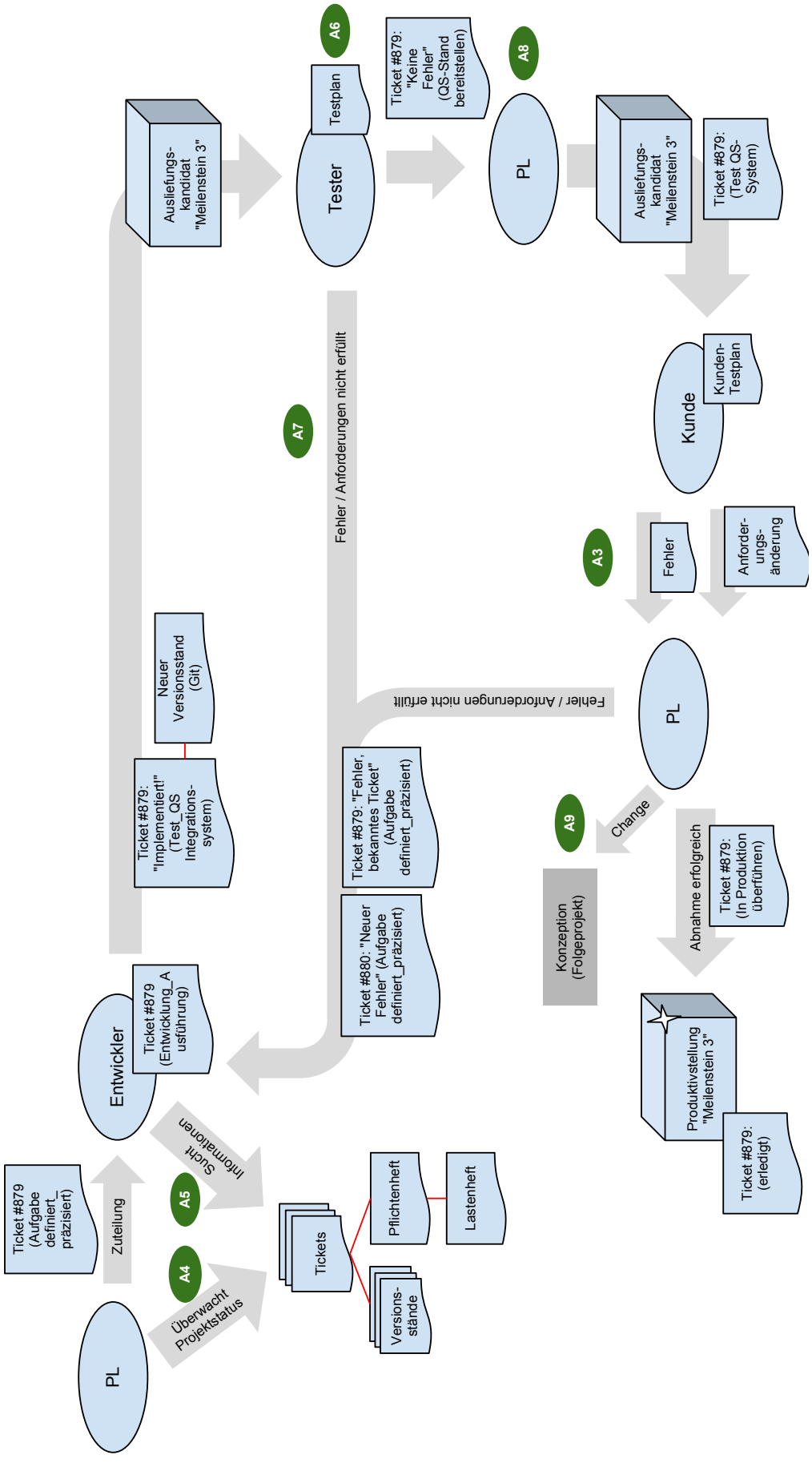


Abbildung 4.2: Anwendungsfälle anhand der Prozesse der Implementierung, Test und Auslieferung

4.2 Anforderungen

Aus den dargelegten Anwendungsfällen leiten sich die in Kapitel 4.2.1 aufgestellten funktionalen Anforderungen ab, die von den Lösungsansätzen implementiert werden müssen. 4.2.2 Nicht-Funktionale Anforderungen umfasst eine Liste der wichtigsten Qualitätsziele, die es zu erreichen gilt.

4.2.1 Funktionale Anforderungen

Nr.	Funktionale Anforderung
F1	Es sollen Tickets der Projektmanagementsoftware „Redmine“, die zu einer Komponente in der Benutzerschnittstelle gehören, zur Laufzeit der Anwendung in der Benutzerschnittstelle erscheinen.
F2	Für jedes Ticket in der Benutzerschnittstelle sollen Titel, Status und Verweis auf die Vollansicht in Redmine erscheinen.
F3	Die Methoden zur *Erfassung* und *Wartung* der dafür nötigen Beziehungen zwischen Anforderungen und Benutzerschnittstelle müssen a) die relevanten Anwendungsfälle unterstützen, b) in bestehende Arbeitsabläufe des Software-Entwicklungsprozesses integriert sein und c) intuitiv und schnell zu bedienen sein.

Tabelle 4.2: Funktionale Anforderungen

4.2.2 Nicht-Funktionale Anforderungen

Nr.	Priorität	Ziel	Erläuterung
NF1	Hoch	Bedienbarkeit	• Intuitive Bedienung und einfache Navigation. • Es soll keine Schulung nötig sein.
NF2	Mittel	Performanz	• Die Anwendung, deren Anforderungen visualisiert werden, darf um nicht mehr als 20% verlangsamt werden.
NF3	Mittel	Installierbarkeit	• Die Anwendung muss von Fachpersonal mithilfe einer Dokumentation ohne zusätzlichen Aufwand installiert und konfiguriert werden können.
NF4	Mittel	Verfügbarkeit	• Verfügbarkeit mindestens genauso groß wie die schlechteste Verfügbarkeit der Nachbarsysteme. • Beeinträchtigungen durch Wartungsarbeiten sind ohne Ankündigung erlaubt. • Die Verfügbarkeit von Nachbarsystemen darf nicht beeinträchtigt werden.
NF5	Mittel	Änderbarkeit	• Die Entwicklung soll so strukturiert und dokumentiert sein, dass eine Wartung oder Weiterentwicklung auch für fremde Entwickler möglich ist.
NF6	Gering	Sicherheit	• Es gelten die gleichen Sicherheitsanforderungen wie für die Nachbarsysteme. Vorzugsweise sind deren Mechanismen zur Authentifizierung zu verwenden. • Es soll keine zusätzlichen Authentifizierungsmöglichkeiten geben.

Tabelle 4.3: Nichtfunktionale Anforderungen

4.3 Technische Randbedingungen

Die Entwicklung findet in einem eng abgesteckten Bereich des Software-Entwicklungsprozesses der intecsoft statt. Insbesondere die vorhandenen Nachbarsysteme legen Randbedingungen fest, die sich stark auf die Wahl der Lösungsmöglichkeit auswirken werden.

Die vorgegebenen Systeme sind eine Webapplikation, deren Benutzerschnittstelle mithilfe des Frameworks „Apache Wicket“ codiert wurde, die Projektmanagementsoftware „Redmine“ und das Versionierungssystem „Git“.

Alle genannten Nachbarsysteme sind für den serverseitigen Einsatz bestimmt[10], sodass die Entwicklung dieser Diplomarbeit sich in die Client-Server-Struktur einfügen muss (Vgl. Abbildung 4.3).

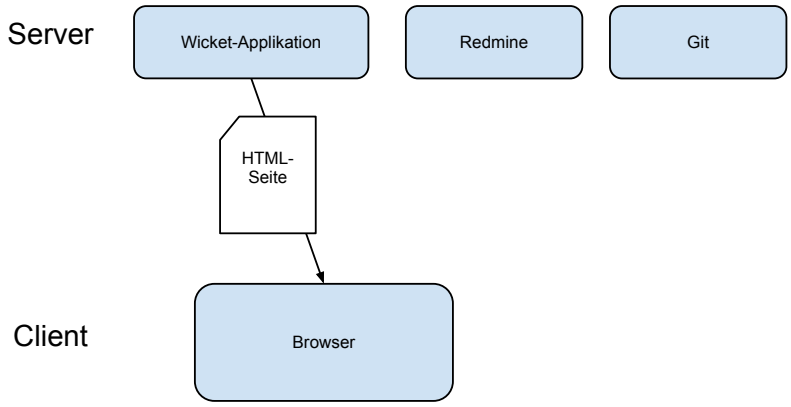


Abbildung 4.3: Ausgangslage der Nachbarsysteme

Die Benutzerschnittstelle der Wicket-Applikation wird als HTML-Seite ausgeliefert, sodass eine Lösung, die Tickets an Komponenten der Benutzerschnittstelle visualisiert, ebenfalls Programmier- und Auszeichnungssprachen verwenden muss, die von Browsern interpretiert werden. Zur Auswahl stehen hier JavaScript (JS), Hypertext Markup Language (HTML), JavaScript Object Notation (JSON) und Extensible Markup Language (XML).

Für den serverseitigen Teil der Entwicklung bietet sich der Tomcat Applikationsserver an, der Java als Programmiersprache voraussetzt. Diese Umgebung wird für die meisten Projekte im Unternehmen eingesetzt. Aus Gründen der dadurch vereinfachten Wartung und Integration empfiehlt es sich daher, diesen Weg fortzusetzen.

Um Daten aus Nachbarsystemen zu beziehen, sollen möglichst nur standardisierte Schnittstellen angesprochen werden. Dies hat den Vorteil, dass die Entwicklung nachhaltig ist,

[10] Git ist streng genommen ein verteiltes System. Jeder Client besitzt ein Codeverzeichnis (Repository) und gleicht dieses nach der Änderung mit dem Master-Repository auf dem Server ab. Das Master-Repository ist dabei für diese Diplomarbeit relevant.

da bei der Weiterentwicklung der Nachbarsysteme die offiziellen Schnittstellen meist abwärtskompatibel bleiben oder sich nur geringfügig ändern. Gleichzeitig existiert eine Dokumentation und eine Nutzergemeinde, die Informationen über bekannte Probleme und Lösungen in Internetforen austauscht.

Als mögliche *Schnittstelle zu Git* exisitiert das Projekt JGit, welches es ermöglicht, Befehle in der Programmiersprache Java an das Versionierungssystem zu richten. *Redmine* bietet eine auf REST basierende Webschnittstelle an, die Informationen zu Tickets in den Datenformaten JSON oder XML ausliefert und auch das Bearbeiten des Ticketbestands ermöglicht.

Es soll nach Möglichkeit keine eigene Datenhaltung existieren. Daten sollen in bestehenden Systemen über deren bereits vorhandenen Schnittstellen untergebracht werden. Z.B. bietet Redmine sog. „Benutzerdefinierte Felder“ an, die es anstelle einer eigenen Datenbank zu verwenden gilt.

5 Lösungsansätze

In diesem Kapitel werden zwei existierende Lösungsmöglichkeiten, sowie ein Lösungsansatz des Autors vorgestellt und bewertet. Die Facharbeit eines Mitarbeiters der intecsoft in Kapitel 5.1 hat nachgewiesen, dass es prinzipiell möglich ist, Komponenten der Benutzerschnittstelle mit Anforderungen aus der Projektmanagementsoftware „Redmine“ zu verknüpfen. Dieser Ansatz könnte weiterverfolgt und ausgebaut werden. Ebenso könnte das, in Kapitel 5.2 beschriebene, kommerzielle Werkzeug des australischen Unternehmens „BugHerd Pty, Ltd“ implementiert werden, welches zum Markieren von Fehlern auf Webseiten entwickelt wurde. Der Lösungsansatz des Autors in 5.3 bezieht Informationen aus dem Quellcodeverzeichnis der Anwendung ein, um Beziehungen zwischen der Benutzerschnittstelle und den Anforderungen herzustellen.

Nach 5.4 Evaluation wird im letzten Kapitel erläutert, warum eine der Lösungsmöglichkeiten favorisiert wurde.

5.1 Facharbeit E. Gorning

Der Fachinformatiker für Anwendungsentwicklung, E. Gorning, hat im Jahr 2011 in seiner Facharbeit einen Lösungsansatz verfolgt, der auf direkte Beziehungen zwischen den als Ticket erfassten Anforderungen und der Benutzerschnittstelle der Wicket-Applikation aufbaute. Dazu wurde im Ticket ein eindeutiges Merkmal einer Komponente der Benutzerschnittstelle eingetragen, das zur Laufzeit der Wicket-Applikation durch das Framework erzeugt wurde (Vgl. Abbildung 5.1).

Das „Wicket-Path“ genannte Merkmal bildet den Ort der Komponente im Komponentenbaum ab, der beim Generieren der HTML-Seite in der Wicket-Applikation existierte (Vgl. Abbildung 5.9). Es wird als Attribut im HTML-Element der betreffenden Komponente hinterlegt, z. B. für ein Texteingabefeld:

```
<input name="begriff" type="text"
wicketpath="suchmaske_form_begriff" />
```

In der Webseite werden dann die zu einer Komponente gehörenden Tickets angezeigt, wenn mit der Maus darüber gefahren wird. In dem Bildschirmfoto in Abbildung 5.2 ist beispielsweise die Ausgabe der Versionsnummer mit einem Ticket verknüpft.

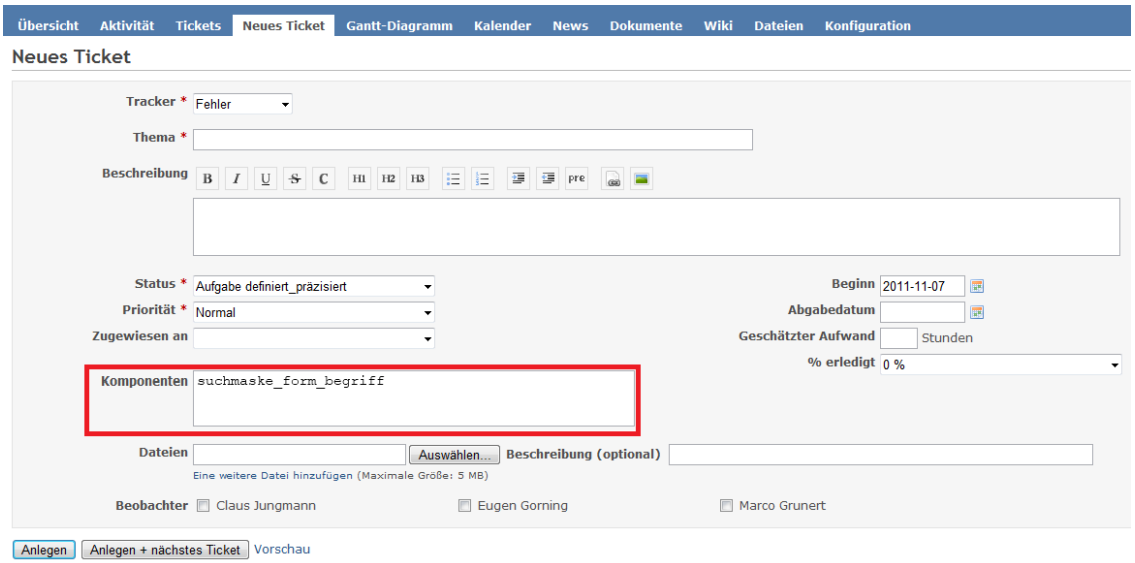


Abbildung 5.1: Erstellen einer Beziehung zwischen Benutzerschnittstelle und Redmine-Ticket in der Facharbeit von E. Gorning durch Eintragen des Wicket-Path im Ticket (rote Markierung)

Abbildung 5.2: Anzeige von Beziehungen zwischen Benutzerschnittstelle und Redmine-Ticket in der Facharbeit von E. Gorning

Die Anzeige der Tickets in der betreffenden Webseite wird durch eine Browsererweiterung [11] erreicht. Diese durchsucht die HTML-Befehle der Webseite nach Wicket-Path-Attributen und fragt für jedes gefundene Attribut einen Webservice ab, der Ticketinformationen zurückgibt, sofern eine Beziehung gefunden wurde (Vgl. Abbildung 5.3).

Die Verwendung des Wicket-Path hat den Vorteil, dass die Beziehungen zwischen Anforderungen und Benutzerschnittstelle fein granular bis auf einzelne Komponenten im Apache Wicket Framework, wie z. B. Texteingabefelder, spezifiziert werden können.

Leider ist das Merkmal nur innerhalb einer Webseite eindeutig – nicht über mehrere von der Applikation ausgelieferte Webseiten hinweg. Ein weiterer Nachteil ist die un-

[11] Es wurde ein sog. „Greasemonkey-Skript“ im Browser installiert. Es beinhaltet JavaScript Quellcode und ist mit Erweiterungen für Browser, die auch „Add-Ons“ oder „Plug-Ins“ genannt werden, vergleichbar.

günstige Nachhaltigkeit der Beziehungen: Da der Wicket-Path aus Bezeichnungen von Komponenten im Komponentenbaum gebildet wird, ändert sich dessen Wert sobald Umbenennungen auftreten oder die Struktur des Baums verändert wird, was während der Implementierungsphase eines Softwareprojekts recht häufig vorkommen kann. In solch einem Fall muss die Verlinkung erneut vorgenommen werden.

Mit dieser Facharbeit konnte nachgewiesen werden, dass es prinzipiell möglich ist, Komponenten der Benutzerschnittstelle mit Tickets zu verbinden. Zur Fortsetzung dieses Lösungsansatzes könnten Markierungsmechanismen implementiert werden, wie sie im folgendem Abschnitt bei „BugHerd“ verwendet werden.

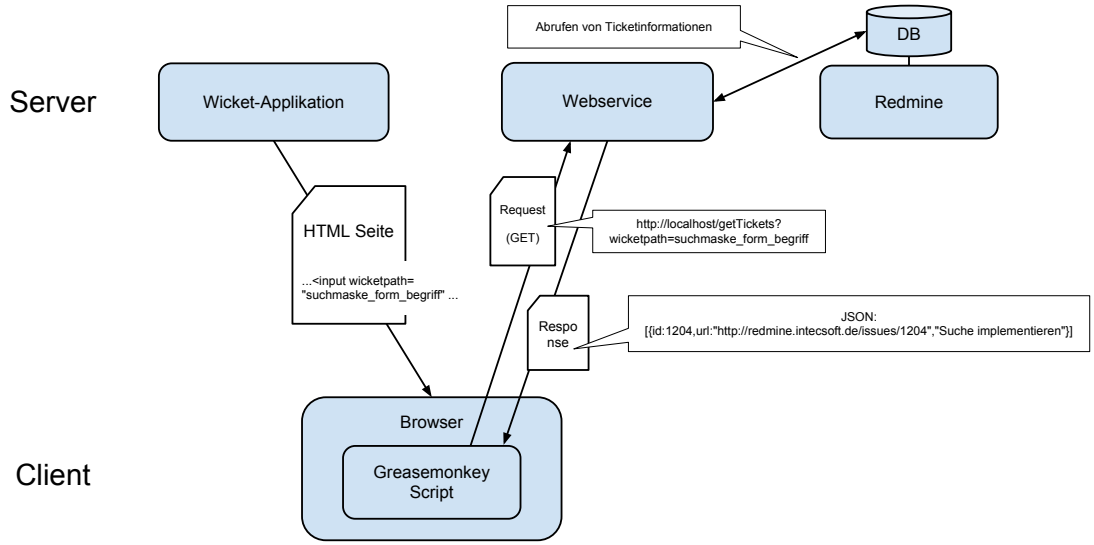


Abbildung 5.3: Übersicht des Datenflusses zwischen den Systemen des Lösungsansatz der Facharbeit von E. Gorning

5.2 BugHerd

Das Produkt „BugHerd“ des gleichnamigen, erst im Jahr 2011 in Australien gegründeten Unternehmens soll lt. Hersteller das Markieren von Fehlern auf Webseiten stark vereinfachen [Bug12a]. Die häufig angewandte Praxis, dass Fehler in E-Mails beschrieben und Bildschirmfotos mit Markierungen angefügt werden, wird durch ein in der Webseite integriertes Werkzeug zum Melden von Fehlern ersetzt.

Sobald der Betreiber die BugHerd-Lösung in seine Webseite eingebunden hat, können wie in Abbildung 5.4 ersichtlich, beliebige Orte einer Webseite markiert werden. Im Gegensatz zu ähnlichen Lösungen wie „Google Feedback“ [Goo12] oder „Bugmuncher“ [Bea12], die lediglich aus der vom Nutzer erstellten Markierung ein Bildschirmfoto anfertigen, kann BugHerd die Meldung erneut zusammen mit anderen Markierungen auf der Webseite darstellen (Vgl. Abbildung 5.5), weshalb das Produkt für diese Diplomarbeit relevant ist.

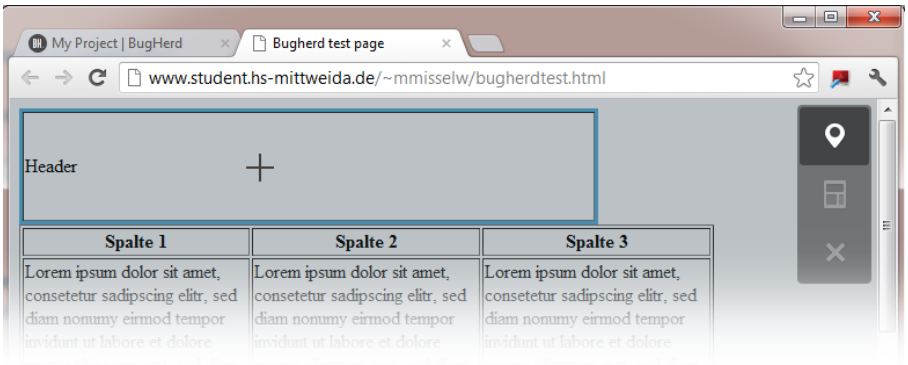


Abbildung 5.4: Melden eines Fehlers in einer Webseite mit Bughherd

Statt Fehlern könnten mit diesem Werkzeug auch Anforderungen auf der als Webseite vorhandenen Benutzerschnittstelle einer Wicket-Applikation markiert werden. BugHerd bietet eine Behandlung der Fehlermeldungen an, die der Ticketverwaltung von Redmine sehr ähnlich ist. So können Fehlermeldungen Mitarbeitern zugewiesen werden, die diese je nach angegebener Priorität bearbeiten. Zusätzlich kann BugHerd mit der REST-Schnittstelle einer Redmine-Instanz kommunizieren und ermöglicht somit eine Zwei-Wege-Synchronisation zwischen in BugHerd erfassten Fehlermeldungen und dem Ticketbestand von Redmine [Bug12b]. Die Daten der Redmine-Tickets (z. B. Titel, Status, Priorität) werden dann in eine BugHerd-Fehlermeldung überführt und umgekehrt.

Abbildung 5.5: Anzeige von Markierungen auf einer Webseite mit BugHerd

Im Gegensatz zur Facharbeit von E. Gorning sind die vorgenommenen Markierungen robuster gegenüber Veränderungen in der Webseite. Dies haben Tests des Autors ergeben.

Es konnte festgestellt werden, dass der Hersteller Anstrengungen unternommen hat, um die Nachhaltigkeit von Verlinkungen zwischen Benutzerschnittstelle und Fehlermeldung zu erhöhen. Für den Test wurde der HTML- Code einer Webseite mit Markierungen manipuliert: Es wurden die Werte von „id“-Attributen verändert, zusätzliche HTML-Elemente eingefügt, umschließender HTML-Code gelöscht oder manipuliert, sowie der Anzeigetext geändert. Vermutlich wird ein Satz mehrerer Identifikationsmerkmale verwendet, beispielweise die x,y-Koordinaten auf dem Bildschirm, XML Path Language (XPath) und ein Ausschnitt des HTML-Code, der sich bei der Markierung befindet.

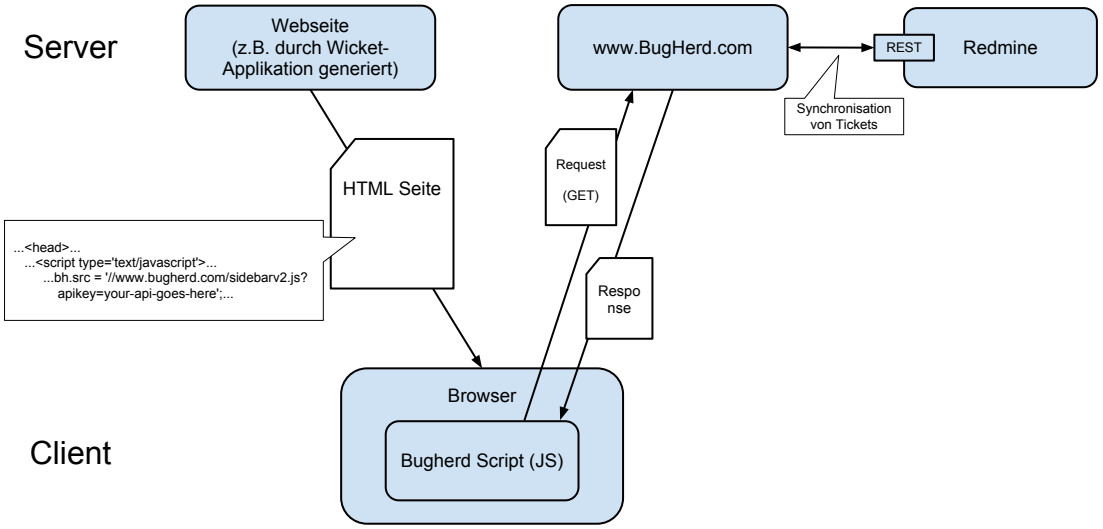


Abbildung 5.6: Übersicht des Datenflusses zwischen den Systemen der Lösung von BugHerd

Lediglich eine Registrierung auf der Webseite des Herstellers ist nötig, um die Lösung zu nutzen. BugHerd erfordert neben der Integration in die Webseite, die über einen kurzen JS-Quellcode im Kopfbereich der HTML-Seite geschieht (Vgl. Abbildung 5.6), keinen weiteren Installationsaufwand. Die Software zur Verwaltung der BugHerd-Fehlermeldungen wird nicht zur Installation auf dem eigenen Server angeboten. Dies hätte zur Folge, dass die intecsoft Daten ihrer Kunden auf fremden Servern im Ausland abgelegen müsste, um BugHerd zu nutzen.

Der ursprüngliche Anwendungsfall des Herstellers „Markieren von Fehlern auf einer Webseite“ wird in diesem Lösungsansatz verwendet, um die Soll-Kriterien dieser Diplomarbeit zu erfüllen. Die vorgestellte Lösungsmöglichkeit wurde jedoch nicht für deren Anwendungsfälle (vgl. Kapitel 4.1) entwickelt, die stark in die Prozesse der intecsoft integriert sind. BugHerd geht z. B. davon aus, dass eine Fehlermeldung immer erstellt wird, indem ein Fehler markiert wird. Prozesse, die eine spätere Markierung erfordern, werden von dieser Lösung nicht unterstützt. Für den Software-Entwicklungsprozess der intecsoft würde dies bedeuten, dass alle Tickets ausschließlich mit BugHerd erfasst und anschließend synchronisiert werden müssten. Die Markierung muss sofort bei der Erfassung erfolgen und kann später nicht angepasst werden.

5.3 Nutzung vorhandener Beziehungen zwischen Quellcode-Versionsständen und Redmine Tickets

Während der Erhebung des Ist-Zustands wurden Mitarbeiter befragt, die aufgrund Ihrer Rolle im Unternehmen potentiell für die Erstellung von Beziehungen zwischen Benutzerschnittstelle und Anforderungen in Frage kamen. Sie wiesen darauf hin, dass sie sich eine manuelle Markierung wie bei BugHerd nur schwer vorstellen können, da hiermit der Software-Entwicklungsprozess um eine weitere Aufgabe ergänzt werden müsste. Gleichzeitig würde im Prozess jedoch bereits eine Station existieren, die Anforderungen mit Quellcodeversionen verknüpft. Wie bereits in Kapitel 3.3.2 Git (Seite 17) erläutert, wird jede Änderung am Quellcode mit einem Verweis auf ein Ticket in Redmine im Versionierungssystem „Git“ eingetragen.

Da der Quellcode auch für die Generierung der Benutzerschnittstelle verantwortlich ist, besitzt er eine natürliche Verbindung zu dieser. Die durch Git erfassten Beziehungen zwischen Benutzerschnittstelle und Quellcode, sowie Quellcode und Redmine Ticket schließen hiermit die Lücke zwischen der Benutzerschnittstelle und den Anforderungen, ohne dass eine direkte Verlinkung wie in der Facharbeit von E. Gorning oder BugHerd notwendig ist (Vgl. Abbildung 5.7). Die Herausforderung dieses Lösungsansatzes bestand darin, die bestehenden Beziehung zwischen den heterogenen Artefakten zur Laufzeit der Anwendung programmatisch zu *erfassen* und zu *nutzen*.

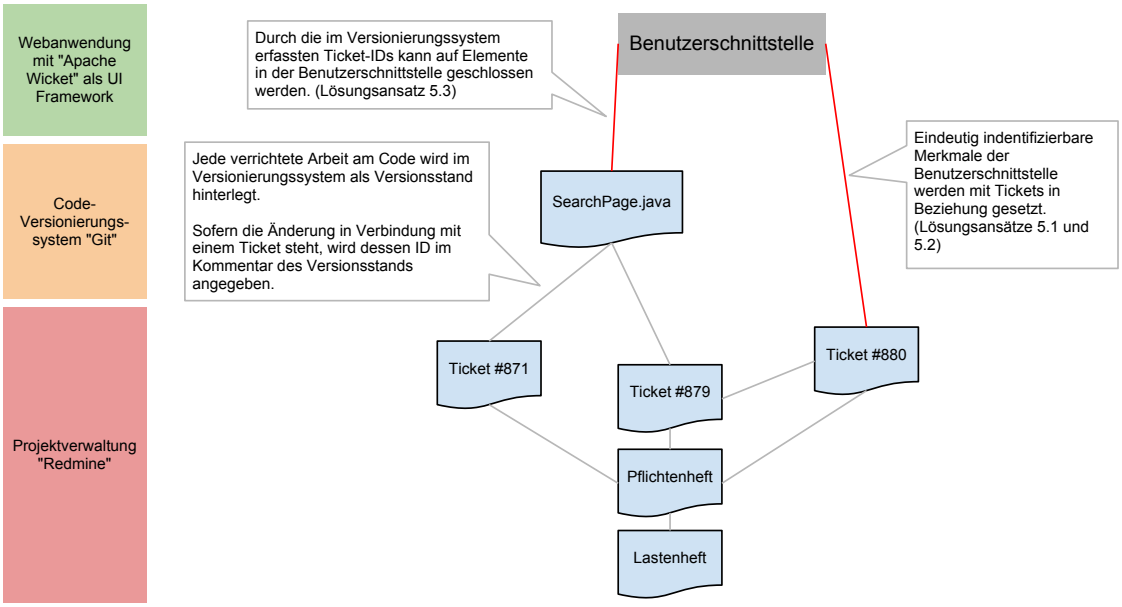


Abbildung 5.7: Vergleich der Verlinkungen der Lösungsansätze

Wie in den beiden bereits vorgestellten Lösungsansätzen musste ein eindeutiges Identifikationsmerkmal für Komponenten in der Benutzerschnittstelle der Anwendung gefunden werden. Dieses Merkmal musste maschinenauslesbare Rückschlüsse auf den Quellcode

zulassen. Es bot sich hierfür eine Entwicklung des US-amerikanischen Unternehmens „42Lines.net“ an. Das „Wicket-Source“ genannte Open-Source-Projekt wurde ursprünglich entwickelt, um das „Springen“ von Stellen in Wicket-Webseiten zurück zum Java-Quellcode zu ermöglichen [nach 42L12]. Es besteht aus einem Modul für die Wicket-Applikation, das zu jeder Komponente der Benutzerschnittstelle im HTML-Quellcode den Ort hinterlegt, wo die Instanziierung im Java-Quellcode stattfand, z. B. für ein Texteingabefeld:

```
<input name="begriff" type="text"
wicketsource="de.intecsoft.projectx.ui:SearchPage.java:28" />
```

Mithilfe von 42Lines.net angebotener Erweiterungen für Browser und Entwicklungsumgebungen kann der Betrachter einer Webseite ein Element der Benutzerschnittstelle untersuchen und anschließend zum entsprechenden Java-Quelltext springen. Diese Art der Auswertung der Wicket-Source-Attribute ist jedoch nicht für diese Diplomarbeit relevant. Wichtiger ist die Verfügbarkeit eines eindeutiges Identifikationsmerkmals für Komponenten in der Benutzerschnittstelle zur Laufzeit der Anwendung, das auf den verwendeten Quellcode referenziert.

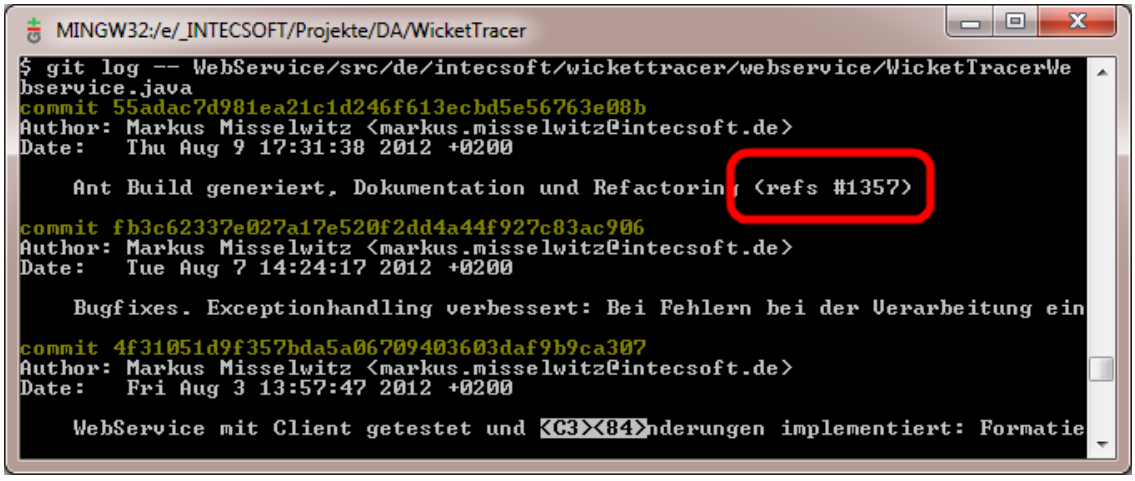


Abbildung 5.8: Log Befehl mit Pfad als Parameter an das Versionierungssystem Git

Die Wicket-Source-Attribute, die den Pfad zur Java-Klasse abbilden in der eine Komponente der Benutzerschnittstelle instanziiert wurde, lassen sich in einer Abfrage an das Versionierungssystem „Git“ als Parameter verwenden. Mit dem Befehl „log“ werden alle Versionsstände (Commits) des versionierten Verzeichnisses (Repository) in chronologischer Reihenfolge ausgegeben. Die Angabe des Pfades schränkt die Ergebnismenge auf einen Ordner oder eine Datei ein. In Abbildung 5.8 wurde eine solche Abfrage durchgeführt. Der Verweis im Commit-Kommentar auf ein Ticket in Redmine ist rot markiert.

Ein großer Vorteil dieser Lösung ist eine Nachhaltigkeit der Beziehungen, die durch die Integration in den aktuellen Software-Entwicklungsprozess bereits sichergestellt wird.

Veränderungen an der Benutzerschnittstelle, die möglicherweise zum Verlust einer Beziehung zu den Anforderungen führen, gehen immer von einer Modifikation im Quellcode aus. Da jedoch jede Quellcodeänderung erneut mit einem Verweis auf ein Redmine-Ticket im Git-Repository hinterlegt wird, werden diese Beziehungen mit jeder Änderung *gewartet*. Zusätzlich bietet der Log-Befehl den optionalen Parameter „follow“, der Umbenennungen im gegebenen Pfad mit einbezieht, sodass die Beziehungen auch dann intakt bleiben, nachdem Quellcode-Dateien umbenannt oder verschoben wurden.

Weiterhin vorteilhaft ist die Menge bereits verfügbarer Beziehungen. Der Quellcode aller bisherigen Software-Projekte im Unternehmen wurden mit Git verwaltet. Genauso wurde Redmine zur Verwaltung der Anforderungen und Apache Wicket für die Codierung der Benutzerschnittstelle eingesetzt. Es existiert daher ein umfangreicher Bestand an Beziehungen, der durch diesen Lösungsansatz *genutzt* werden kann, ohne dass zusätzlicher Aufwand zur *Erfassung* erforderlich ist.

Mit der Einbeziehung von Git als zusätzliches System in diesen Lösungsansatz sind allerdings Beschränkungen nicht völlig zu vermeiden. Während BugHerd eine beliebige Stelle auf der Benutzerschnittstelle und der Ansatz von E. Gorning Komponenten mit Wicket-Path-Attributen markieren kann, unterstützt dieser Lösungsansatz nur eine gröbere Granularität der Beziehungen auf Klassenebene. Beispielsweise kann dieser Lösungsansatz in Abbildung 5.9 nicht zwischen den Komponenten 5 und 7 unterscheiden, da beide in der gleichen Klasse instanziiert worden sind.

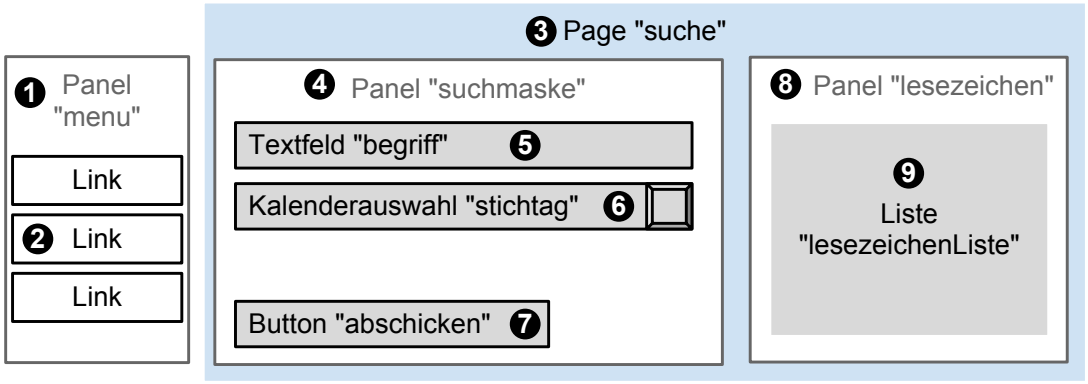
Beziehungen dieses Lösungansatz können erst *genutzt* werden, nachdem sie nach einer Änderung im Quellcode *erstellt* wurden. Allerdings kann es im Anwendungsfall A7[12] vorkommen, dass Tickets zum gleichen Fehler schon existieren, aber die Entwickler noch nicht darauf eingegangen sind und folglich noch kein Versionsstand in Git dazu existiert. Insbesondere niedrig priorisierte Fehler werden oft gesammelt und zu einem späteren Zeitpunkt umgesetzt. Der Anwendungsfall A7 wird daher von dieser Lösungsmöglichkeit nur unterstützt, wenn bereits Quellcode-Änderungen mit den betreffenden Tickets verlinkt wurden. Als Lösung könnte ein hybrider Ansatz entwickelt werden, der sowohl Verlinkungen aus dem Versionierungssystem anzeigt, als auch manuelles Markieren wie in den Kapiteln 5.1 und 5.2 zulässt (vgl. 8.2.1 im Kapitel Ausblick, Seite 58).

5.4 Evaluation

Es wurde festgestellt, dass alle Lösungsansätze Tickets der Projektmanagementsoftware Redmine in der Benutzerschnittstelle darstellen können und somit die Anforderung F1[13]

[12] A7: Tester sucht Tickets, die mit einem Fehler in Verbindung stehen können, um nicht unnötig ein neues Ticket zu öffnen. (Vgl. Kapitel 4.1 Anwendungsfälle, Seite 22)

[13] F1: Es sollen Tickets der Projektmanagementsoftware „Redmine“, die zu einer Komponente in der Benutzerschnittstelle gehören, zur Laufzeit der Anwendung in der Benutzerschnittstelle erscheinen. (Vgl. Kapitel 4.2.1 Funktionale Anforderungen, Seite 24)



Wicket-Path-Attribute	Wicket-Source-Attribute
❶ menu	❶ de.intecsoft.projectx.ui:MainPage.java:79
❷ menu__relative__path__prefix__2	❷ (nicht vorhanden)
❸ suche	❸ de.intecsoft.projectx.ui:MainPage.java:123
❹ suchmaske	❹ de.intecsoft.projectx.ui:SearchPage.java:28
❺ suchmaske_form_begriff	❺ de.intecsoft.projectx.ui:SearchPanel.java:59
❻ suchmaske_form_stichtag	❻ de.intecsoft.projectx.ui:CalendarPanel.java:45
❼ suchmaske_form_abschicken	❼ de.intecsoft.projectx.ui:SearchPanel.java:59
❽ lesezeichen	❽ de.intecsoft.projectx.ui:SearchPage.java:30
❾ lesezeichen_lesezeichenListe	❾ de.intecsoft.projectx.ui:LesezeichenPanel.java:25

Abbildung 5.9: Vergleich der Attribute von Wicket-Path und Wicket-Source anhand der schematischen Darstellung einer mit Apache Wicket generierten HTML-Seite

erfüllen. Die Unterschiede liegen hier in der Granularität der Beziehungen. Markierungen bei BugHerd sind frei platzierbar. Im Lösungsansatz der Facharbeit von E. Gorning kann eine ausreichende Anzahl von Komponenten mit dem Wicket-Path-Attribut markiert werden. Die Granularität beim Lösungsansatz des Autors ist vom Quellcode der Wicket-Applikation abhängig und umfasst dort die Ebene von Klassen. Daher bietet dieser Ansatz die gröbste Granularität.

Titel und Status eines Tickets können von allen Ansätzen sichtbar gemacht werden. Lediglich der Lösungsansatz von BugHerd bietet keinen Verweis zu Vollansicht des Tickets in Redmine an, jedoch sind die Daten der Tickets im BugHerd-System durch die 2-Wege-Synchronisation verfügbar. Anforderung F2 wird bis auf eine Einschränkung bei BugHerd von allen Lösungsmöglichkeiten erfüllt.

Anforderung F3 definiert die Art der Erfassung und Wartung von Beziehungen. F3a verlangt, dass alle relevanten Anwendungsfälle (Vgl. Kapitel 4.1, Seite 22) unterstützt werden müssen. Nach F3b muss die Lösung in bestehende Arbeitsabläufe integriert und F3c intuitiv und schnell zu bedienen sein. In der Art und Qualität der Erfüllung dieser Anforderungen unterscheiden sich die drei vorgestellten Ansätze wesentlich.

Die Erfüllung aller relevanten Anwendungsfälle (F3a) ist von mehreren Faktoren abhängig. Zunächst muss sichergestellt werden, dass die Beziehungen vollständig und korrekt

Nr.	Gorning (5.1)	BugHerd (5.2)	Autor (5.3)
F1	Ja, feine Granularität	Ja, sehr feine Granularität	Ja, grobe Granularität
F2	Ja	Ja, außer fehlender Verweis auf Vollansicht von Tickets	Ja
F3a	Ja, aber Nachhaltigkeit und Quantität erfordern hohen Aufwand		Ja, aber Anwendungsfall A7 wird nur teilweise unterstützt
F3b	Zusätzliche Aufgabe im Software-Entwicklungsprozess	Zusätzliche Aufgabe im Software-Entwicklungsprozess, Markierung nur einmalig bei Erstellung in BugHerd möglich	Keine Veränderung für Software-Entwicklungsprozess
F3c	Ja, nach Weiterentwicklung	Ja	Ja

Tabelle 5.1: Erfüllung der funktionalen Anforderungen

erfasst worden sind. Insbesondere die Quantität muss ein ausreichendes Maß erreichen, damit die Lösung für die aufgestellten Anwendungsfälle brauchbar ist. Beispielsweise nutzt es bei der Projektüberwachung (Anwendungsfall A4) wenig, wenn wesentliche Anforderungen in einer Maske fehlen, weil sie nicht mit der Benutzerschnittstelle verlinkt sind.

In den Lösungsansätzen von E. Gorning und BugHerd ist die Erfassung und Wartung von Beziehungen nicht im bereits etablierten Software-Entwicklungsprozess integriert (F3b). Erheblicher Aufwand ist daher erforderlich, um die Vollständigkeit der Beziehungen bei diesen zwei Ansätzen zu gewährleisten. Weiterhin wurde eine geringe Nachhaltigkeit der Beziehungen festgestellt, die nur durch erhöhten Wartungsaufwand gemindert werden kann. Die bereits im Software-Entwicklungsprozess eingebettete Lösung zur Erfassung und Wartung von Verlinkungen des Autors erfordert hingegen keinen zusätzlichen Aufwand. Jedoch wird der Anwendungsfall A7 nur eingeschränkt unterstützt. Hier bedarf es Entwicklungsarbeit, die über den Rahmen dieser Diplomarbeit hinausgeht (vgl. 8.2.1 im Kapitel Ausblick, Seite 58).

Da Verlinkungen nur beim erstmaligen Erstellen des Tickets in BugHerd erfasst werden können, schränkt dieser Lösungsansatz zudem die gewohnte Arbeitsweise mit Redmine stark ein.

Die Anforderung nach einer intuitiven und schnell bedienbaren Erfassung und Wartung von Beziehungen (F3c) wird vom kommerziellen Werkzeug BugHerd umgesetzt, indem

Stellen auf Webseiten durch den Mauszeiger markiert werden können. Die Lösung der Facharbeit von E. Gorning müsste um BugHerd-ähnliche Funktionen ausgebaut werden, um dieses Soll-Kriterium zu erreichen. Der Ansatz des Autors baut auf eine Erfassungsmethode auf, die für die Rolle „Entwickler“ hinreichend intuitiv und schnell bedient werden kann. Sie wird bereits erfolgreich im Unternehmen angewandt wird.

5.5 Favorisierung eines Lösungsansatzes

Der Lösungsansatz des Autors, der vorhandene Beziehungen zwischen Quellcode-Versionsständen und Redmine Tickets verwendet, unterscheidet sich in der Erfassungsmethode von den anderen Ansätzen wesentlich. Er wurde für diese Diplomarbeit favorisiert, da er die am Bestem an den Software-Entwicklungsprozess angepasste Erfassungsmethode besitzt. Weiterhin ist er in der Nachhaltigkeit von Beziehungen den anderen Lösungen überlegen. Da keine zusätzliche Aufgabe im Prozess nötig ist, entstehen keine Kosten für das Erfassen und Warten von Beziehungen.

Formal werden bei den Lösungsansätzen von E. Gorning und BugHerd ebenso alle Anwendungsfälle unterstützt, jedoch verlangen diese einen hohen Aufwand für die Erfassung von Beziehungen, was eine hohe Hemmschwelle für die Verwendung dieser Systeme im mittelständischen Unternehmen aufbaut. Die eingeschränkte Unterstützung für den Anwendungsfall A7 im Vorschlag des Autors kann durch anschließende Entwicklungen behoben werden, sodass dieser Mangel keine Relevanz bei der Entscheidung hatte.

Die systembedingte gröbere Granularität der Beziehungen des favorisierten Ansatzes auf Java-Klassen wird hingenommen, da die aufgestellten Anwendungsfälle dennoch unterstützt werden. Diese Entscheidung wird von den Ergebnissen einer Untersuchung von Eqyed et al. unterstützt. In drei Studien wurden Kosten und Nutzen von Verlinkungen zwischen Packages, Klassen und Methoden der Programmiersprache Java mit Anforderungen erhoben, wobei die Granularität auf Klassenebene den besten Kompromiss ergab. Die Autoren empfahlen zudem, zunächst eine geringere Granularität zu verwenden und in begründeten Fällen die Qualität von Beziehungsinformationen zu erhöhen [Nach EGHB09, S. 8, 10], was durch eine Weiterentwicklung dieses Ansatzes möglich wird.

6 Konzeption

Nachdem spezifiziert ist, welche Lösung die Anforderungen am Besten erfüllt, muss im nächsten Schritt die konkrete Implementierung geplant werden. Es sind mehrere Systeme zu integrieren, die über unterschiedliche Schnittstellen angesprochen werden. Hier ergeben sich unterschiedliche Möglichkeiten der Umsetzung. Die Erstellung und schrittweise Verfeinerung der Konzeption wurde wesentlich durch Erfahrungen aus prototypischen Implementierungen kritischer Stellen unterstützt.

6.1 Systemkomponenten

In Abbildung 6.1 wird der Aufbau des Systems und die Kommunikation zwischen den Systemkomponenten veranschaulicht. Mit der dritten und letzten Version des Systementwurfs wurde die Lösung „WicketTracer“ genannt. Sie besteht aus drei Teilen:

- WicketTracer Module
- WicketTracer Client
- WicketTracer Webservice

Dieser Entwurf ist gegenüber den früheren im Anhang B ersichtlichen Versionen durch eine starke Entkoppelung von der Wicket-Applikation, einem „dünnen“ Client und einem separaten Webservice charakterisiert. Die Gründe, die zu diesem Design geführt haben, werden in der folgenden Dokumentation der Systemkomponenten erläutert.

6.1.1 WicketTracer Module

Es handelt sich hierbei um eine Klasse für die Wicket-Applikation, die den Client konfiguriert und ausliefert, der später die Wicket-Source-Attribute auswertet. Es werden Methoden des Wicket-Frameworks verwendet, um die erforderlichen JS-, und CSS-Dateien in den Kopfbereich der HTML-Seite einzubinden.

Eine Interaktion mit dem *Wicket-Source Module* oder anderen Systemen wie Git oder Redmine findet auf dieser Ebene nicht statt. In der ersten Version des Entwurfs war eine Kommunikation mit diesen Systemen vorgesehen (Abbildung B.1, Seite 65). Ein JS-Client wie in diesem Szenario existierte nicht, da die Benutzerschnittstelle bereits in der Wicket-Applikation generiert werden sollte. Dieser Ansatz wurde jedoch verworfen. In der prototypischen Erprobung wurde festgestellt, dass die Seitengenerierung von Apache Wicket zu komplex ist, um die Instanziierungsorte der Wicket-Komponenten gleichzeitig zu ermitteln und auszuwerten. Da die Applikation meist nur Teile des HTML-Quellcodes austauscht, kann auf Serverseite nur erahnt werden, was dem Nutzer gerade angezeigt

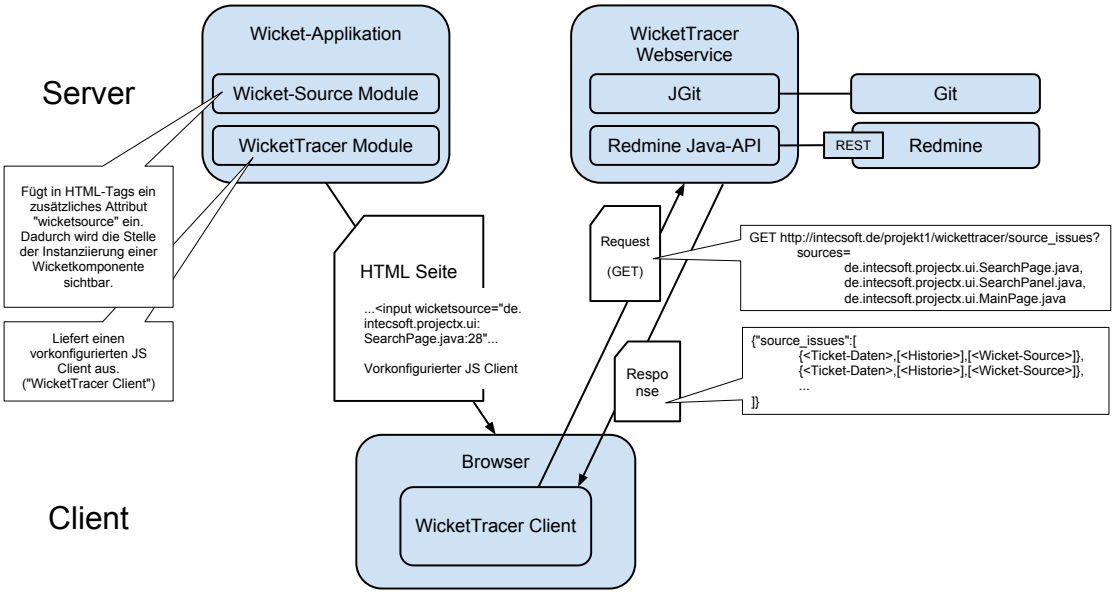


Abbildung 6.1: Komponenten und Datenfluss der Ziellösung

wird. Hier empfahl es sich, eine clientseitige Technologie vorzuziehen, um festzustellen, was tatsächlich gerade im Browser dargestellt wird. Eine stärkere Trennung der Systeme hatte darüber hinaus den Vorteil inne, dass der Prozess der Seitengenerierung nicht verlangsamt wird.

6.1.2 WicketTracer Client

Unmittelbar nach der Auslieferung der HTML-Seite beginnt der in JS codierte Client mit seiner Arbeit. Er durchsucht den HTML-Code nach Wicket-Source-Attributen und übermittelt diese in einer Anfrage an den Webservice, der alle Informationen zu den gefundenen Attributen zurückgibt. Sie werden daraufhin vom Client in der Benutzerschnittstelle dargestellt (vgl. Kapitel 6.2).

Der Client beschränkt sich nur auf das Erheben von Wicket-Source-Attributen und deren Übermittlung an den Webservice, sowie die Präsentation der Antwort. Eine direkte Kommunikation mit Git oder Redmine wie in der zweiten Version des Entwurfs (Abbildung B.2, Seite 65) findet nicht statt. Beim Generieren einer HTML-Seite sind oft mehrere Java-Klassen involviert, die durch einen gemeinsamen Versionsstand mit einem Redmine-Ticket verknüpft sind. Da Git jedoch für jede Klasse einzeln abgefragt werden muss, entsteht ein hohes Maß an Redundanz, welche bei clientseitiger Abfrage unnötig über das Netzwerk übertragen würde.

6.1.3 WicketTracer Webservice

Der *WicketTracer Webservice* bietet eine REST-Schnittstelle an. Über diese werden Wicket-Source-Attribute entgegen genommen und Informationen aus Redmine sowie Git zurückgegeben.

Per „git log"-Befehl (vgl. Kapitel 5.3) wird im Git-Verzeichnis gesucht, ob zu den gegebenen Java-Klassen Commits bestehen, die möglicherweise Verweise auf Redmine-Tickets tragen. Der in Java codierte Webservice verwendet die Open-Source-Bibliothek „JGit" [JGi12] zur Abfrage des Git-Quellcodeverzeichnisses. Anschließend werden die Tickets über die offizielle REST-Schnittstelle des Redmine-System abgerufen [Red12b]. Die Informationen aus den beiden Systemen werden aggregiert und für den Client aufbereitet. Als Datenformat wird für die Rückgabe JSON verwendet.

6.2 Benutzerschnittstelle

Es musste eine informative, auf die Soll-Kriterien fokussierte Benutzerschnittstelle geschaffen werden. Das Ziel war ein unterstützendes Werkzeug, welches sich bis zum Eintreten eines Anwendungsfalls im Hintergrund hält.

Mit einem Vektorgrafikeditor wurden mehrere Varianten veranschaulicht, die zur finalen Version geführt haben. Die Entwürfe (vgl. Anhang C) ermöglichten es, verschiedene Szenarien gedanklich durch zu spielen. Beispielsweise war zunächst vorgesehen, dass mehrere Optionen angeboten werden, um die Anzeige einzuschränken, indem nur Tickets, Commits oder Java-Klassen angezeigt werden. Durch deren Visualisierung konnte erkannt werden, dass für die Anwendungsfälle an oberster Stelle die Anzeige der Anforderungen relevant ist. Die zugeordneten Commits und Kommentare bildeten lediglich eine Historie, die den aktuellen Stand und die Entwicklung beschreiben. Inhalte der Wicket-Source-Attribute, die Java-Klassen referenzieren, wurden im finalen Entwurf nicht mehr angezeigt. Sie wurden nur noch genutzt, um Stellen auf der Benutzeroberfläche zu identifizieren, die farblich hervorgehoben werden.

Als Ergebnis ist eine interaktive Liste von Anforderungen entstanden, die alle verfügbaren Informationen auf geeignete Weise präsentiert (vgl. Abbildung 6.2). Sie wird in einem Dialogfeld über die von der Wicket-Applikation generierte Webseite gelegt. Die normale Interaktion mit der Haupt-Anwendung soll hierbei nicht behindert werden. Damit die Anmutung eines unscheinbaren, jedoch nützlichen, Werkzeugs gewahrt wird, ist das Dialogfeld im Ausgangszustand geschlossen und kann über einen immer präsenten Button geöffnet werden. Zusätzlich lässt sich die Größe und Position des Feldes per Mauszeiger verändern.

Aus den Entwürfen konnte zudem gelernt werden, dass die Datensätze am Besten in

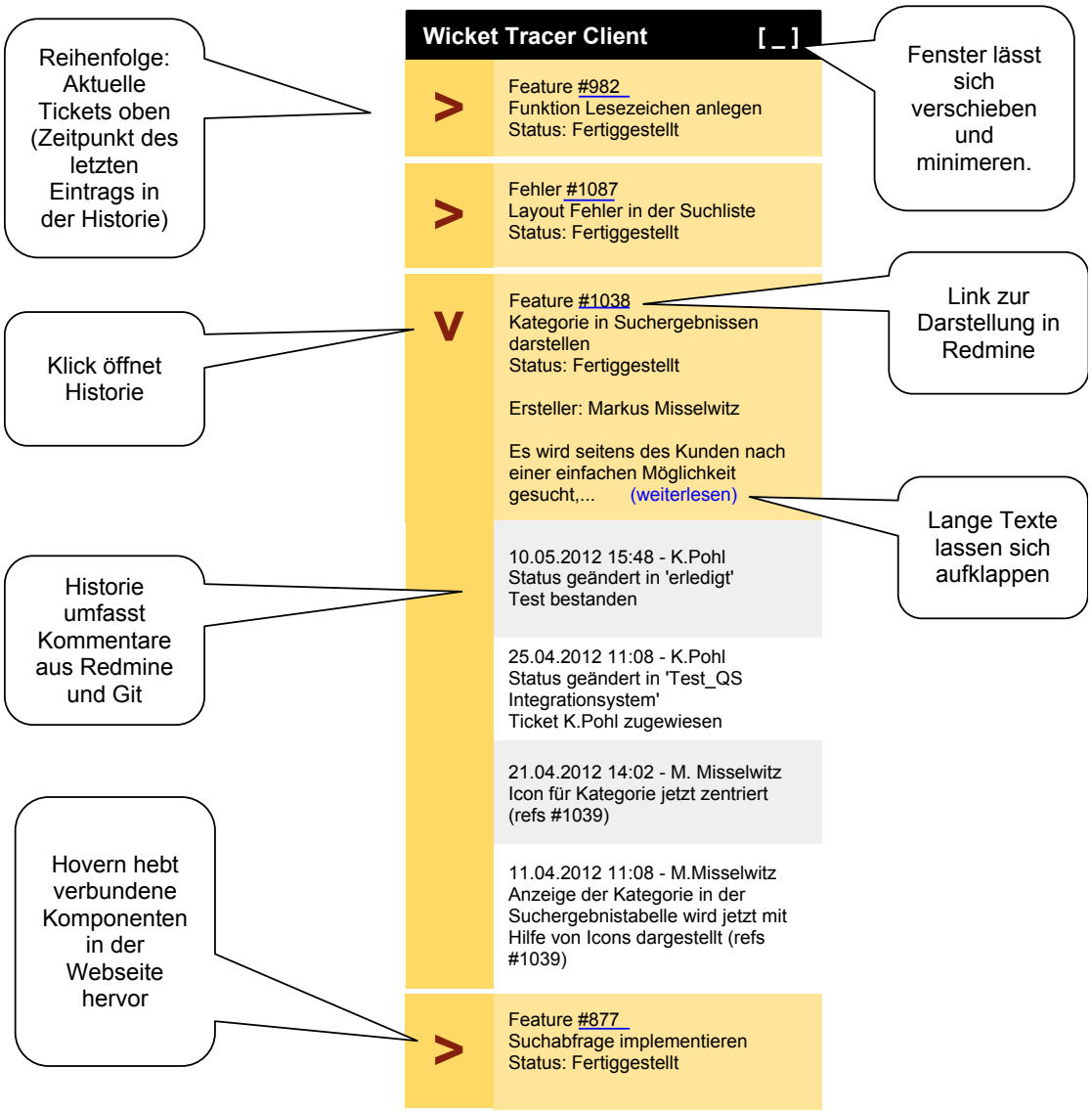


Abbildung 6.2: Finaler Entwurf der Benutzerschnittstelle des WicketTracer Client

zeitlicher Reihenfolge angeordnet werden, da die Anwender mehrheitlich die Tickets bereits kennen und sich lediglich über die aktuellen Geschehnisse informieren wollen.

6.3 Programmierschnittstellen

Wie die Autoren der arc42-Vorlage für Software- und Systemarchitektur korrekterweise feststellen, gehört die „Festlegung der Schnittstellen zu Nachbarsystemen (..) zu den kritischsten Aspekten eines Projektes.“ [HS12] Folglich war dies ein weiterer wichtiger Punkt in der konzeptionellen Arbeit. Um Risiken während der anschließenden Realisierung zu vermeiden, wurde für jede Schnittstelle zunächst prototypisch die Machbarkeit für den geplanten Einsatz überprüft. Erst dann wurde sie in die finale Konzeption aufgenommen.

6.3.1 WicketTracer Client - WicketTracer Webservice

Zur Ausgestaltung der Kommunikation zwischen Client und Webservice wurde REpresentational State Transfer (REST) ausgewählt, was sich in den letzten Jahren zu einem bedeutenden Standard für die Client-Server-Kommunikation im Internet etabliert hat [WP11, S. 41]. Ein weiterer Grund für diese Wahl waren – neben der guten Eignung für Webanwendungen – ein geringer Overhead bei der Übertragung und die gute Integration in Java durch die Programmierschnittstelle (API) „JAX-RS“, sowie die gereifte Referenzimplementierung „Jersey“ [jav12].

Als Datenformat wurde JSON festgelegt, welches gleichzeitig die Objekt-Notation von JS darstellt – der Programmiersprache des Clients. Die Weiterverarbeitung wird dadurch stark vereinfacht, da kein zusätzlicher Quellcode für die Interpretation der Daten geschrieben werden muss.

Der Webservice bietet den Endpunkt „`/source_issues`“ an. „Issues“ ist die interne Bezeichnung für „Tickets“ in Redmine, welche mit Informationen aus dem Quellcode (englisch „Source“) durch den WicketTracer Webservice kombiniert wurden.

6.3.2 WicketTracer Webservice - Git

Die Schnittstelle zu Git bildet die Open Source Software „JGit“ [JGi12]. Es handelt sich um eine Java-Bibliothek, die das Versionsverwaltungssystem Git implementiert. Ein durch Git versioniertes Quellcodeverzeichnis kann somit programmatisch mit einer Java Applikation ausgelesen und verändert werden. Die Software erledigt dafür die nötigen Dateizugriffe auf dieses Verzeichnis. Beispielsweise werden Differenzen von Quellcodeänderungen als Datei abgelegt und Einträge in Kontrolldateien vorgenommen.

Eine prototypische Implementierung hatte ergeben, dass JGit nur einen Teil der Befehle beherrscht, die von dem sonst üblichen in C geschriebenen Kommandozeilenprogramm angeboten werden. Beispielweise wurde die in Kapitel 5.3 diskutierte Option „forward“ für den „Log“-Befehl nicht unterstützt. Lediglich das Basiskommando mit dem Pfad als Parameter (vgl. Abbildung 5.8) wurde angeboten. Es bestand jedoch die Möglichkeit, die „forward“-Option zum Erkennen von Umbenennungen durch eine Eigenimplementation bereit zustellen.

6.3.3 WicketTracer Webservice - Redmine

Während in der Facharbeit von E. Gorning direkt auf die von Redmine verwendete Datenbank zugegriffen wurde (vgl. Abbildung 5.3), spricht der *WicketTracer Webservice* die REST-Schnittstelle der Projektmanagementsoftware an [Red12b]. Die offiziell

angebotene Schnittstelle hat den Vorteil, dass sie dokumentiert ist und eine gewisse Kontinuität bei der Weiterentwicklung von Redmine verspricht.

Darüberhinaus existiert mit „Redmine Java API“ ein Open-Source-Projekt, welches die Redmine-REST-Schnittstelle in Java verfügbar macht [TA12]. Leider hat die prototypische Implementierung dieser Bibliothek ergeben, dass sie eine höhere Version von Redmine voraussetzt als die im Unternehmen installierte. Für den kurzfristigen Rahmen der Diplomarbeit konnte keine Aktualisierung erfolgen. Daher wurde festgelegt, dass die REST-Schnittstelle zunächst direkt angesprochen wird, aber Vorkehrungen getroffen werden, um die nachträgliche Implementierung der „Redmine Java API“ zu erleichtern.

7 Realisierung

7.1 WicketTracer Module

Gemäß der Nicht-Funktionalen Anforderung NF3 muss die Anwendung von Fachpersonal mit wenig Aufwand anhand einer Dokumentation installiert werden können (vgl. Kapitel 4.2.2, Seite 25). Für das Wicket-Module wird diese Anforderung erfüllt, indem lediglich der Aufruf einer Klassenmethode nötig ist. Der Aufruf der „`configure`“ genannten Methode findet, ähnlich dem des Wicket-Source-Module, in der Klasse „WicketApplication“ der Webanwendung statt (Abbildung 7.1). Die von „WebApplication“ erbende Klasse wird beim Start der Anwendung ausgeführt. Hier können Befehle zur Konfiguration untergebracht werden. In diesem Fall wird die eigene Objektinstanz und die URL zum *WicketTracer Webservice* übergeben.

```
1  /**
2   * Application object for your web application.
3   */
4  public class WicketApplication extends WebApplication {
5
6      @Override
7      public void init() {
8          super.init();
9          //... weiterer Code zur Konfiguration der Webanwendung ...
10
11         // Wicket-Source
12         WicketSource.configure(this);
13         // WicketTracer
14         WicketTracer.configure(this, "http://intecsoft.de/projekt1/wickettracer");
15     }
16 }
```

Abbildung 7.1: WicketTracer-Klasse wird in der Wicket-Applikation aufgerufen

In Abbildung 7.2 sind die wesentlichen Teile der „WicketTracer“-Klasse ersichtlich. Entsprechend dem vom Framework vorgesehenen Weg wird eine Klasseninstanz registriert, die beim Generieren des Kopfbereichs des HTML-Dokuments aufgerufen wird. Dabei wird zur Liste der „zum Kopfbereich beitragenden Objekte“ ein weiteres Objekt hinzugefügt, dessen Klasse die Schnittstelle „`IHeaderContributor`“ implementiert. Laut dieser Schnittstelle muss die Methode „`renderHead`“ vorhanden sein. Hier wurde die Logik zum Einbinden der CSS- und JS-Klassen, sowie des JS-Quellcodes zur Konfiguration des JS-Clients untergebracht.

```
/**
 * Klasse zum Konfigurieren und Ausliefern des WicketTracer Clients
 *
 * @author Markus Misselwitz
 */
public class WicketTracer {

    public static void configure(   WicketApplication application,
                                    final String wicketTracerWebserviceUrl  ) {

        //"HeaderContributor" zur Applikation hinzuf"ugen
        application.getHeaderContributorListenerCollection().add(new IHeaderContributor() {

            @Override
            public void renderHead(IHeaderResponse iHeaderResponse) {
                //CSS-Datei im Header einbinden
                iHeaderResponse.renderCSSReference(new PackageResourceReference(
                    WicketTracer.class, "css/wicket-tracer.css"));
                //JS-Datei im Header einbinden, z. B. die JS-Klasse des Clients
                iHeaderResponse.renderJavaScriptReference(new PackageResourceReference(
                    WicketTracer.class, "js/wicket-tracer-client.js"));
                //JS im Header einbinden, z. B. zur Konfiguration des Clients
                iHeaderResponse.renderJavaScript(
                    "WiTr.Config = {\"webservice_url\":\"" + wicketTracerWebserviceUrl + "\"},");
            }
        });
    }
}
```

Abbildung 7.2: Quellcode der Klasse „WicketTracer“

7.2 WicketTracer Client

7.2.1 Benutzerschnittstelle

Das Bildschirmfoto in Abbildung 7.3 zeigt die Benutzerschnittstelle des *WicketTracer Clients*, der die Tickets eines realen Projekts[14] in einer Wicket-Applikation darstellt.

Die Titelleiste des Dialogfelds vom *WicketTracer Client* signalisiert in diesem Beispiel, dass 22 Tickets zu der Eingabemaske „Suche“ gefunden wurden. In diesem Bildausschnitt wird ein Teil dieser Liste angezeigt. Der Status der Tickets ist sowohl durch Text, als auch durch die Farbe des Rahmens erkennbar.

Wie konzipiert, werden die Einträge nach dem Datum des letzten Ereignisses (Ticket- oder Commit-Kommentar) geordnet. Um einen schnellen Überblick über die aktuellen Ereignisse zu ermöglichen, wird darüber hinaus immer der aktuellste Eintrag der Historie dargestellt. Der restliche Teil der Auflistung von Ereignissen zu einem Ticket ist im Ausgangszustand ausgeblendet und kann bei Bedarf angezeigt werden, wie beim „Feature #949“ im Bildschirmfoto.

Da der Mauszeiger gerade auf dem Eintrag für das „Feature #949“ liegt, werden die

[14] Art und Umfang der Tickets und der dargestellten Anwendung sind einem realen Projekt entnommen. Der Inhalt wurde für das Bildschirmfoto verfremdet, sodass keine konkreten Bezüge zu einem Projekt bestehen.

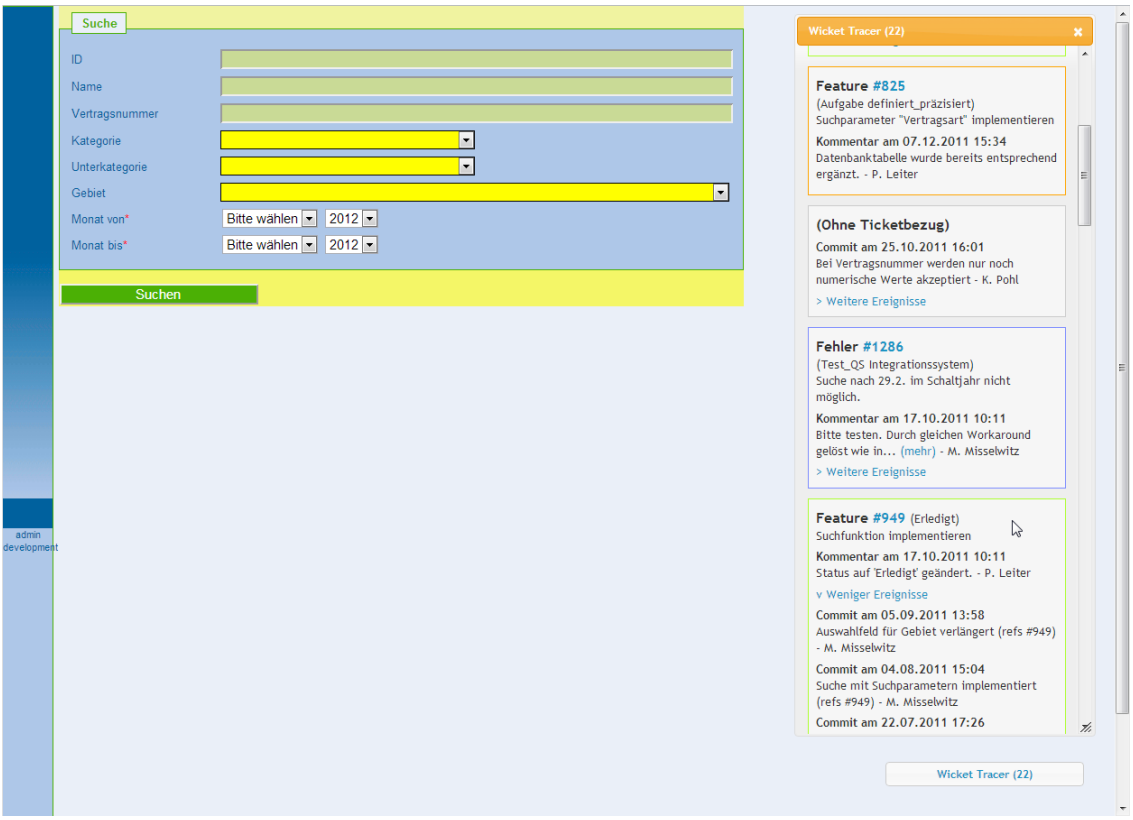


Abbildung 7.3: Bildschirmfoto des WicketTracer Clients

zugehörigen Komponenten in der Benutzerschnittstelle der Webanwendung hervorgehoben. Dies geschieht durch die vorübergehende Änderung der Hintergrundfarbe zu einem halbtransparenten Gelb.

Commits, die mit der angezeigten Seite in Verbindung stehen, jedoch keinen Bezug zu einem Ticket tragen, werden in einem Ticket mit dem Titel „(Ohne Ticketbezug)“ zusammengefasst. Im korrekten Sinne werden daher 21 Redmine-Tickets und ein „Pseudo-Ticket“ angezeigt. Diese Funktionalität war ursprünglich nicht in der Konzeption vorgesehen, wurde aber wegen der relativ einfachen Implementierung und des hohen Nutzens umgesetzt. Beim Testen in einer realen Anwendung wurde festgestellt, dass pro Maske der Benutzerschnittstelle in ca. 10 Prozent der Commits kein Bezug zu einem Ticket erfasst wurde. Dafür existierten mehrere Ursachen. In manchen Fällen wurde die Erfassung schlicht vergessen. Einige Quellcode-Änderungen waren geringfügig und wurden daher nicht erfasst. In anderen Fällen existierte kein Ticket für die Aktion, z. B. für das „Aufräumen“ (Refactoring) von Quellcode. Mit dieser Funktion werden somit alle tatsächlich vorhandenen Vorgänge auf dieser Seite der Benutzerschnittstelle angezeigt und gehen nicht verloren, wenn in den Commit-Kommentaren keine Ticket-IDs angegeben wurden.

Im Ausgangszustand wird lediglich die unten rechts im Bild vorhandene Schaltfläche angezeigt. Sie beinhaltet wie die Titelleiste Informationen zu dem Status des *WicketTracer*

Clients, etwa die Anzahl der gefundenen Tickets oder den Status des Ladevorgangs. Die hauptsächliche Benutzeroberfläche des *WicketTracer Clients* wird erst durch das Betätigen dieses Buttons durch den Nutzer geöffnet, wenn ein Anwendungsfall eingetreten ist.

7.2.2 Codierung

Die Programmierlogik wurde im Sinne der Objektorientierung entsprechend ihrer Bedeutung in Klassen unterteilt (Abbildung 7.4).

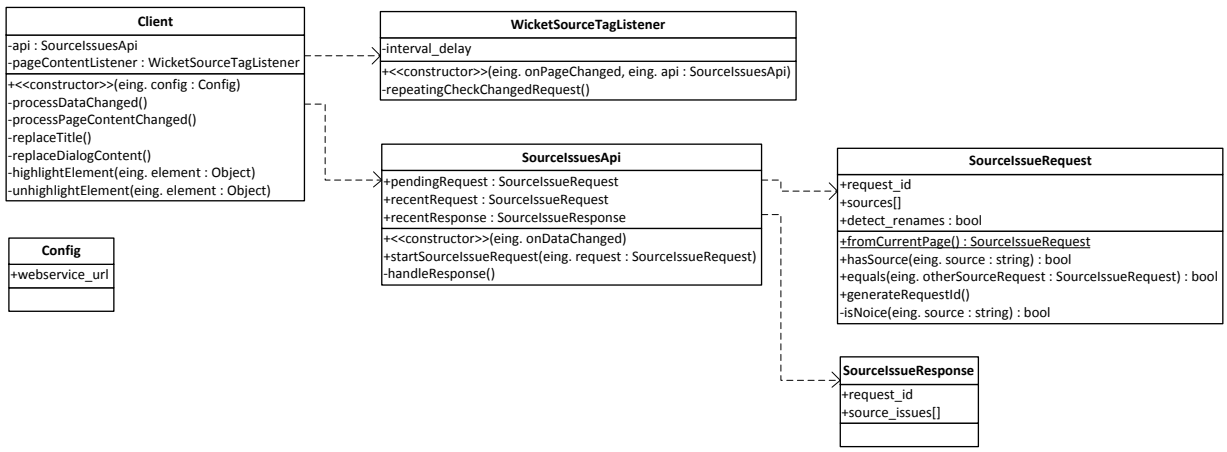


Abbildung 7.4: UML-Klassendiagramm des WicketTracer Client

„`Client`“ umfasst die Hauptklasse, die zunächst die Konfiguration vom WicketSource-Module entgegen nimmt und weitere Klassen instanziiert. *„`WicketSourceTagListener`“* ist Teil von „`Client`“ und bildet die aktive Komponente, die entscheidet, wann eine Anfrage an den *WicketTracer Webservice* gestellt werden soll. Sie erkennt durch regelmäßiges Überprüfen des HTML-Quellcodes, ob neue Wicket-Source-Attribute vorhanden sind. Tritt dieser Fall ein, wird die hinterlegte Rückrufmethode (engl. callback method) „`onPageChanged`“ der Klasse „`Client`“ aufgerufen. Hier befindet sich der Befehl zum Starten einer Anfrage an den Webservice.

Die Klasse *„`SourceIssuesApi`“* stellt die REST-Schnittstelle des *WicketTracer Webservice* als Programmierschnittstelle (API) im JS-Code bereit. Während auf die Antwort des Webservice gewartet wird, kann mit der Webseite der Wicket-Applikation weiterhin gearbeitet werden. Die Kommunikation mit dem Webservice findet asynchron statt. Aus diesem Grund wird nach dem Empfang der Antwort die Rückrufmethode „`onDataChanged`“ der Klasse „`Client`“ aufgerufen. Sollte „`startSourceIssueRequest`“ aufgerufen werden, während noch eine Antwort auf eine vorhergehende Anfrage aussteht, wird die früher gestartete Anfrage verworfen. In diesem Fall muss sichergestellt werden, dass

nur die Antwort verarbeitet wird, auf die gerade gewartet wird. Aus diesem Grund wird für jede Anfrage eine ID generiert, bestehend aus dem aktuellen Zeitstempel und einer Zufallszahl. Diese wird dann mit der „`request_id`“ der Antwort verglichen.

Nachdem der Client informiert wurde, dass neue Daten eingetroffen sind, wird durch dessen Methoden „`replaceTitle`“ und „`replaceDialogContent`“ die Antwort des Servers verarbeitet und der Inhalt der Benutzerschnittstelle entsprechend angepasst. Die Instanz der Klasse „`SourceIssuesApi`“ hält dafür die Antwort des Webservice im Attribut „`recentResponse`“ bereit. Weiterhin befindet sich in der Klasse Client die Logik zum farblichen Hervorheben von Komponenten in den Methoden „`highlightElement`“ und „`unhighlightElement`“.

Für die konkrete Implementierung wurden Funktionen der JS-Bibliothek jQuery verwendet [jQu12b]. Diese Softwarebibliothek findet in regulären Projekten der intecsoft breite Anwendung und hilft beispielsweise beim Selektieren von HTML-Elementen oder bei der Kommunikation mit dem Webservice. Effekte wie das Ein- und Ausblenden von Text können ebenfalls mit wenigen Zeilen Code implementiert werden.

Für die Gestaltung und Funktionen des Dialogfelds (Größe ändern, Verschieben, Öffnen, Schließen) wurde auf „jQuery UI“ zurückgegriffen [jQu12a]. Diese JS-Bibliothek verwendet als Grundlage jQuery und bietet u. a. konkrete Komponenten für die Benutzerschnittstelle an.

7.3 WicketTracer Webservice

Die Referenzimplementation „Jersey“ [jav12], der durch „JAX-RS“ spezifizierten Programmierschnittstelle für „RESTful Webservices“ in Java, bietet eine komfortable Möglichkeit einen REST-Webservice mit wenigen Zeilen Code bereitzustellen. Eine zentrale Rolle spielen dabei „Annotationen“. Sie stellen ein Sprachelement der Programmiersprache Java dar, um Metadaten im Quellcode unterzubringen. Erkennbar sind sie durch das @-Zeichen am Beginn der Befehle.

In Abbildung 7.5 ist ersichtlich, wie in einer gewöhnlichen Java-Klasse mithilfe von Annotationen zusätzliche Informationen untergebracht werden. In diesem Beispiel dienen sie dazu, der JAX-RS-Implementierung Details der REST-Schnittstelle bekannt zu machen. Somit wird hier ein Webservice bereitgestellt, der auf dem Endpunkt „`/source_issues`“ Anfragen per „`HTTP GET`“ mit den Parametern „`request_id`“ und „`sources`“ beantwortet, sofern der Client JSON als Datenformat für die Rückgabe erwartet.

Beim Aufruf wird folglich die Methode „getResponse“ ausgeführt. Mithilfe der entgegen genommenen Parameter wird die Klasse „`SourceIssueRequest`“ in Zeile 12 instanziiert. Sie wird in Zeile 13 dem Konstruktor der Klasse „`SourceIssueResponse`“ über-

```
1  @Path("/source_issues") //endpoint
2  public class WicketTracerWebservice {
3
4      /**
5       * Returns SourceIssues by given request
6       */
7      @GET //listenes for GET
8      @Produces(MediaType.APPLICATION_JSON) //output format
9      public SourceIssueResponse getResponse( @QueryParam("request_id") String request_id,
10                                             @QueryParam("sources") String sources ) {
11
12         SourceIssueRequest request = new SourceIssueRequest(request_id, sources);
13         SourceIssueResponse response = new SourceIssueResponse(request);
14         return response;
15     }
16 }
```

Abbildung 7.5: Quellcode der Haupt-Klasse des Webservice (vereinfacht)

geben. Abbildung 7.6 veranschaulicht in einem UML-Aktivitätsdiagramm die Vorgänge, die während der Instanziierung dieser Klasse ablaufen. Demnach wird zunächst für jeden Pfad aus einem Wicket-Source-Attribut im Git-Verzeichnis nach Commits (Versionsständen) gesucht. Diese beinhalten in ihren Kommentaren Verweise zu Tickets in Redmine. Für die weitere Verarbeitung müssen, wie im Beispiel ersichtlich, doppelt vorhandene Commits zusammengeführt und in der Hierarchie den Tickets untergeordnet werden (Vorgang „Nach Tickets aggregieren“). Im letzten Schritt werden die Ticketdaten vom Redmine System abgerufen.

In Abbildung 7.7 können die Vorgänge „Commits für jeden Pfad ermitteln“ und „Ticketdaten ermitteln“ aus der Log-Ausgabe während eines Aufrufs abgelesen werden. Um Zeit zu sparen wird der Code innerhalb der Vorgänge parallelisiert ausgeführt. Dies kann im Bildschirmfoto daran erkannt werden, dass verschiedene Kommandos unterschiedlich schnell fertiggestellt werden, was der Unbestimmtheit nebenläufiger Prozesse geschuldet ist.

In Zeile 14 (Abbildung 7.5) wird das Objekt „`response`“ zurückgegeben. Die Annotation „`@Produces(MediaType.APPLICATION_JSON)`“ in Zeile 8 signalisiert, dass der Inhalt des Objekts in das JSON-Format umgewandelt werden soll. Jersey unterstützt standardmäßig bereits übliche Datenaustauschformate wie JSON oder XML, um Java-Objekte in solche Formate zu serialisieren. Über weitere Annotationen in den Objekten kann die Serialisierung konfiguriert werden. Beispielsweise kann festgelegt werden, welche Attribute oder Methoden mit einbezogen werden sollen.

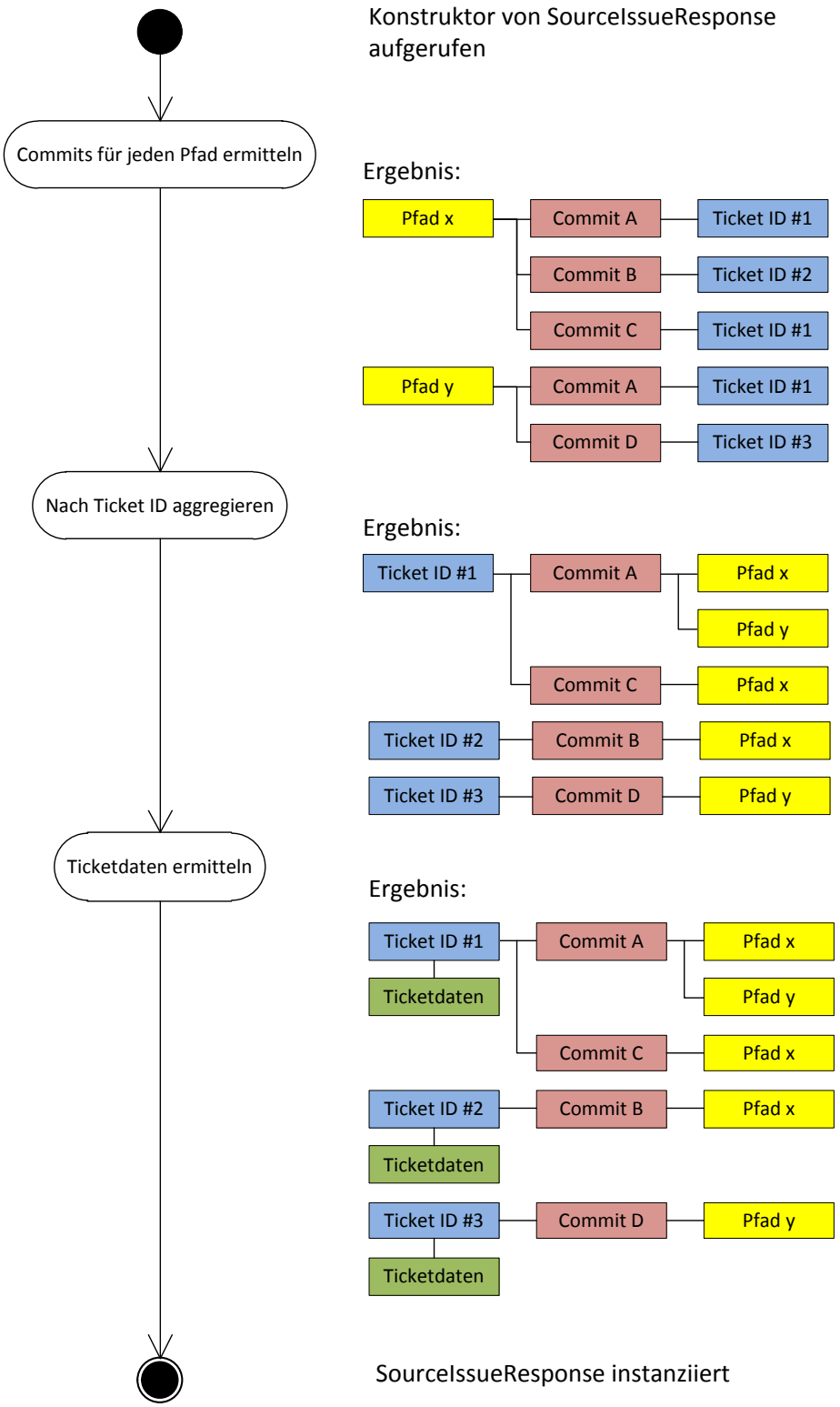


Abbildung 7.6: Vorgänge im Konstruktor der Klasse SourceIssueResponse

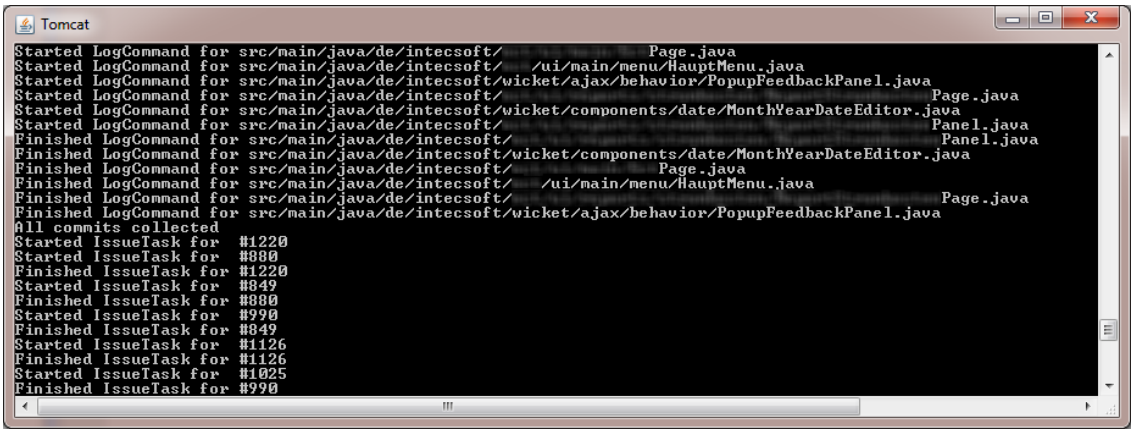


Abbildung 7.7: Bildschirmfoto der Log-Ausgabe des WicketTracer Webservice (Bezüge zu konkretem Projekt entfernt)

8 Fazit

8.1 Zusammenfassung und Bewertung

Ziel der Arbeit war die Entwicklung eines Lösungsansatzes zum Verknüpfen von Anforderungen mit Komponenten der Benutzerschnittstelle. Es sollte eine Lösung zum Rückverfolgen von Anforderungen entstehen, die im Software-Entwicklungsprozess des Unternehmens integriert ist.

Schon frühzeitig wurde erkannt, dass die Einbindung in bestehende Prozesse eine zentrale Herausforderung bildete. Bei der Pflege der vielen hundert Rückverfolgbarkeitsbeziehungen muss äußerst sparsam mit der Ressource „Arbeitszeit“ umgegangen werden. Keinesfalls sollte der Eindruck entstehen, dass die Rückverfolgbarkeitslösung nur zum Selbstzweck existiert. Auf dieses Schlüsselkriterium wurde intensiv eingegangen. Besonders deutlich kann dies an der Wahl der Technologien zur Umsetzung von *WicketTracer* abgelesen werden. Mit dem Versionsverwaltungssystem Git wurde ein zusätzliches System in die Lösung mit einbezogen und nicht eine direkte Verlinkung von Anforderungen mit der Benutzerschnittstelle verfolgt. Der vermeintliche Umweg liegt jedoch viel näher am eigentlichen Entwicklungsgeschehen in Softwareprojekten. Da jede Änderung am Quellcode inklusive eines Verweises zur Anforderung im Versionierungssystem erfasst wird, geschieht die Pflege von Beziehungen genau im Moment der Änderung und bedarf keines zusätzlichen Arbeitsschrittes, wie es bei den diskutierten alternativen Lösungswegen der Fall gewesen wäre.

Es wurde eine Lösung implementiert, um Beziehungen zwischen den heterogenen Artefakten *Quellcode-Versionsstände* (Commits), *Anforderungen* (Tickets) und *Komponenten der Benutzerschnittstelle* zur Laufzeit einer Webapplikation (des Frameworks *Apache Wicket*) zu visualisieren. Damit konnte eine alternative Sichtweise auf die Anforderungen geschaffen werden, welche vorher in Listen ohne Bezug auf die Benutzerschnittstelle organisiert waren.

Der Implementierung ging eine intensive Recherche des Ist-Zustand beim Praxispartner intecsoft voraus. Dabei wurde der aktuelle Software-Entwicklungsprozess und die darin verankerten Rollen ermittelt. Basierend auf diesen Informationen konnten auf die Prozesse angepasste Soll-Kriterien formuliert werden. Anschließend wurden verschiedene Lösungsmöglichkeiten evaluiert – darunter auch jene, die nicht den „Umweg“ über die Quellcode-Versionierung gingen. Die Konzeptionsphase war davon geprägt, dass für kritische Elemente wie etwa Schnittstellen zunächst prototypisch die Machbarkeit überprüft wurde.

Diese Diplomarbeit bestätigt nicht nur, dass es möglich ist, Beziehungen zwischen An-

forderungen und Benutzerschnittstelle herzustellen und auszuwerten. Es wurde zudem eine praktikable Lösung geschaffen, diese Art der Rückverfolgbarkeit im Projektalltag zu nutzen. Hiermit beschäftigt sich die Arbeit mit den Herausforderungen der Rückverfolgbarkeit von Anforderungen, die in der Literatur derzeit diskutiert werden. Für das allgegenwärtige Problem der Kosten einer Rückverfolgbarkeitslösung wurden Lösungsmöglichkeiten herausgearbeitet und im System *WicketTracer* implementiert. Dazu gehört eine starke Integration in bestehende Prozesse, die keinen neuen Arbeitsschritt erfordert und dafür Kompromisse in der Granularität der Beziehungen eingeht – nicht jedoch in deren Nachhaltigkeit.

Um die Ergebnisse der Arbeit anhand objektiver Kriterien einzuordnen, werden sie in Tabelle 8.1 mit hypothetischen Annahmen zur Rückverfolgbarkeit im Jahr 2035 verglichen. Diese Liste wurde von renommierten Forschern des Fachgebiets (Gotel, Cleland-Huang, Zisman, Egyed und weitere) in einem Aufsatz unter dem Titel „The Grand Challenge of Traceability (v1.0)“ veröffentlicht (vgl. Kapitel 2.3.4, Seite 11). Die Beurteilung hier bezieht sich nur auf die Rückverfolgbarkeit *von Anforderungen zu Komponenten der Benutzerschnittstelle* (Thema dieser Arbeit), während diese Einschränkung auf bestimmte Artefakttypen im Jahr 2035 nach den Autoren des Aufsatzes keine Rolle mehr spielen soll.

Aus der Bewertung geht hervor, dass die Kriterien nach einer bedarfsgerechten, kosteneffizienten und omnipräsenten Lösung zur Rückverfolgbarkeit in besonderem Maße innerhalb der definierten Randbedingungen erfüllt wurden. Für die Zuverlässigkeit und Wertschätzung der geschaffenen Rückverfolgbarkeit wurden Anstrengungen unternommen, jedoch liegen für eine umfassende Bewertung keine ausreichenden Daten aus dem Projektalltag vor. Für Kriterien, die eine allgemeine Flexibilität der Rückverfolgbarkeit fordern (konfigurierbar, skalierbar und portabel) existierten keine Anforderungen, sodass diese nicht oder nur in geringem Maße erfüllt wurden. Nach Meinung des Autors sollten diese Eigenschaften im nächsten Schritt verwirklicht werden, nachdem die Rückverfolgbarkeit innerhalb einer definierten Umgebung ausreichend implementiert wurde. Die Nichterfüllung dieser Kriterien zeigt dennoch deutlich, dass die Fokussierung auf konkrete Systeme, wie ein bestimmtes Framework der Benutzerschnittstelle, ein hohes Risiko birgt. In der schnelllebigen Softwareentwicklung werden oft neue Technologien eingeführt, sodass die Nachhaltigkeit dieser Lösung gefährdet ist, sobald eines der Systeme ausgetauscht wird. Dies bestätigt die Relevanz der Kriterien zur Flexibilität der Rückverfolgbarkeit.

Mit der Entwicklung *WicketTracer* wurde eine praktikable Lösung zur Rückverfolgung von Anforderungen bis zur Benutzerschnittstelle des fertigen Softwareprodukts geschaffen. Gleichzeitig wurden in dieser Diplomarbeit Antworten auf drängende Probleme der Rückverfolgbarkeit von Anforderungen angeboten. In Anlehnung an Gotel et al. müssen bis 2035 dennoch erhebliche Anstrengungen unternommen werden, um eine Rückverfolgbarkeit zu etablieren, die nicht nur die Vorteile dieser Lösung besitzt, sondern auch über

organisatorische sowie technische Grenzen hinweg in einem weitaus größeren Rahmen einsetzbar ist.

Kriterium	Bewertung
Bedarfsgerecht	Die Anwendungsfälle der Stakeholder werden ermöglicht.
Kosteneffizient	Da keine zusätzlichen Kosten generiert werden, stehen Kosten und Nutzen in einem adequaten Verhältnis.
Konfigurierbar	Dies Kriterium wird nicht erfüllt, da die Rückverfolgbarkeit nicht ohne erneuten Programmieraufwand an neue Anforderungen angepasst werden kann.
Zuverlässig	Es wurde eine technische Umsetzung gewählt, die Wartung von Beziehungen zum Zeitpunkt der Änderung implementiert, sodass Inkonsistenzen gering gehalten werden können. Es besteht bereits ein großer Datensatz an Rückverfolgbarkeitsbeziehungen im Versionierungssystem „Git“, sodass bereits viele Anforderungen visualisiert werden können. Jedoch werden nur Anforderungen erfasst, die in mindestens einem Versionsstand eine Verbindung zum Quellcode der Benutzerschnittstelle besitzen. Es liegen noch keine Daten vor, die aussagen in welchem Umfang sich die Stakeholder auf diese bereitgestellte Rückverfolgbarkeit verlassen.
Skalierbar	Diese Lösung ist wenig skalierbar. Alle Elemente wie Artefakttypen, Anwendungen, Granularität der Beziehungen wurden zuvor definiert und können nicht ohne weiteren Aufwand ausgetauscht oder verändert werden.
Portabel	Innerhalb der definierten Systeme kann *WicketTracer* in verschiedenen Projekten angewandt werden. Darüberhinausgehende Portabilität wird nicht unterstützt.
Geschätzt	Rückverfolgbarkeit ist eine strategische Priorität, was aus den Motivationen von Hochschule und Praxispartner hervorgeht. *WicketTracer* wurde so geplant, dass wenig Verantwortlichkeiten von den Stakeholdern verlangt werden. Evtl. mangelnde Wertschätzung beeinflusst daher die Funktionsweise von *WicketTracer* nicht.
Omnipräsent	Das System *WicketTracer* ist jederzeit vorhanden, ohne das dessen Präsenz aktiv wahrgenommen wird, da es in den Software-Engineering-Prozess eingebettet ist. (Innerhalb der definierten Randbedingungen)

Tabelle 8.1: Bewertung der Ergebnisse anhand „The Grand Challenge of Traceability (v1.0)“

8.2 Ausblick

Diese Diplomarbeit hat gezeigt, dass es gute Gründe gibt, die Rückverfolgbarkeit von Anforderungen auf die Benutzerschnittstelle des fertigen Softwareprodukt oder dessen Zwischenstände auszudehnen. Mit *WicketTracer* wurde ein System geschaffen, welches nicht nur unter Laborbedingungen anwendbar ist, sondern für den Einsatz im Projektalltag konzipiert wurde. Es wäre erfreulich, wenn weitere wissenschaftliche oder kommerzielle Entwicklungen folgen, die die aufgestellten Ansätze weiterverfolgen. Denkbar wäre beispielsweise eine Übertragung des technischen Konzepts auf andere Systeme oder gar eine Weiterentwicklung hin zu einer universellen Lösung, die weniger von zuvor definierten Artefakttypen abhängig ist.

Für das System *WicketTracer* selbst ist gleichermaßen genügend Raum für weiterführende Entwicklungen vorhanden. Die folgenden Abschnitte erläutern konkrete Ideen, wie das System weiter ausgebaut werden kann.

8.2.1 Manuelle Markierung auf Benutzeroberfläche

In der aktuellen Version des WicketTracer-Systems muss Quellcode der Benutzerschnittstelle in einem Versionsstand mit einem Ticket verknüpft sein. Voraussetzung ist daher eine begonnene Implementierung der Anforderung. Jedoch entstehen Anforderungen im Software-Entwicklungsprozess bevor sie in Quellcode gegossen werden. Der Zeitraum von der Erstellung des Tickets bis zur ersten Referenzierung in einem Commit-Kommentar wird derzeit noch nicht abgedeckt. Weiterhin existieren Anforderungen, die nie mit der Benutzerschnittstelle in Berührung kommen und deshalb nicht vom System erfasst werden.

Um in diesen Fällen dennoch Anforderungen mit der Benutzerschnittstelle zu verlinken, könnte eine Weiterentwicklung von WicketTracer Abhilfe schaffen. Wie in Abbildung 8.1 ersichtlich, wäre eine Lösung denkbar, die in der Erfassungsmethode dem System Bugherd (vgl. Kapitel 5.2, Seite 31) ähnelt. Mit dem WicketTracer Client könnten Komponenten der Benutzerschnittstelle per Mausklick markiert und mit Tickets verknüpft werden. Als eindeutiges Merkmal der Benutzerschnittstelle könnte eine Kombination aus den Attributen „Wicket-Path“ und „Wicket-Source“ verwendet werden (vgl. Kapitel 5.1, Seite 29 und Kapitel 5.3, Seite 34). Wicket-Path-Attribute steigern erheblich die Granularität der Beziehung, sodass Markierungen nahezu frei auf der Webseite platziert werden können. Anhand der Wicket-Source-Attribute kann festgestellt werden, welche Maske der Benutzerschnittstelle gemeint ist, da Wicket-Path-Attribute nur für eine Maske eindeutig sind. Weiterhin können Wicket-Source-Attribute als Rückfallebene (Fallback) dienen, für den Fall dass sich Wicket-Path-Attribute durch Quellcodeänderungen verändert haben. Durch diesen hybriden Ansatz besitzen diese Beziehungen die gleiche Nachhaltigkeit wie jene im aktuellen System.

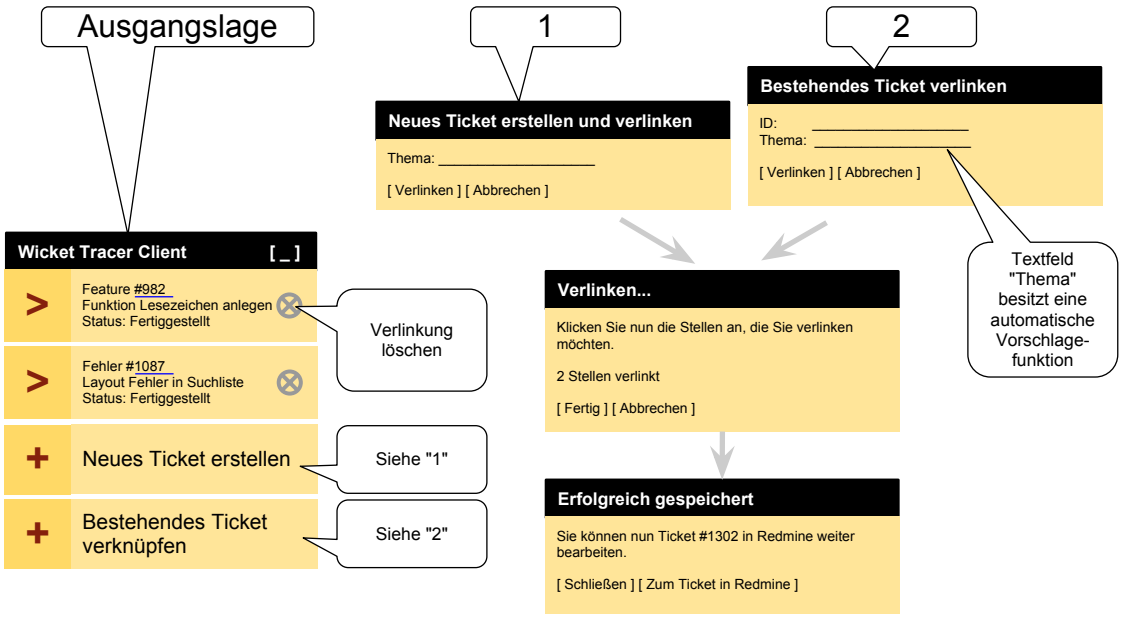


Abbildung 8.1: Entwurf der Benutzerschnittstelle der Weiterentwicklung zur manuellen Markierung

8.2.2 Vorwärts gerichtete vertikale Rückverfolgung

Mithilfe WicketTracer können ausgehend von einer Komponente der Benutzerschnittstelle der ausgeführten Applikation die dazugehörenden Anforderungen nachvollzogen werden. Eine „vorwärts gerichtete“ Rückverfolgbarkeit von einem Ticket in der Projektmanagementsoftware „Redmine“ auf die referenzierten Komponenten der Benutzerschnittstelle ist hingegen nicht implementiert. Im Rahmen der Projektüberwachung könnte es jedoch vorteilhaft sein, wenn Tickets nach bestimmten Masken der Benutzerschnittstelle gruppiert werden können. Somit müsste die Webapplikation nicht ausgeführt werden, um WicketTracer zu verwenden.

Die Gruppierung von Anforderungen nach bestimmten Kategorien wird bereits von Redmine unterstützt. Für ein Softwareprojekt im Unternehmen wurde die Projektmanagementsoftware bereits so konfiguriert, dass über die Kategorie die Zuständigkeit für eine Maske kenntlich gemacht werden kann. Im Projektalltag hat sich diese Lösung jedoch aufgrund des hohen Pflegeaufwandes als wenig praktikabel erwiesen, sodass nicht in allen Tickets die Masken-Zuordnung gepflegt wurde. Folglich war diese Lösung nicht mehr ausreichend zuverlässig. Weiterhin kann ein Ticket nicht für mehrere Masken gleichzeitig zuständig sein, da keine Mehrfachvergabe von Kategorien unterstützt wird.

WicketTracer hingegen kann präzise Tickets bestimmten Masken zuordnen. Diese Zuordnungen könnten bei jeder Ausführung des Systems für die spätere Auswertung abgespeichert werden. Es könnte eine Erweiterung (Plugin) für Redmine entwickelt werden, die diese Informationen ausliest und beispielsweise eine Gruppieren-Funktion bereitstellt. Eine manuelle Pflege der Kategorien und Zuordnungen wäre dann nicht mehr nötig.

8.2.3 Verbesserungen des WicketTracer Clients

Um die Bedienung des Clients komfortabler zu gestalten sind weitere Verbesserungen denkbar.

Aktuell werden im WicketTracer Client alle Anforderungen einer Maske der Webapplikation als Liste dargestellt. Fährt der Anwender mit dem Mauszeiger über diese Listeneinträge, werden die verknüpften Komponenten in der Benutzerschnittstelle farblich hervorgehoben. In einer Erweiterung zur Listendarstellung könnte der Nutzer einen direkteren Bezug zu den Komponenten erhalten, indem Informationen zu den Anforderungen erscheinen, sobald er mit dem Mauszeiger in der Applikation über die Elemente fährt. Im Entwurf in Abbildung 8.2 werden z. B. Anforderungen zu einem Textfeld eingeblendet. Diese Art der Visualisierung wurde in der Facharbeit von E. Gorning angewandt (vgl. Kapitel 5.1, 29).

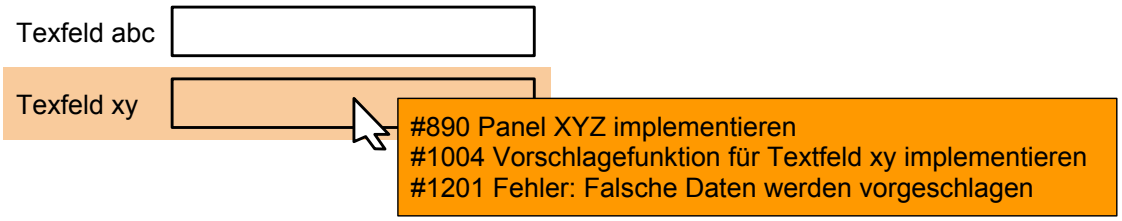


Abbildung 8.2: Anforderungen erscheinen direkt bei einer Komponente der Benutzerschnittstelle

Die Erprobung des WicketTracer-Systems in einem realen, nahezu fertiggestellten Softwareprojekt hat ergeben, dass im Schnitt 20 bis 60 Anforderungen pro Maske angezeigt werden. Durch die Sortierung nach dem letzten Ereignis werden dennoch in der Informationsfülle die relevanten Tickets auf den ersten Plätzen dargestellt. Um die Menge an Einträgen der Liste weiter einzugrenzen, könnten Filtermöglichkeiten implementiert werden. Sehr nützlich könnte sich das Filtern nach bestimmten Status anbieten, um beispielsweise fertiggestellte Anforderungen auszublenden.

Anhang A: Interviewfragen Ist-Zustand

Um eine geeignete Lösung für die intecsoft GmbH zu finden, muss zunächst der Ist-Zustand erhoben werden. Hier stellen sich Fragen zu existierenden Rollen, Tätigkeiten und Prozessen im Unternehmen. Im Anschluss werden Möglichkeiten zur Optimierung der Rückverfolgbarkeit von Anforderungen im Unternehmen diskutiert.

A – Eigene Rolle im Unternehmen

1. Bezeichnung
2. Welche eigenen Tätigkeiten werden wie ausgeführt?
    a. **Kundenkontakt** (Vertrieb, Angebot schreiben, Vertragsgestaltung, Kontaktperson während des Projekts, Leitung von Meetings, sporadischer Kontakt bei spezifischen Fragen)
    b. **Projektmanagement** (Management von Zeit, Ressourcen, Budget, Entscheidungsgewalt)
    c. **Anforderungsmanagement** (Kundenwünsche analysieren, Anforderungen erheben, warten, Anforderungsdokumente erstellen)
    d. **Spezifikation** (Aufgrund von Anforderungen System spezifizieren)
    e. **Codierung** (Aufgrund von Spezifikation Funktionen implementieren, Wer wird bei Unklarheiten in Spezifikation kontaktiert?)
    f. **Testen** (Wie?)
    g. **Dokumentation** (Welche Themen werden zu welchen Zeitpunkten im Wiki dokumentiert? Tickets warten. Abschlussdokumentation)
    h. **Controlling** (Daten, Kennzahlen analysieren, Optimierungen herausarbeiten)

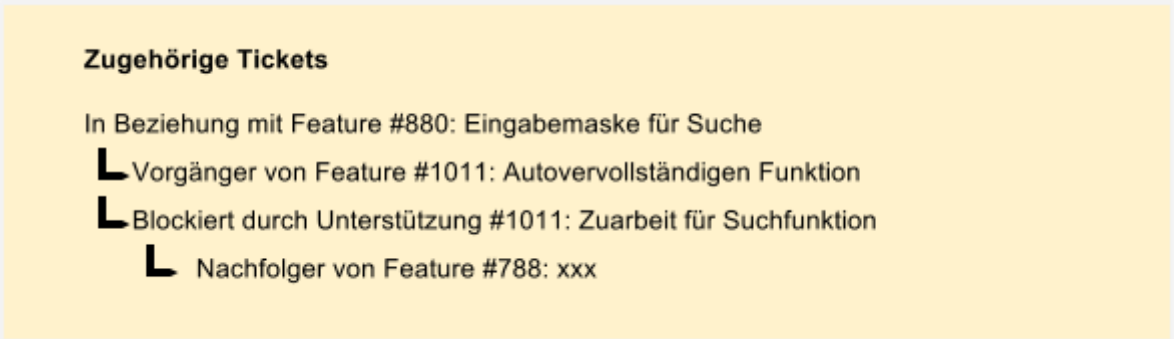
B – Weitere Tätigkeiten und Prozesse im Unternehmen

1. Welche Prozesse - bezogen auf die Softwareentwicklung - gibt es derzeit im Unternehmen?
2. Wie findet die Anforderungserhebung statt?
3. Wie werden Anforderungsdokumente erstellt, verwaltet und gewartet?
4. Was wird derzeit unternommen, um Rückverfolgbarkeit von Anforderungen zu gewährleisten? (Begriff vorher erklären!)
5. Welche Tools und Plugins werden für das Anforderungs- und Projektmanagement verwendet?

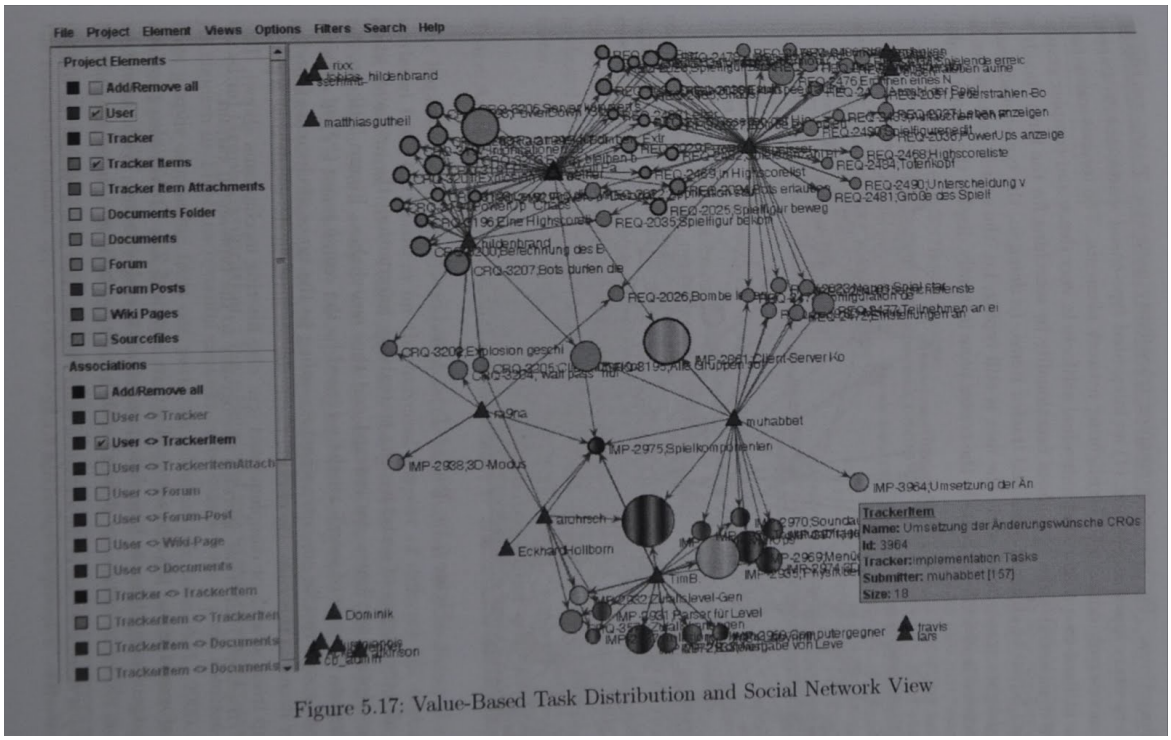
C – Vorschläge zur Verbesserung der Rückverfolgbarkeit von Anforderungen

Nach einer Einführung (Traceability, Awareness) in das Diplomthema sollen folgenden Vorschläge zur Optimierung der Rückverfolgbarkeit **diskutiert und evaluiert** werden. Insbesondere sollen **Nutzen, Schlüsselanforderungen, Machbarkeit und Risiken** besprochen werden – *z.B. "Schönes Tool, aber um es effektiv einzusetzen, müssten Prozesse anders laufen oder die Benutzung extrem schnell gehen."*

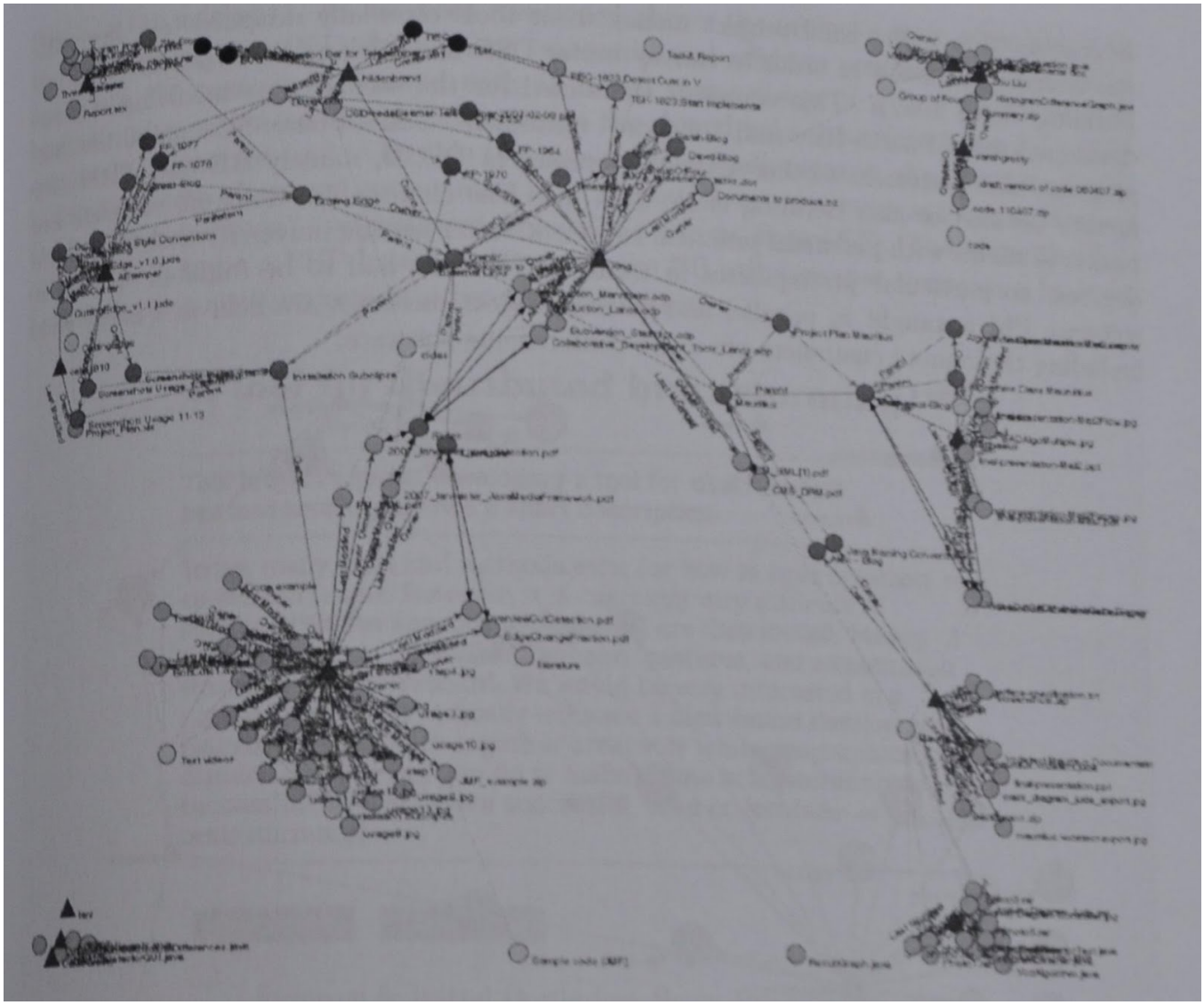
1. **Visualisierung**: Tiefergehende Darstellung von Beziehungen in Tickets (Baumdiagramm in jedem Ticket)



2. **Visualisierung**: Graphendarstellung aller Tickets (wie Hildebrand)[1]



[1] Hildenbrand, Tobias: Improving Traceability in Distributed Collaborative Software Development : A Design Science Approach. 1. Aufl.. : Peter Lang Pub Incorporated, 2008.



3. **Weitere Artefakte verlinken / Visualisierung**: Markierung von Elementen
    a. im Code (Lösungsvorschlag? Annotations? Versionierung ausreichend?)
    b. in der Benutzerschnittstelle in der ausgeführten Anwendung (Markierung ähnlich Bugherd[2]?)
    c. in der Benutzerschnittstelle in Entwürfen
    d. in Diagrammen, z.B. UML
4. **Eigene Vorschläge?**

Feedback zur Befragung

Optimierungvorschläge bzgl. Methode, einzelner Fragen, … ?

[2] http://www.bugherd.com/, abgerufen am 20. Juni 2012

Anhang B: Frühere Versionen des Systementwurfs

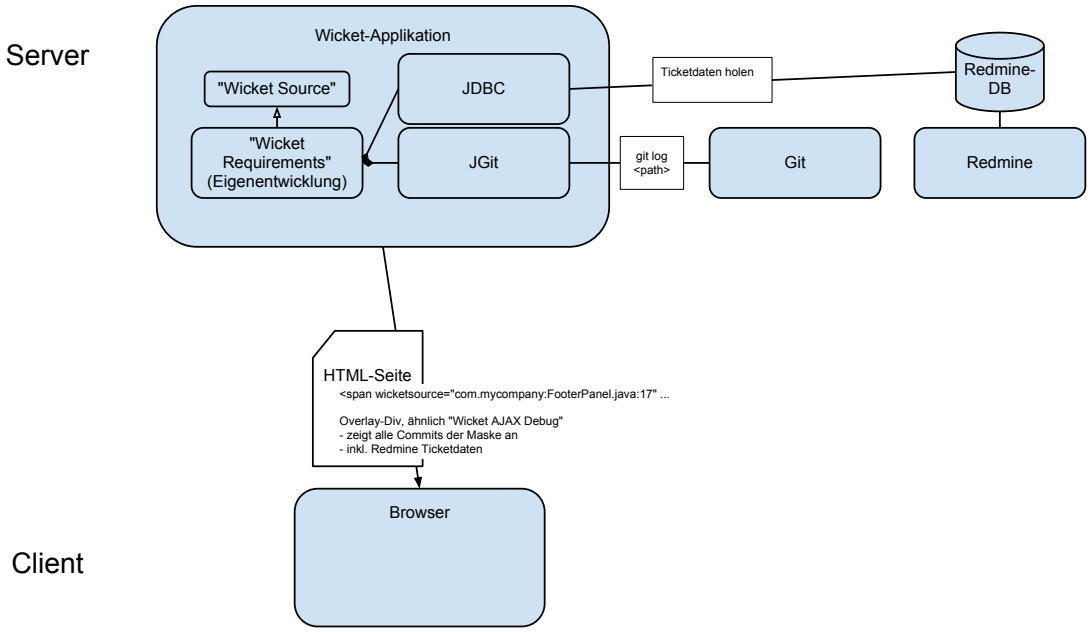


Abbildung B.1: Version 1 „Starke Integration in Wicket-Applikation, geringe Entkoppelung, keine clientseitige Logik“

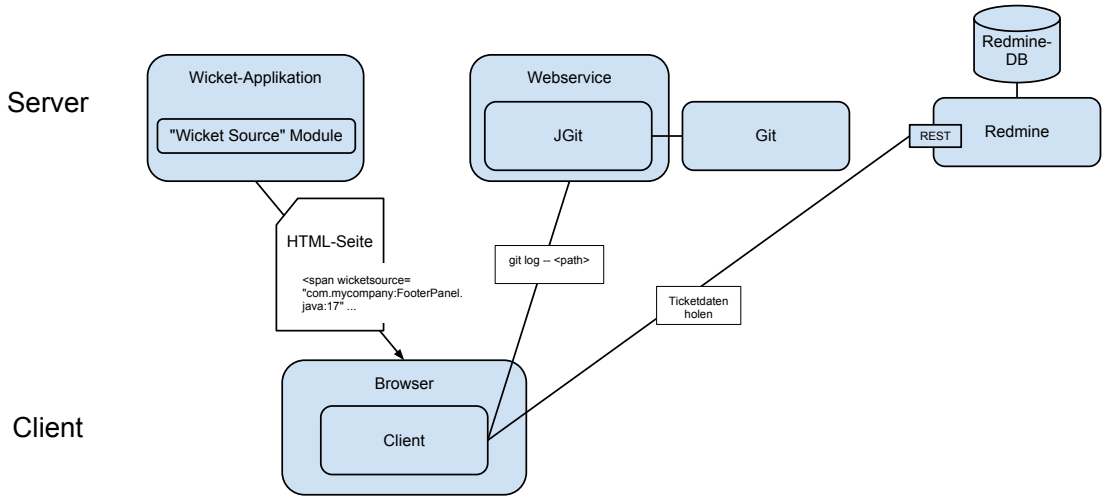


Abbildung B.2: Version 2 „Geringe Integration in Wicket-Applikation, hohe Entkoppelung, hauptsächliche Logik im Client“

Anhang C: Frühere Entwürfe der Benutzerschnittstelle

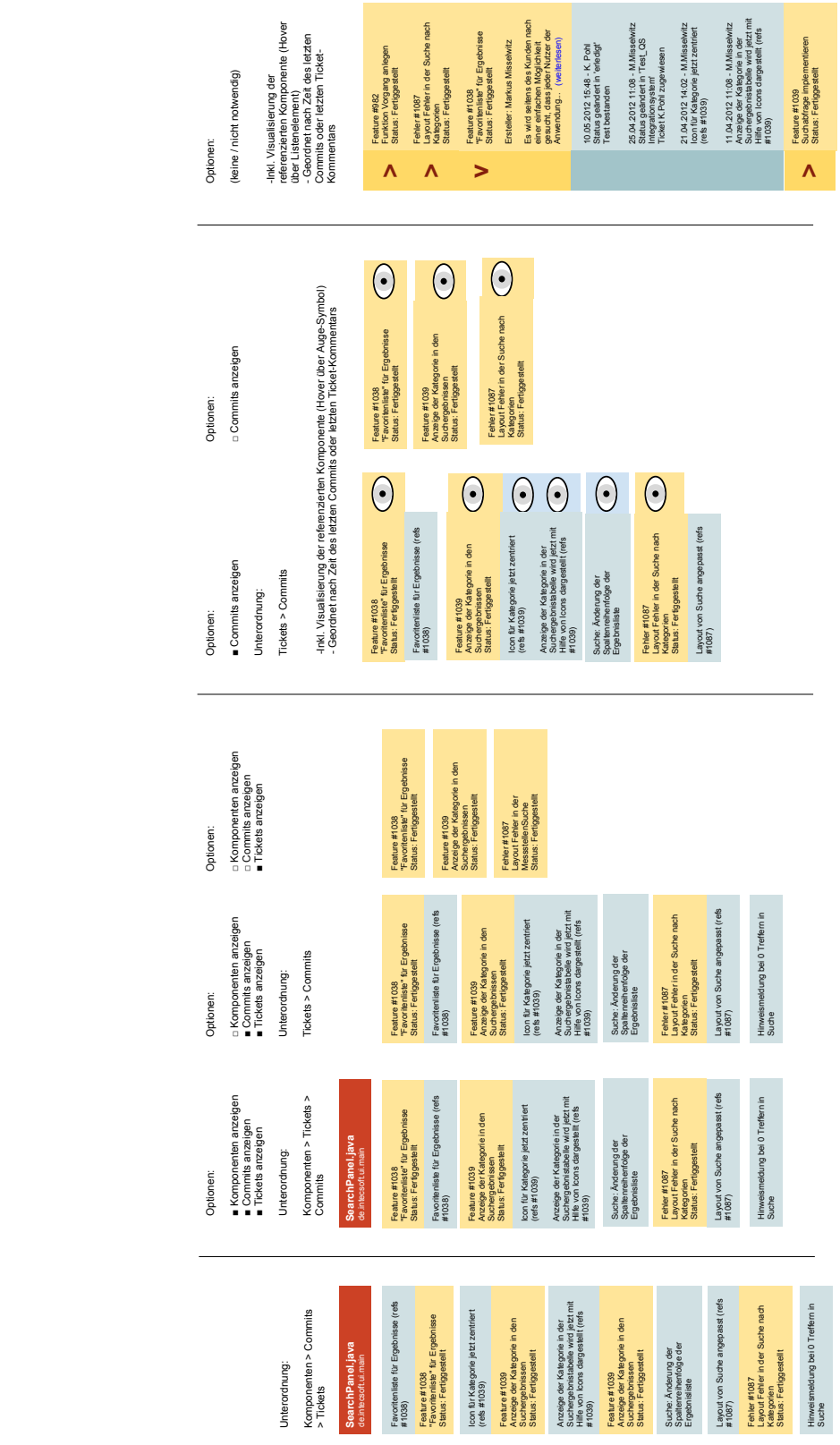


Abbildung C.1: Frühere Entwürfe der Benutzerschnittstelle

Literaturverzeichnis

[42L12] 42Lines.net: *Announcing Wicket-Source and Plugins*. Webseite, 2012. – https://www.42lines.net/2012/01/31/announcing-wicket-source/, verfügbar am 18. September 2012.

[AMR01] Arkley, Paul ; Mason, Paul ; Riddle, Steve: Position Paper: Enabling Traceability. In: *Proceedings of 1st International Workshop on Traceability in Emerging Forms of Software Engineering (TEFSE02). In conjunction with the 17th IEEE International Conference on Automated Software Engineering (ASE02)*. Edinburgh, UK : IEEE Computer Society, 2001, S. 26–31

[ARNR06] Aizenbud-Reshef, Neta ; Nolan, Brain T. ; Rubin, Julia: Model traceability. In: *IBM Systems Journal 45(3)*. IBM Haifa Research Lab, Haifa University Campus, Mount Carmel, 31905, Israel, 2006. – ISSN 0018–8670, S. 515–526. – http://dx.doi.org/10.1147/sj.453.0515, verfügbar am 30. August 2012.

[Bea12] Bearman, Matt: *Bugmuncher - Feedback Tab Widget for Websites*. Webseite, 2012. – http://bugmuncher.com, verfügbar am 17. September 2012.

[BP09] Bouillon, Elke ; Philippow, Ilka: Requirements-Traceability in der industriellen Praxis - Ziele und Einsatz - Auswertung einer Umfrage / Technische Universität Ilmenau, Fachgebiet Softwaresysteme/Prozessinformatik. 2009. – Forschungsbericht. – http://www.theoinf.tu-ilmenau.de/~riebisch/traceability/BouillonPhilippow-TraceabilityUmfrage120703.pdf, verfügbar am 31.08.2012 - E-Mail: elke.bouillon|ilka.philippow@tu-ilmenau.de

[Bug12a] Bugherd: *Bugherd*. Webseite, 2012. – http://www.bugherd.com/, verfügbar am 17. September 2012.

[Bug12b] Bugherd: *Redmine: BugHerd-Support*. Webseite, 2012. – http://support.bugherd.com/entries/20419122-redmine, verfügbar am 17. September 2012.

[EGHB09] Egyed, Alexander ; Grünbacher, Paul ; Heindl, Matthias ; Biffl, Stefan: Value-Based Requirements Traceability: Lessons Learned. In: Lyytinen, Kalle (Hrsg.) ; Loucopoulos, Perticles (Hrsg.) ; Mylopoulos, John (Hrsg.) ; Robinson, Bill (Hrsg.): *Design Requirements*

Engineering: A Ten-Year Perspective - Design Requirements Workshop, Cleveland, OH, USA, June 3-6, 2007, Revised and Invited Papers. 1. Aufl. Berlin, Heidelberg : Springer, 2009. – ISBN 978–3–540–92965–9, S. 240–257. – http://www.egyed.org/alex/publications/Value_Based_Requirements_Traceability-Lessons_Learned.pdf, verfügbar am 29. August 2012.

[EGHB12] Egyed, Alexander ; Grünbacher, Paul ; Heindl, Matthias ; Biffl, Stefan: *Value-Based Requirements Traceability: Lessons Learned*. PDF, 2012. – http://www.egyed.org/alex/publications/Value_Based_Requirements_Traceability-Lessons_Learned.pdf, verfügbar am 29. August 2012.

[GCHH+11] Gotel, Orlena ; Cleland-Huang, Jane ; Hayes, Jane H. ; Zisman, Andrea ; Egyed, Alexander ; Grünbacher, Paul ; Dekhtyar, Alex ; Antoniol, Guiliano ; Maletic, Jonathan: The Grand Challange of Traceability (v1.0). In: *Software and Systems Traceability*. 2011, S. 343–409. – Technical Report #CoEST-2011-001 from Center of Excellence for Software Traceability www.coest.org

[GCHH12] Gotel, Orlena ; Cleland-Huang, Jane ; Hayes, Jane H. ; ..: Traceability Fundamentals. In: *Software and Systems Traceability*. Springer, 2012

[GF94] Gotel, Orlena ; Finkelstein, Arthur: An analysis of the requirements traceability problem. In: *Proceedings of the 1st IEEE International Conference on Requirements Engineering*. Colorado Springs, CO, USA, 4 1994, S. 94–101. – http://eprints.ucl.ac.uk/749/1/2.2_rtprob.pdf, verfügbar am 30.08.2012

[Git12] Git: *Git*. Webseite, 2012. – http://git-scm.com/, verfügbar am 29. Oktober 2012.

[Goo12] Google: *Google Feedback*. Webseite, 2012. – http://www.google.com/tools/feedback/intl/en/index.html, verfügbar am 17. September 2012.

[HS12] Hruschka, Peter ; Starke, Gernot: *arc42: Ressourcen für Software-Architekten*. Webseite, 2012. – http://www.arc42.com/template/template/3-context.html, verfügbar am 27. September 2012.

[IR12] Ingram, Claire ; Riddle, Steve: Cost-Benefits of Traceability. In: *Software and Systems Traceability*. 2012, S. 23–43. – Newcastle University, NE1 7RU, England, UK, E-Mail: claire.ingram@ncl.ac.uk

[jav12] JAVA.NET: *GlassFish » Jersey*. Webseite, 2012. – http://jersey.java.net/, verfügbar am 10. Oktober 2012.

[JBG12] JUNGMANN, Claus ; BARTL, Peter ; GRUNERT, Marco: *Intecsoft Unternehmen*. Webseite, 2012. – http://www.intecsoft.de, verfügbar am 20. Juni 2012.

[JGi12] JGIT: *JGit*. Webseite, 2012. – http://www.eclipse.org/jgit/, verfügbar am 26. September 2012.

[jQu12a] JQUERY: *Dialog | jQuery UI*. Webseite, 2012. – http://jqueryui.com/dialog/, verfügbar am 9. Oktober 2012.

[jQu12b] JQUERY: *jQuery: The Write Less, Do More, JavaScript Library*. Webseite, 2012. – http://jquery.com/, verfügbar am 9. Oktober 2012.

[Jun11] JUNGMANN, Claus: *Modellierung von Geschäftsprozessen*. Präsentation, 2011. – anlässlich zu einem internen Mitarbeiterworkshop im November 2011

[LMOP12] LUCIA, Andrea D. ; MARCUS, Andrian ; OLIVETO, Rocco ; POSHYVANYK, Denys: Information Retrieval Methods for Automated Traceability Recovery. In: *Software and Systems Traceability*. 2012, S. 71–98. – University of Molise, Pesche (IS), Italien, E-Mail: rocco.oliveto@unimol.it

[Mä09] MÄDER, Patrick: *Rule-Based Maintenance of Post-Requirements Traceability* -. 1. Aufl. Münster : MV-Verlag, 2009. – ISBN 978—3–8–69–91–0

[MRP06] MÄDER, Patrick ; RIEBISCH, Matthias ; PHILIPPOW, Ilka: Aufrechterhaltung von Traceability Links während evolutionärer Softwareentwicklung. In: *Softwaretechnik Trends: Mitteilungen der Fachgruppen Ada (Ada), Objektorientierte Software-Entwicklung (OOSE), Requirements Engineering (RE), Software-Architektur (SWA), Software-Reeng*. Bonn : Köllen, 2006. – ISBN 0720–8928. – Technische Universität Ilmenau, Fachgebiet Softwaresysteme/Prozessinformatik, E-Mail: patrick.maeder@tu-ilmenau.de

[Poh96] POHL, Klaus: *PRO-ART - Enabling Requirements Pre-traceability*. 96-4. Aachen : RWTH, Fachgruppe Informatik, 1996

[Red12a] REDMINE: *Redmine - Overview*. Webseite, 2012. – http://www.redmine.org/, verfügbar am 29. Oktober 2012.

[Red12b] REDMINE: *Rest api - Redmine*. Webseite, 2012. – http://www.

redmine.org/projects/redmine/wiki/Rest_api, verfügbar am 30. August 2012.

[TA12] Task-Adapter: *taskadapter/redmine-java-api - Github*. Webseite, 2012. – https://github.com/taskadapter/redmine-java-api, verfügbar am 27. September 2012.

[Wic12] Wicket: *Apache Wicket*. Webseite, 2012. – http://wicket.apache.org/, verfügbar am 29. Oktober 2012.

[WP11] Wilde, Erik ; Pautasso, Cesare: *REST: From Research to Practice* -. 1st Edition. Berlin, Heidelberg : Springer, 2011. – ISBN 978–1–441–98302–2

Glossar

Apache Wicket Ein auf Java basierendes Framework zur Erstellung von Webapplikationen. Wicket ist im Gegensatz zu herkömmlichen Model-View-Controller-Webframeworks stark komponentenorientiert, ähnlich wie die Java Swing Oberfläche. Die Open-Source-Software wurde von der Apache Foundation entwickelt und steht unter der Apache License, Version 2.0.

Artefakt Ein Artefakt der Rückverfolgbarkeit ist ein nachverfolgbares Objekt, z. B. eine Anforderung, eine Gruppe von Anforderungen, eine UML-Klasse, eine Java-Klasse, ein Quellcode-Versionsstand, eine Komponente der Benutzerschnittstelle.

Artefakttyp Eine Bezeichnung, die Artefakte gleicher bzw. ähnlicher Struktur und/oder Zweck charakterisiert. Z. B. können Anforderungen, Spezifikation, Testfälle, Versionsstände, Quellcode, Komponenten in der Benutzerschnittstelle unterschiedliche Artefakttypen sein.

Client-Server-Struktur Bei dieser Struktur wird die Lösung von Aufgaben durch ein zweigeteiltes System durchgeführt. Im Regelfall sind die zwei Komponenten im Netzwerk auf unterschiedlichen Computern verteilt. Der Client stellt Anfragen an den Server und erhält von diesem eine Anwort, die z. B. einen angeforderten Datensatz beinhaltet.

Framework Zu dt. 'Rahmenstruktur' oder im weiteren Sinne 'Ordnungsrahmen' bezeichnet ein wiederverwendbares, universales Programmiergerüst zum Entwickeln von Anwendungen. Im Regelfall gibt es die Softwarearchitektur vor. Das *Framework 'Apache Wicket'* bildet beispielsweise den Rahmen einer Webanwendung. Die Art, wie die HTML-Seiten generiert werden, ist bereits vorgegeben. Der Entwickler muss seine Logik in die dafür vorgesehenen Abschnitte integrieren.

Git Verteiltes Versionsverwaltungssystem, welches ursprünglich zur Verwaltung des Quellcodes des Linuxkernels entwickelt wurde. Git ist Open Source und steht unter GNU General Public License v2 (GPL).

JGit JGit ist eine Java-Bibliothek, die das Versionsverwaltungssystem Git implementiert. Die Entwicklung wurde ursprünglich begonnen, um einen Git-Client für die Entwicklungsumgebung 'Eclipse' zu schaffen. Die Open-Source-Software steht unter Eclipse Distribution License, Version 1.0.

Komponente der Benutzerschnittstelle Bezeichnet ein Element der grafischen Benutzeroberfläche. Es existieren Interaktionselemente (z. B. Menüs, Schaltflächen, Auswahllisten, Hyperlinks, Texteingabefelder), sowie Elemente, die keine Interaktion unterstützen und nur zur Organisation (z. B. Rahmen, die Elemente zusammenfassen) bzw. Darstellung von Daten (z. B. nummerierte Listen) dienen. Das

komponentenbasierte Framework ‘Apache Wicket’ bietet eine reiche Auswahl an Klassen, die Komponenten der Benutzeroberfläche implementieren.

Lastenheft Enthält die Anforderungen aus Sicht des Kunden. Nach DIN 69901-5 (Begriffe der Projektentwicklung) repräsentiert das Lastenheft die ‘vom Auftraggeber festgelegte Gesamtheit der Forderungen an die Lieferungen und Leistungen eines Auftragnehmers innerhalb eines Auftrages’.

Maske der Benutzerschnittstelle Wird auch als ‘Bildschirmmaske’, ‘Bildschirmformular’ oder nur ‘Maske’ bezeichnet. Es handelt sich um ein auf dem Bildschirm angezeigtes Formular zur Eingabe von Daten. Eine Maske beinhaltet Komponenten der Benutzerschnittstelle.

Overhead Zusätzliche Informationen, die nicht zu den Nutzdaten zählen, keinen direkten Nutzen erzeugen und evtl. entbehrlich sind.

Persistenzschicht Nichtflüchtige Datenrepräsentationsschicht, die beispielsweise durch Datenbanken oder Dateisysteme realisiert wird.

Pflichtenheft Im Pflichtenheft beschreibt der Auftragnehmer, wie und womit er gedenkt, die Anforderungen des Kunden zu erfüllen. Nach DIN 69901-5 umfasst das Pflichtenheft die ‘vom Auftragnehmer erarbeiteten Realisierungsvorgaben aufgrund der Umsetzung des vom Auftraggeber vorgegebenen Lastenhefts’.

Redmine Webbasierte Software zum Management von Projekten. Enthalten sind Ticketverwaltung, Wikis, Anbindung an Versionsverwaltungssysteme, REST-Schnittstelle, Dokumentenablage, Diskussionsforen und eine Benutzerverwaltung. Die Software lässt sich durch zahlreich verfügbare Plugins erweitern. Redmine ist Open Source und steht unter der GNU General Public License v2 (GPL).

REST REpresentational State Transfer (REST) ist ein Softwarearchitektur-Stil für Webservices. Der Begriff wurde in der Dissertation von Roy Fielding aus dem Jahre 2000 vorgestellt und definiert. Ein REST-Dienst basiert auf den fünf Prinzipien: Addressierbarkeit, Unterschiedliche Repräsentationen (z. B. HTML, JSON oder XML), Zustandslosigkeit, Operationen (wie GET, POST, PUT, DELETE), Verwendung von Hypermedia (Ressourcen, die Links zu weiteren Ressourcen enthalten).

Review Manuelle Überprüfung von Arbeitsergebnissen der Softwareentwicklung durch eine oder mehrere andere Person(en).

Stakeholder Projektbeteiligte oder auch nur vom Projekt betroffene Personen oder Personengruppen. Typische Stakeholder in Softwareprojekten sind z. B. Auftraggeber, Auftragnehmer, Kunden des Auftraggebers, Anwender.

Versionsstand Beschreibt einen Datensatz im Versionsverwaltungssystem, der eine Quellcodeänderung enthält. Im System *Git* wird dieser als ‘Commit’ bezeichnet. Er enthält neben den Differenzen zum letzten Versionsstand auch einen Kommentar, der u. a. dazu genutzt wird, die ID der Anforderung zu vermerken.

Versionsverwaltungssystem System, das Änderungen von Dateien erfasst und verwaltet. Wird in Softwareprojekten zum Verwalten des Quellcodes verwendet, wobei jeder Bearbeiter seine Änderungen nach Erfüllung einer Aufgabe eincheckt. Ermöglicht u. a., dass mehrere Mitarbeiter am Quellcode gleichzeitig arbeiten, Änderungen nachvollzogen oder Versionen zurückgesetzt werden können.

Erklärung

Hiermit erkläre ich, dass ich die vorliegende Arbeit selbstständig und nur unter Verwendung der angegebenen Literatur und Hilfsmittel angefertigt habe.

Stellen, die wörtlich oder sinngemäß aus Quellen entnommen wurden, sind als solche kenntlich gemacht.

Diese Arbeit wurde in gleicher oder ähnlicher Form noch keiner anderen Prüfungsbehörde vorgelegt.

Mittweida, 1. November 2012