Versionierung von Anforderungen

MASTER THESIS

Sebastian Geitner

Eingereicht am 21. September 2021

Friedrich-Alexander-Universität Erlangen-Nürnberg
Technische Fakultät, Department Informatik
Professur für Open-Source-Software

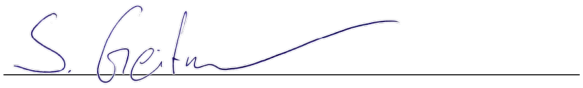
Betreuer:
Julia Krause M.Sc.
Prof. Dr. Dirk Riehle, M.B.A.

FRIEDRICH-ALEXANDER
UNIVERSITÄT
ERLANGEN-NÜRNBERG

TECHNISCHE FAKULTÄT

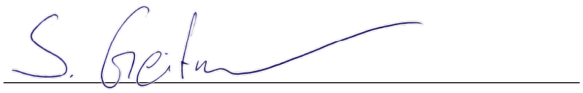
Versicherung

Ich versichere, dass ich die Arbeit ohne fremde Hilfe und ohne Benutzung anderer als der angegebenen Quellen angefertigt habe und dass die Arbeit in gleicher oder ähnlicher Form noch keiner anderen Prüfungsbehörde vorgelegen hat und von dieser als Teil einer Prüfungsleistung angenommen wurde. Alle Ausführungen, die wörtlich oder sinngemäß übernommen wurden, sind als solche gekennzeichnet.



Erlangen, 21. September 2021

License



Erlangen, 21. September 2021

Abstract

The versioning of requirements ensures completeness and traceability. This means that it is possible to trace which user has made changes at any time. The mandatory specification of a reason for change means that a version history can be clearly structured by means of a filter mechanism, even in the case of a large number of changes. In addition, versioning can be used to determine key figures of a project with regard to a change rate. In conjunction with the QDAcity-RE method and a link between codes and requirements, necessary changes to requirements can be visualized. This ensures an increase in productivity and a reduction in the error rate.

Zusammenfassung

Die Versionierung von Anforderungen sorgt für Vollständigkeit und Nachverfolgbarkeit. So kann zu jedem Zeitpunkt nachvollzogen werden, welcher Nutzer Änderungen vorgenommen hat. Durch die verpflichtende Angabe eines Änderungsgrundes kann eine Versionshistorie auch bei einer großen Anzahl an Änderungen durch einen Filtermechanismus übersichtlich gestaltet werden. Zudem lassen sich durch eine Versionierung Kennzahlen eines Projektes bezüglich einer Änderungsrate ermitteln. In Verbindung mit der QDAcity-RE Methode und einer Verknüpfung von Codes und Anforderungen lassen sich notwendige Änderungen an Anforderungen visualisieren. Das sorgt für einen Anstieg der Produktivität und eine geringere Fehlerquote.

Inhaltsverzeichnis

Acronyms

RE Requirements Engineering

QDA Qualitative Data Analysis

GUI Graphical User Interface

JSON JavaScript Object Notation

CSV Comma separated value

Abbildungsverzeichnis

Tabellenverzeichnis

1 Einleitung

Gerade in der Branche der Softwareentwicklung kommt es im Laufe eines Projektes zu einer Vielzahl an Änderungen der Anforderungen (Ebert, 2019). Zum einen scheinen Anpassungen von Software oftmals einfacher als das Anpassen von Hardware. Andererseits sorgen beispielsweise Gesetzesänderungen oder technologische Neuheiten für Veränderungen. Dadurch steigen mit zunehmender Komplexität und Dauer eines Softwareprojekts auch die Anforderungen an eine Anwendung zum Verwalten von Anforderungen. Oftmals ist es nicht möglich, den Ursprung einer Anforderung zurückzuverfolgen (Dick et al., 2017). Gerade bei größeren Projekten kann dies zu Problemen führen. Ist zum Beispiel der ursprüngliche Autor bereits aus einem Projekt ausgeschieden, kann es bei fehlender Versionierung schnell zu Verständnisverlust und Fehlinterpretationen der Anforderungen kommen, da die ursprüngliche Intention des Autors nicht zurückzuverfolgen ist. Hier kann eine Versionierung die Verfolgbarkeit der Anforderung erleichtern.
Eine gute Versionierung kann zudem die Vollständigkeit gewährleisten. So ist es möglich, dass alle Änderungen gesichert und somit zugeordnet und nachvollzogen werden können.
Diese Versionierung ermöglicht des Weiteren das Auswerten der Änderungen. Dies kann den Requirements Engineer durch umfangreiche Analysen bei seiner Tätigkeit unterstützen. Auch eine Einordnung oder Steuerung nachfolgender Projekte kann dadurch erleichtert werden.

Ziel der Arbeit

Das Ziel der Arbeit ist die Entwicklung eines Versionierungskonzeptes für Anforderungen. Anschließend soll dieses in einem bestehenden Tool umgesetzt werden. Dem Nutzer soll hierbei ohne großen Mehraufwand eine Versionshistorie bereitgestellt werden. Durch eine Verknüpfung der Anforderungen mit dem bestehendem Tool kann so eine vollständige Verfolgbarkeit der Anforderung bis hin zu den Ausgangsdaten sichergestellt werden.
Zu Beginn der Arbeit wird sich mit Fragestellungen bezüglich des Speicherns der Versionen und dem Zeitpunkt des Anlegens von neuen Versionen auseinandergesetzt. Auch das Anlegen geeigneter Attribute und die Analyse der neu angelegten

Versionierung werden hier betrachtet. Anschließend gilt es zwischen verschiedenen Designentscheidungen abzuwiegen und eine geeignete grafische Oberfläche zu entwickeln. Abschließend folgt nach vollständiger Implementierung eine Evaluation durch Domainexperten.

2 Grundlagen

2.1 Stand QDAcity

In der folgenden Arbeit soll das Tool QDAcity, insbesondere der bestehende Specification-Editor, welcher in Abbildung 2.1 zu sehen ist, um die Funktionalität der Versionierung von Anforderungen erweitert werden. Im kommenden Abschnitt wird die Ausgangslage des Tools beschrieben.

Um die transparente Dokumentation der Erstellung von Domain-Modellen zu ermöglichen, wurde die QDAcity-RE Methode entwickelt, welche von der Qualitative Data Analysis (QDA) abgeleitet wurde (Kaufmann & Riehle, 2017). Um die Qualität der Ergebnisse im Requirements Engineering (RE) und somit auch den Erfolg eines Softwareprojektes zu steigern, werden die häufig qualitativen und unstrukturierten Ausgangsdaten mittels dieser definierten Methode analysiert und verarbeitet. Hierbei wird sich darauf fokusiert, nur die relevanten Informationen zu extrahieren, welche dann interpretiert und abstrahiert werden können. Des Weiteren sorgt die QDAcity-RE Methode für einen hohen Grad an Vollständigkeit sowie für eine Verfolgbarkeit des Ergebnisses zurück bis zu den Ausgangsdaten. Über das zu entwickelnde Tool QDAcity lässt sich bereits eine QDA durchführen. Hierbei handelt es sich um systematisches Zuordnen von Textsegmenten *(Codings)* zu verschiedenen Themen *(Codes)*. So lässt sich, wie in Abbildung 2.2

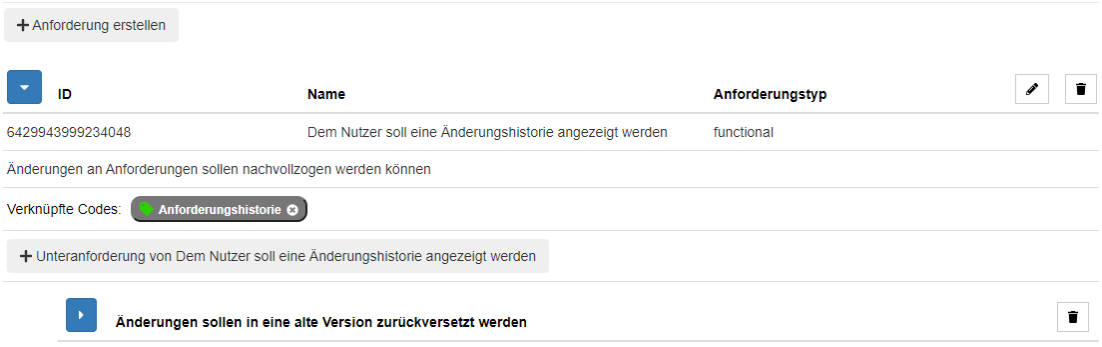


Abbildung 2.1: QDAcity Specification-Editor Ausgangslage

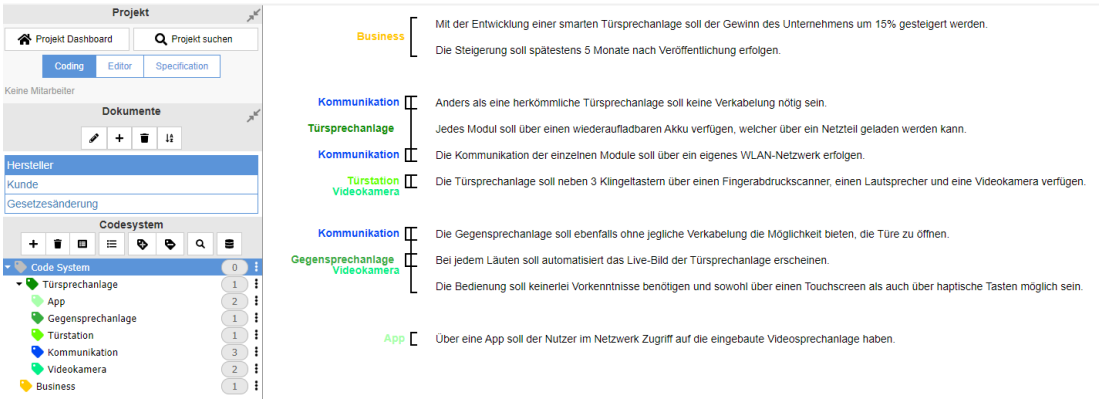


Abbildung 2.2: Beispielhafte Darstellung des CodingEditors im Tool QDAcity

beispielhaft dargestellt, aus Interviews oder Anforderungsdokumente ein zusammenhängendes Codesystem über mehrere Dokumente hinweg erstellen. Das Codesystem kann hierbei als Grundlage für die Ermittlung von Anforderungen dienen. Im Tool QDAcity existiert des Weiteren bereits der in Abbildung 2.1 Specification-Editor. Dieser ermöglicht bereits das hierarchische Anlegen von rudimentären Anforderungen. Um die Verknüpfung zwischen den Anforderungen und den mittels der QDAcity-RE Methode gesammelten Codes herzustellen, kann jede Anforderung per Drag&Drop mit beliebig vielen Codes verknüpft werden. Durch diese Verknüpfung der Codes mit Anforderungen kann ein Mehrwert im Vergleich mit bereits bestehenden Systemen geschaffen werden. So lassen sich Anforderungen auch zu einem späteren Zeitpunkt bis zu den Anforderungsdokumenten zurückverfolgen.

2.2 Requirements Engineering

> "Das Requirements Engineering ist ein systematischer und disziplinierter Ansatz zur Spezifikation und zum Management von Anforderungen mit den Zielen die Wünsche und Bedürfnisse der Stakeholder zu verstehen und das Risiko zu minimieren, ein Produkt, das nicht diese Wünsche erfüllt, auszuliefern."(SOPHISTen, 2021, S. 9)

Deshalb ist es wichtig, dass das Arbeiten mit Anforderungen wiederholbar, nachvollziehbar und begründet ist (SOPHISTen, 2021). Dies kann durch verschiedene Arten der Dokumentation sichergestellt werden. Bei größeren Projekten empfiehlt es sich jedoch auf ein geeignetes Tool zurückzugreifen, welches den Requirements Engineer bei der Dokumentation unterstützt und so dazu beiträgt, Fehler zu vermeiden. Ein weiterer essenzieller Teil des Requirements Engineering beschäftigt sich mit der Kommunikation zwischen verschiedenen Personen, welche oftmals unterschiedliche Ziele verfolgen (Ebert, 2019). Ein Softwareentwickler verfolgt so

oftmals auf Grund eines anderen Blickwinkels ein anderes Ziel als beispielsweise der Kunde. Dies führt häufig zu diversen Änderungen der Anforderungen, welche transparent verwaltet werden sollten. Einhergehend mit den nötigen Anforderungen und Eigenschaften an den Requirements Engineer, kann auch hier ein geeignetes Tool bei der Verwaltung der Anforderungen unterstützend eingesetzt werden.
Abstrahiert betrachtet lassen sich Anforderungen, neben der bekannten Differenzierung zwischen funktionalen und nicht-funktionalen Anforderungen, auch durch Unteranforderungen feiner spezifizieren (SOPHISTen, 2021). Je nach Anwendungsfeld bestehen Anforderungen aus verschiedenen Attributen, wobei vor allem auf Konsistenz, Vollständigkeit und Nachvollziehbarkeit zu achten ist.
Der Vorgang des Requirements Engineerings lässt sich nach SOPHISTen, 2021 in folgende vier Haupttätigkeiten einteilen:

I. Ermitteln von Wissen

Die erste Tätigkeit beläuft sich auf das Ermitteln von Wissen, bei dem erste Visionen und Ziele definiert werden. Oftmals sind dort jedoch noch nicht alle Anforderungsquellen und Stackeholder bekannt. Auch diese gilt es in der ersten Haupttätigkeit zu ermitteln, da diese die Grundlage für alle weiteren Haupttätigkeiten darstellen.

II. Herleiten von guten Anforderungen

Das Herleiten von guten Anforderungen stellt die zweite Haupttätigkeit im RE dar. Hier werden die zuvor ermittelten Vorgaben analysiert, um dann, je nach Vorgehensmodell, daraus gute und vollständige Anforderungen ableiten zu können.

III. Vermitteln der Anforderungen

Die dritte Haupttätigkeit behandelt das Vermitteln der Anforderungen. Bei dieser Tätigkeit werden die Anforderungen entweder natürlich sprachlich oder modellbasiert dokumentiert. Da Anforderungen meist für andere Personen, wie Entwickler oder Tester, hergeleitet werden, ist das Vermitteln der Anforderungen durchaus von nicht unerheblicher Bedeutung. Die Art der Vermittlung ist hierbei stark von den Rollen und der Expertise der Personen abhängig, an welche die Anforderungen vermittelt werden sollen.

IV. Verwalten der Anforderungen

Beim Verwalten von Anforderungen wird sich unter anderem mit einem geeignetem Versionierungskonzept beschäftigt. Auf dieser Haupttätigkeit liegt das Au-

genmerk dieser Arbeit, weshalb das Verwalten von Anforderungen in Abschnitt 2.2.1 ausführlicher betrachtet wird.

2.2.1 Anforderungsmanagement

> "Das Requirements-Management umfasst alle Maßnahmen, die notwendig sind, um Anforderungen und anforderungsbezogene Artefakte zu dokumentieren, zu ändern und nachzuverfolgen."(Rupp, 2021, S. 392)

Das Management von Anforderungen ist selbst in einem schon zu Beginn gut strukturierten Projekt von großer Bedeutung, da sich Anforderungen in jedem Projekt und zu jedem Zeitpunkt ändern können (Bühne & Hermann, 2015). Dies ist auf die sich mit der Zeit ändernden Bedingungen zurückzuführen. Bei zunehmender Projektlaufzeit spielen insbesondere Gesetzesänderungen oder Innovation eine bedeutende Rolle. Um auf solche Änderungen flexibel und schnell reagieren zu können, ist gutes Anforderungsmanagement gerade in Projekten mit zunehmender Komplexität unerlässlich. So lassen sich zu den Aufgaben beim Verwalten von Anforderungen mehrere Grundannahmen für jedes Projekt treffen (Rupp, 2021). Die Komplexität eines Projekts steigt, je mehr Personen an einem Projekt beteiligt sind, da eine Anforderung so häufiger vermittelt oder kommuniziert werden muss. Jede Person muss also wissen, wo eine Anforderung zu welchem Zeitpunkt zu finden ist. Eine zentrale Verwaltung über ein geeignetes Tool erleichtert diese Kommunikation hierbei erheblich. Auch die Wiederverwendung von Anforderungen spielt bei der Verwaltung eine wichtige Rolle. So ist es hier von Bedeutung, die Anforderungen möglichst übersichtlich aufzubereiten. Gerade im Zusammenhang mit älteren Versionen oder anderen Anforderungen darf auch hier die zunehmende Komplexität selbst in weniger umfangreichen Anforderungssammlungen nicht unterschätzt werden.
Unter dem Begriff des Anforderungsmanagements versteht man zum einen das Verwalten der Anforderung an sich, zum anderen aber auch das Änderungsmanagement. Das Verwalten der Anforderung beinhaltet hierbei vor allem das Wählen geeigneter Attribute, aber auch die Verfolgbarkeit oder die Einteilung in mehrere Unteranforderungen sind von Bedeutung (Bühne & Hermann, 2015). Um die Bedürfnisse vieler verschiedener beteiligter Personen so unkompliziert wie möglich zu erfüllen, ist es das Ziel des Anforderungsmanagements, die Anforderungen so zu verwalten, dass sie systematisch durchsucht, geändert und verfolgt werden können. Gutes Anforderungsmanagement kann somit dazu beitragen das Risiko und den Arbeitsaufwand eines Projektes zu senken und trotz geringerem Arbeitsaufwand die Qualität der Anforderungen zu steigern. Zudem kann die Kommunikation und die Kundenzufriedenheit dadurch verbessert werden.

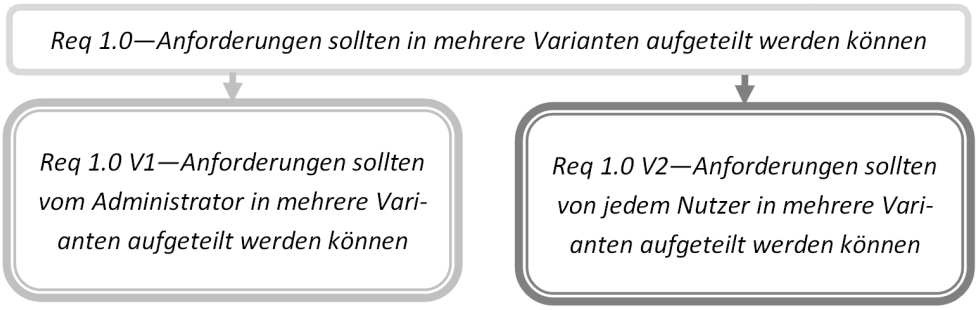


Abbildung 2.3: Beispiel: Varianten von Anforderungen

Verwalten von Anforderungen

Wie zu Beginn von Kapitel 2 bereits definiert, stellt das Verwalten von Anforderungen eine der vier Haupttätigkeiten im RE dar. Die Bedeutung dieser Tätigkeit steigt jedoch erst im Laufe eines Projekts (SOPHISTen, 2021). Hierbei muss vor allem die Nachverfolgbarkeit der Anforderungen sichergestellt werden. Des Weiteren sind beim Verwalten von Anforderungen oftmals zusätzliche Informationen zum Änderungszeitpunkt oder dem Autor zu dokumentieren.
Neben einer linearen Versionierung von Anforderungen, auf welche im Abschnitt *Versionierung und Verfolgbarkeit von Anforderungen* genauer eingegangen wird, sind auch verschiedene Varianten einer Anforderung denkbar. Abbildung 2.3 beschreibt beispielhaft, wie zwei unterschiedliche Varianten der selben Anforderung für zwei verschiedene Kunden aussehen können. Dies ist der Fall, wenn sich Anforderungen von der gleichen Basis in verschiedene Richtungen weiterentwickeln (Ebert, 2019). Diese Varianten einer Anforderung unterscheiden sich oftmals nur in Details, sei es für unterschiedliche Märkte oder für andere Kunden, müssen jedoch ebenfalls dokumentiert und verwaltet werden. So soll zum Beispiel eine Fehlerkorrektur an der Basisanforderung auch in ihren Varianten korrigiert werden.
Diese Varianten oder Stadien einer Anforderung eignen sich auch, um einen gewissen Stand der Anforderungen für beispielsweise einen Reviewprozess oder einen Release festzuhalten. So können für einen Release noch spezielle Änderungen in die Anforderungen eingebracht werden oder Anforderungen vernachlässigt werden, welche erst in einem späteren Stand von Bedeutung sind. Dieser sogenannte *Freeze* sorgt für einen unveränderlichen Stand der Anforderungen und muss deshalb ebenso festgehalten und verwaltet werden können (Bühne & Hermann, 2015). Das Verwalten von Anforderungen und insbesondere von deren Versionen und Varianten bedarf mehrerer Attribute und Eigenschaften. In der folgenden Tabelle 2.2 werden die Attribute genauer ausgeführt und definiert, welche dann als Grundlage für eine Versionierung dienen. Im nachfolgenden Abschnitt *Versionierung und Verfolgbarkeit von Anforderungen* werden diese Attribute genauer ausgeführt.

Version/Versionsnummer
Hierfür sind mehrere Schemata denkbar. Die einfachste und naheliegendste Variante ist es, ganzzahlige Versionsnummern zu verwenden *(1 → 2, 2 → 3)* (Ebert, 2019). Je nach Komplexität und Änderungsrate eines Projekts ist es jedoch empfehlenswert, eine Versionierung auf Basis von Inkrementen durchzuführen, was jedoch zu einem erhöhten Verwaltungsaufwand führt (Bühne & Hermann, 2015). So würde bei einer marginalen Änderung, wie beispielsweise einem Rechtschreibfehler, nur das Inkrement erhöht werden *(1.1 → 1.2)*. Bei inhaltlichen Änderungen würde die Version erhöht werden *(1.2 → 2.1)*. Dieses Attribut ist essentiell für eine klare Versionierung, welche im Anschnitt *Versionierung und Verfolgbarkeit von Anforderungen* von großer konzeptionellen Bedeutung ist.

Status/Zustand
Je nach Art des Workflows eines Projekts kann es von Vorteil sein, den Zustand oder auch Status, in welchem sich eine Anforderung befindet, zu dokumentieren (Ebert, 2019). Denkbar sind zum Beispiel *„angelegt“*, *„in Prüfung“* oder *„freigegeben“* (Bühne & Hermann, 2015). Dadurch lässt sich im Laufe eines Projekts nachvollziehen, wie weit die Bearbeitung einer Anforderung bereits fortgeschritten ist.

Autor
Ein weiteres nicht zu vernachlässigendes Attribut bei der Verwaltung von Anforderungen stellt der Autor dar (Bühne & Hermann, 2015; Ebert, 2019). So lässt sich bei Unklarheiten auch zu einem späteren Zeitpunkt der richtige Ansprechpartner zu einer Änderung ermitteln.

Zeitpunkt der Änderung
Analog zum Autor, sollte auch der Änderungszeitpunkt dokumentiert werden, um die Vollständigkeit und Nachvollziehbarkeit zu gewährleisten (Bühne & Hermann, 2015). Hierbei ist das Datum und auch die Uhrzeit der Änderung zu speichern.

Änderungsgrund/Kommentar
Daneben ist zu jeder Änderung ein Änderungsgrund anzugeben. So kann später im Projekt nachvollzogen werden, welches Motiv für diese Änderung ausschlaggebend war (Bühne & Hermann, 2015). So können auch Änderungen, welche nur aus Rechtschreibfehlern oder ähnlichem resultieren schnell gefiltert werden.

Quelle
Ähnlich zum Änderungsgrund kann auch der Ursprung einer Anforderung im späteren Projektverlauf von Bedeutung sein (Ebert, 2019). Dies kann zum Beispiel durch eine Verknüpfung zum Anforderungsdokument realisiert werden (SOPHISTen, 2021).

 Tabelle 2.2: Attribute zum Verwalten von Anforderungen

Historie Eine Historie der Änderungen sorgt für ein einfacheres Verständnis der Änderungen und erleichtert die Nachverfolgung durch den Nutzer (Ebert, 2019). Diese Eigenschaft wird im nachfolgenden Abschnitt von größerer Bedeutung sein.
Verfolgbarkeit Die Verfolgbarkeit stellt die Möglichkeit dar, eine Anforderung mit mehreren anderen Anforderungen zu verknüpfen und diese somit in Beziehung zueinander zu setzen (Dick et al., 2017). So können Anforderungen in verschiedenen Schichten strukturiert werden und mehrere übergeordnete und untergeordnete Anforderungen aufweisen. Auch eine Verfolgbarkeit von Anforderungen untereinander kann von Nutzen sein. Unter horizontaler Verfolgbarkeit versteht man, die Möglichkeit mehrere im Zusammenhang stehende Anforderung zu verfolgen. Dies können beispielsweise Markt- oder Kundenanforderungen in Beziehung zu anderen Marktanforderungen sein (Ebert, 2019). Analog dazu stellt die vertikale Verfolgbarkeit die Möglichkeit dar, zum Beispiel Markt- oder Kundenanforderungen im Zusammenhang mit Produkt- oder Komponentenanforderungen zu verfolgen.
Möglichkeit eines Revert Diese Eigenschaft kann hauptsächlich durch geeignete RE-Tools realisiert werden. So hat hier der Nutzer die Möglichkeit, einen Revert einer Anforderung durchzuführen und so einen alten Stand der Anforderung wiederherzustellen (Ebert, 2019). Auch diese Eigenschaft wird im Bezug auf die Versionierung von Anforderungen im späteren Verlauf der Arbeit noch von größerer Bedeutung sein.

Tabelle 2.4: Eigenschaften zum Verwalten von Anforderungen

Analog zu der vom Nutzer zu setzenden Attribute können auch Eigenschaften bei der Verwaltung von Anforderungen helfen. Hierbei handelt es sich um Merkmale, deren Darstellung zum Beispiel durch ein Tool zur Anforderungsverwaltung unterstützt werden kann. Diese Eigenschaften sind in Tabelle 2.4 dargestellt.

Versionierung und Verfolgbarkeit von Anforderungen

> "Die Versionierung von Anforderungen stellt sicher, dass der jeweils aktuelle Stand der Anforderung allgemein verbindlich und transparent genutzt wird."(Ebert, 2019, S. 266)

Die Gründe für eine Änderung einer Anforderung sind sehr vielseitig. Dabei kann es sich von kleineren Änderungen, wie zum Beispiel Rechtschreibfehler, über Umstrukturierungen, bis hin zu größeren inhaltlichen Änderungen handeln. Wichtig hierbei ist jedoch, dass strukturiert vorgegangen wird und der Umgang mit solchen Änderungen klar festgehalten ist (SOPHISTen, 2021). Vor allem in der Softwareentwicklung ist eine Änderung nicht als schlecht zu bezeichnen, da es sich um eine Korrektur handelt, und damit vielmehr als Teil des Entwicklungsprozesses anzusehen ist (Bühne & Hermann, 2015). Die Änderungsrate sollte jedoch im Laufe des Projekts abfallen, was in Abschnitt 2.2.2 genauer betrachtet wird. Zudem ist zu bedenken, dass Anforderungen auch für weitere Projekte weiterverwendet werden, weshalb besonders auf vollständige und verständliche Dokumentation zu achten ist (SOPHISTen, 2021). So haben beispielsweise verschiedene Webanwendungen oftmals grundlegend ähnliche Anforderungen an eine Nutzerverwaltung oder die Benutzbarkeit der Nutzeroberfläche. Um dort bei aus anderen Projekten wiederverwendeten Anforderungen Unklarheiten zu vermeiden, kann eine vollständige und verständliche Dokumentation hilfreich sein.
Um die Verfolgbarkeit und Transparenz zu gewährleisten, muss jede Änderung dokumentiert und eine neue Version erzeugt werden (Bühne & Hermann, 2015). Eine Version sollte hierbei immer eine vollständige Anforderung sein und nicht nur die Änderung zur vorangegangenen Version darstellen (Ebert, 2019). Beim Erzeugen einer neuen Version wird also erst eine Kopie der Vorgängerversion erzeugt, welche dann mit der neuen Version in einer linearen Kette verknüpft wird. Frühere Versionen einer Anforderung werden dann als nicht mehr aktiv gekennzeichnet (SOPHISTen, 2021). Ein Beispiel für das Bilden einer solchen linearen Kette ist in Abbildung 2.4 dargestellt. Beginnend mit *Req 1.0* wird jede neue Version mit der Vorgängerversion verknüpft, was zu einer linearen Kette führt.

Um diese Änderungen inhaltlich gut nachverfolgen zu können muss für jede Änderung eine kurze Begründung hinterlegt werden (Ebert, 2019). Das kann durch einen einfachen Änderungsgrund oder einen Kommentar des Nutzers geschehen. Zudem ist es hilfreich, den Zeitpunkt der Änderung zu dokumentieren. Dies kann durch RE-Tools erledigt werden, sodass kein aktives Handeln des Nutzers erforderlich ist. Zudem sollte eine Anforderung zum Gewährleisten der Nachverfolgbarkeit nicht gelöscht werden können. Um fälschlich angelegte oder nicht mehr benötigte Anforderungen dennoch ausblenden zu können, können bestehende Anforderungen archiviert werden (Rupp, 2021). In Verbindung mit der Versionierung von Anforderungen wird somit eine Verfolgbarkeit dieser Anforderungen über

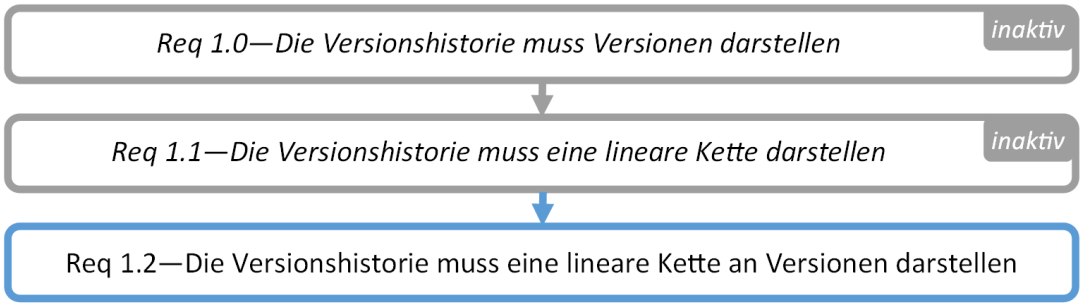


Abbildung 2.4: Beispiel: Lineare Versionen von Anforderungen

den gesamten Lebenszyklus hinweg sichergestellt (Bühne & Hermann, 2015). Dadurch lassen sich auch Aussagen über die Änderungshäufigkeit und den Ursprung der Anforderung treffen. Diese Analyse der Versionierung wird im folgenden Abschnitt 2.2.2 behandelt.

2.2.2 Analyse von Versionierung

Der Mehrwert der Versionierung von Anforderungen liegt zum einen, wie bereits beschrieben, in der vollständigen Dokumentation und Verfolgbarkeit der Änderungen. Zum anderen lassen sich aus den dokumentieren Informationen verschiedene Kennzahlen ableiten, welche für das aktuelle Projekt von Bedeutung sein können, aber auch für die Optimierung kommender Projekte genutzt werden können. Diese Kennzahlen und die Einordnung dieser wird im aktuellen Abschnitt genauer betrachtet. Zu beachten gilt es, dass bei der Auswertung der Kennzahlen auch das Bezugssystem von Bedeutung ist, um die Zahlen richtig ins Projekt einordnen zu können (Ebert, 2019). Es bietet also keinen Mehrwert die Kennzahlen ohne Intention zu sammeln. Es ist ebenso von Bedeutung die Kennzahlen nach einem festgelegtem Prozess auszuwerten und auf die richtige Zielgruppe anzuwenden. Eine Definition dieser Kennzahlen ist also essentiell. Um das Messen der Kennzahlen möglichst vergleichbar zu gestalten eignet sich der ISO/IEC/IEEE 15939 Standard („ISO/IEC/IEEE International Standard - Systems and software engineering–Measurement process“, 2017), welcher erstmals im April 2017 veröffentlicht wurde. Darin werden verschiedene Prozesse zum Erfassen von Kennzahlen spezifiziert. Wichtig hierbei ist, dass sich dieser Prozess wiederholbar gestaltet, sodass die Vergleichbarkeit der Daten gewährleistet ist.
Im Folgenden werden die relevantesten Kennzahlen eines Projekts vorgestellt:

Anforderungsanzahl

Die Anforderungszahl stellt die absolute Anzahl an Anforderungen in einem Projekt dar. Dies kann dazu dienen, den groben Umfang und den Aufwand eines Projektes abzuschätzen (SOPHISTen, 2021). Das verwenden einer Satzschablone

zum Formulieren der Anforderungen erhöht hierbei die Aussagekraft der Kennzahl erheblich (Ebert, 2019). Durch einen strukturierten Aufbau der Sammlung an Anforderungen können redundante Anforderungen vermieden werden. Ebenso kann diese Zahl bei der Betrachtung einer einzelnen Anforderung angewendet werden, indem hier die Anzahl an Versionen ausgewertet wird. Zusätzlich zur Betrachtung einer einzelnen Anforderung ist es auch möglich die gesamte Anzahl an Versionen auf ein Projekt auszuwerten. Um ein besseres Verhältnis zur Anforderungsanzahl zu erhalten, eignet sich die Betrachtung der Änderungsrate, welche später genauer ausgeführt wird.

Anforderungssatus

Hierbei wird die Gesamtzahl an Anforderungen, welche sich in einem gewissen Zustand befinden, gemessen (Ebert, 2019). Diese Zahl zeigt vor allem den Fortschritt eines Projekts und wird üblicherweise wöchentlich ausgewertet. Mit zunehmender Projektdauer steigt auch die Bedeutung und Aussagekraft dieser Kennzahl. Vor dem Start eines Projekts sollte jedoch die Anforderungsanzahl in Betracht gezogen werden, da sich die meisten Anforderungen noch im Ausgangsstatus befinden. Zudem lassen sich analog zum Projekt wieder die Versionen auswerten, welche sich in einem gewissen Zustand befinden.

Änderungsrate

Die Änderungsrate einer Anforderung lässt sich sowohl auf eine einzelne Anforderung und deren Versionen, als auch auf ein gesamtes Projekt darstellen und ist eine der bedeutendsten Kennzahlen im RE (SOPHISTen, 2021). Die Änderungsrate stellt dabei den Anteil der Anforderungen dar, die sich in einem Zeitraum geändert haben (Bühne & Hermann, 2015). Um daraus Rückschlüsse auf das Projekt zu ziehen, kann dieser Anteil ins Verhältnis zum Gesamtumfang eines Projekts gesetzt werden. Dadurch kann abgeleitet werden, wie robust ein Projekt definiert wurde (Ebert, 2019). Eine außergewöhnlich hohe Änderungsrate kann ein Zeichen dafür sein, dass die Analyse und Spezifikation nicht sauber abgeschlossen wurde. Bei einer zu niedrigen Rate kann eine zu geringe Absprache mit dem Kunden der Grund sein. Eine Änderungsrate von 1-5% pro Monat wird als angemessen angesehen. Zudem ist zu vermerken, dass die Änderungsrate mit zunehmender Laufzeit eines Projekts abnehmen sollte. Die Änderungsrate lässt sich auch auf einzelne Anforderungen anwenden. Eine besonders hohe Änderungsrate sollte hierbei die Anforderung für genauere Betrachtungen in den Fokus rücken, da an dieser Stelle eventuell noch Unklarheiten vorherrschen könnten.

Fehler in Anforderungen

Eine weitere Kennzahl die ausgewertet werden sollte, ist die Kennzahl "Fehler in Anforderungen" (Ebert, 2019). Hierbei wird am Ende eines Projekts jeder Fehler

in einer Anforderung gezählt. Hinsichtlich des Umfangs eines Projekts lässt sich daraus die Qualität in einem Projekt ableiten. Dieser Prozess ist jedoch durch die fehlende Einordnung, wann ein Fehler aufgetreten ist, nur schwer durch Tools zu automatisieren.

Anzahl der Beteiligten am Prozess

Die Anzahl der Beteiligten am Prozess beschreibt die Anzahl der Personen welche im Projekt an den Anforderungen mitgewirkt haben (SOPHISTen, 2021). Zu viele Autoren können hierbei zu erschwerter Verständlichkeit einer Anforderung führen (Ebert, 2019).

2.2.3 Nutzen von Versionierung

Da die Gründe für das Ändern einer Anforderung vielseitig sein können, ist eine saubere Versionierung für einen erfolgreichen Projektabschluss essenziell. Die Argumente für das Einsetzen einer Versionierung lassen hierbei nach Ebert, 2019 in folgende drei Gruppen untergliedern:

- Unklare ursprüngliche Anforderungen
- Falsche Annahmen und Unsicherheiten
- Sich ändernde Kundenanforderungen oder Marktbedürfnisse

Eine kontrollierte Verwaltung der Anforderungen stellt sicher, dass die Änderungen nicht das Projekt beherrschen. Wenn das Steuern des Projekts über dessen Anforderungen nicht möglich ist, droht gar ein Scheitern eines Projekts. Durch Versionierung und Änderungsmanagement kann dieses Problem eingedämmt werden. So lassen sich während des Projekts alle Änderungen für alle beteiligten Personen exakt nachvollziehen. So kann beispielsweise auch ein Softwareentwickler, welcher bei der Erstellung der Anforderung nicht beteiligt war und ohne Versionierung den eigentlichen Ursprung der Anforderung nicht kennt, genau begreifen, was die ursprüngliche Intention der Anforderung war. Hiermit können eine Fehlinterpretation sowie zusätzliche Nacharbeiten vermieden werden.
Des Weiteren lassen sich durch Versionierung, wie in Abschnitt 2.2.2 erläutert, einige Kennzahlen zum Projekt ableiten, welche im Nachgang für Optimierungen in weiteren Projekten genutzt werden können.

2.3 Änderungskontrolle

Das folgende Kapitel beschreibt bereits bekannte Ansätze der Änderungskontrolle. Zum einen wird die Vorgehensweise bei den beiden Vorreitern *Git*[1] und *Sub-*

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

version[2] erläutert. Zum anderen wird die Versionskontrolle bei bekannten Tools zum Anforderungsmanagement betrachtet. Durch die Analyse dieser bereits bekannten Mechanismen lässt sich anschließend eine Lösung zum Speichern und Versionieren der Anforderungen ableiten.

2.3.1 Versionsverwaltungsansätze/-systeme

Gerade im Umfeld der Softwareentwicklung existieren bereits zwei bekannte Ansätze zum Verwalten von verschiedenen Versionen eines Programmcodes oder auch von Dokumenten. Um eine Übersicht über die Unterschiede zwischen verschiedenen Versionen zu bewahren, kommen Tools zur Versions- und Änderungskontrolle zum Einsatz. Die beiden gängigsten Tools sind hierbei *Git* und *Subversion*, welche grundlegend das gleiche Ziel verfolgen, dabei jedoch unterschiedlich vorgehen. Im Gegensatz zum Versionieren von Anforderungen liegt beim Versionieren von Programmcode das Augenmerk eher darauf, dass Änderungen später noch nachvollzogen und mit anderen Entwicklern geteilt werden können (Blischak et al., 2016). Die Vollständigkeit und die Auswertung dieser Änderungen spielt hierbei eher eine untergeordnete Rolle.
Um die Unterschiede zwischen den einzelnen Änderungen, welche bei beiden Systemen über einen Commit festgehalten und mit einer Kommentarnachricht versehen werden, darzustellen, müssen bei beiden Herangehensweisen die Änderungen in Relation zueinander gesetzt und abgespeichert werden (Blischak et al., 2016). Diese beiden unterschiedlichen Ansätze werden nun genauer erläutert. Grundsätzlich unterscheiden sich *Git* und *Subversion* darin, dass es sich bei *Git* um ein verteiltes System und bei *Subversion* um ein zentralisiertes System handelt (Chacon & Straub, 2014). Zudem unterscheiden sich beide Systeme in der Art Änderungen zu speichern. Dieses Vorgehen ist auch bei der Versionierung von Anforderungen relavant und wird deshalb im folgenden genauer betrachtet.

Subversion

Subversion speichert die Information zu Änderungen als eine Liste von dateibasierten Änderungen (Chacon & Straub, 2014). Wie in Abbildung 2.5 dargestellt werden in jeder Version nur die Änderungen zur Vorgängerversion abgespeichert. Das bedeutet, dass eine Kette an Veränderungen gespeichert wird. Basierend auf der Ausgangsdatei lässt sich so der aktuelle Stand der Datei ableiten.
In Abbildung 2.5 wird angenommen, dass sich in Version 2 die Dateien *File A* und *File C* geändert haben. Beim Speichern dieser Änderungen werden nun in $\Delta 1$ nur die Informationen zur Änderung für *File A* und *File C* gespeichert, nicht aber die ganze Datei. Angenommen von Version 2 auf Version 3 wird *File C* geändert, so werden in $\Delta 2$ analog zu Version 2 nur die Änderungen in *File C* gespeichert.

[2] https://subversion.apache.org/

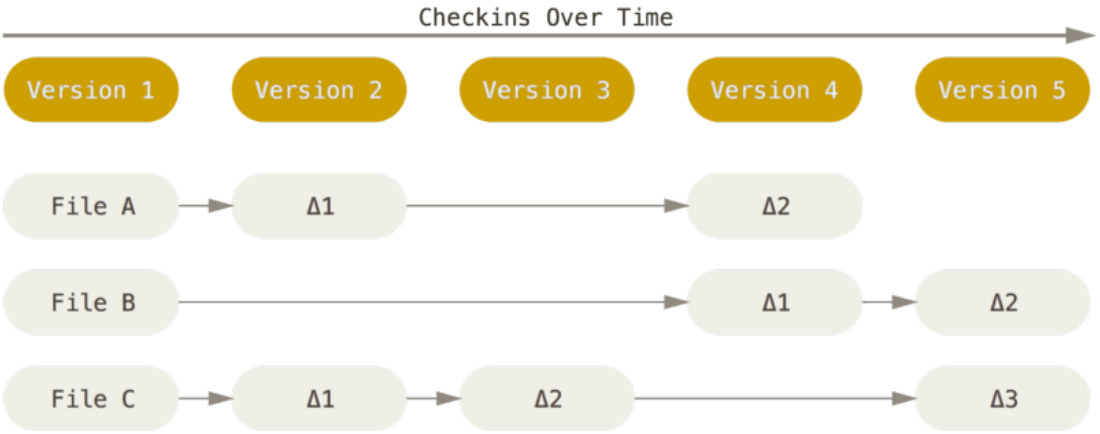


Abbildung 2.5: Speichern der Änderungen zur Ursprungsdatei (Chacon & Straub, 2014)

Dieses Vorgehen erweist sich vor allem für größere Binärdateien als effizient.

GIT

Anders als bei *Subversion* werden die Informationen nicht als eine Liste von dateibasierten Änderungen gespeichert, sondern werden eher als eine Reihe an Schnappschüssen betrachtet (Chacon & Straub, 2014). Wie in Abbildung 2.6 zu sehen, werden die Daten ähnlich eines Stapels an Schnappschüssen gespeichert. Bei jedem Commit wird ein Schnappschuss aller Dateien angelegt und der Verweis auf diesen gespeichert. Bei unveränderten Dateien wird hierbei jedoch anstatt einer vollständigen Kopie nur ein Verweis auf die Vorgängerversion angelegt. Dies sorgt für eine höhere Effizienz im Vergleich zum Kopieren aller Dateien. Dieses Vorgehen ist jedoch bei größeren Binärdateien nachteilig. Im Gegensatz zu *Subversion* wird ‚anstatt die Änderungen zu speichern, eine vollständige Kopie der geänderten Binärdatei angelegt. Dies sorgt für einen erhöhten Speicherbedarf.
Analog zu Abbildung 2.5 wird in Abbildung 2.6 eine Änderung der Dateien *File A* und *File C* angenommen. *GIT* speichert nun jedoch nicht die Änderungen, sondern erzeugt eine Kopie der Dateien *File A1* und *File C1*. Wie bereits erwähnt, wird bei unveränderten Dateien aus Effizienzgründen nur ein Verweis auf die Vorgängerversion angelegt.

Einordnung

Im Bezug auf die Versionsverwaltung von Anforderungen wären somit beide Ansätze denkbar und umsetzbar. Da die Datenmengen beim Versionieren von Anforderungen im Vergleich zu großen Binärdateien sehr gering sind, kann dieser Nachteil im folgenden Verlauf der Arbeit vernachlässigt werden. So dienen beide vorgestellten Ansätze als Grundlage für das Design zum Speichern der Änderun-

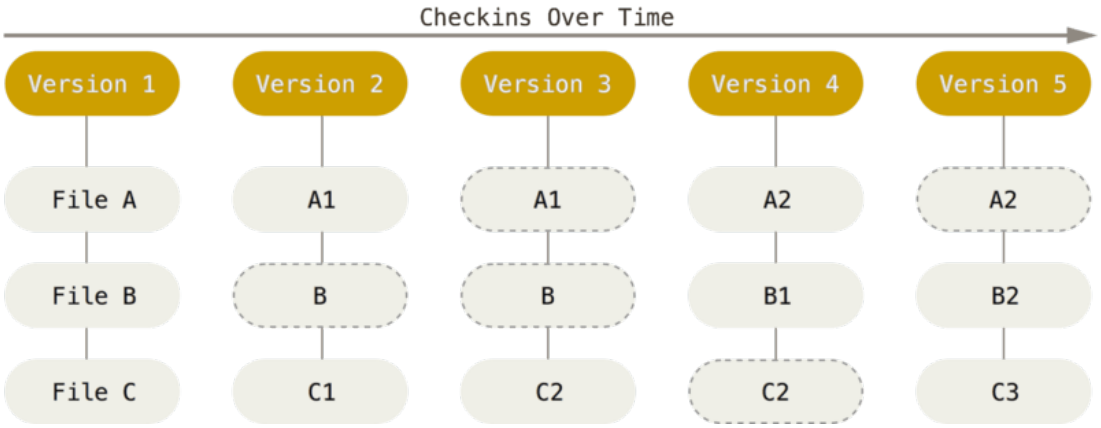


Abbildung 2.6: Speichern der Historie als Reihe von Schnappschüssen (Chacon & Straub, 2014)

gen in Abschnitt 4.1.1.

2.3.2 Tools zum Anforderungsmanagement

Analog zu den bestehenden Tools zur Versionsverwaltung werden in diesem Abschnitt gängige Tools zum Anforderungsmanagement und insbesondere deren Versionshistorie betrachtet. Viele der bekannten Tools wie etwa *DOORS*[3], *Jira*[4] oder *Redmine*[5] stellen keine vollständige Versionshistorie zur Verfügung, sondern dokumentieren ähnlich wie *Subversion* nur die Änderungen im Vergleich zu Vorgängerversion (Sarkan et al., 2011). Beispielhaft wir hier in Abbildung 2.7 die Versionshistorie bei *Jira* dargestellt. Dies sorgt zwar für eine vollständige Nachverfolgbarkeit der Anforderungen, erschwert jedoch die Auswertung oder das Zurücksetzen einer auf eine Vorgängerversion.

[3]https://www.ibm.com/docs/de/ermd/9.6.1?topic=overview-rational-doors
[4]https://www.atlassian.com/de/software/jira
[5]https://redmine.org/

Abbildung 2.7: Jira Versionshistorie

3 Anforderungen

Im folgenden Abschnitt werden die zu Beginn des Projekts definierten Anforderungen vorgestellt:

Req 1.0 - Der Specification Editor muss dem Nutzer die Möglichkeit bieten eine Versionshistorie anzuzeigen

Dem Nutzer muss eine graphische Oberfläche zur Verfügung stehen, über welche er den gesamten Versionsverlauf einer Anforderung einsehen kann. Zudem soll sichtbar sein, welche Änderungen im Vergleich zur Vorgängerversion zu welchem Zeitpunkt und von welchem Nutzer durchgeführt wurden.

Req 1.1 - Die Versionshistorie sollte Änderungen zur Vorgängerversion darzustellen

Um Veränderungen leichter zu erkennen, sollte die Versionshistorie die Möglichkeit bieten, nur die geänderten Attribute im Vergleich zur Vorgängerversion darzustellen.

Req 1.2 - Die Versionshistorie sollte dem Nutzer die Möglichkeit bieten Kennzahlen zu den Versionen der Anforderungen darzustellen

Das Darstellen verschiedener Kennzahlen zur Versionierung einer Anforderung soll dem Nutzer das Auswerten und die Analyse von Änderungen erleichtern. Als Grundlage sollen hier die in Abschnitt 2.2.2 dargestellten Kennzahlen aus der Literatur dienen.

Req 2.0 - Die Versionshistorie muss gegen nachträgliche Änderungen geschützt sein

Die Versionshistorie muss gegenüber Veränderungen durch den Nutzer geschützt sein, um Änderungen verfolgbar, nachvollziehbar und konsistent zu versionieren.

Req 2.1 - Die Versionshistorie soll einem Administrator die Möglichkeit geben Anforderungen zu löschen

Administratoren sollen die Möglichkeit haben, fälschlich angelegte Anforderungen dennoch vollständig zu löschen.

Req 2.2 - Der Specification Editor sollte dem Nutzer die Möglichkeit bieten eine Anforderung zu archivieren

Anstatt eine Anforderung vollständig zu löschen, soll ein normaler Nutzer in der Lage sein, Anforderungen zu archivieren, falls diese nicht mehr benötigt werden.

Req 3.0 - Anforderungen müssen dem Nutzer die Möglichkeit bieten eine neue Version anzulegen

Beim Ändern einer Anforderung, soll der Nutzer die Möglichkeit haben eine neue Version der Anforderung anzulegen.

Req 3.1 - Anforderungen müssen dem Nutzer die Möglichkeit bieten beim Abspeichern einen Kommentar zum Änderungsgrund anzulegen

Req 3.2 - Die Versionshistorie sollte dem Nutzer die Möglichkeit bieten den Stand einer älteren Version wiederherzustellen

Durch das Anlegen von neuen Versionen, soll dem Nutzer die Möglichkeit gegeben werden den Stand einer älteren Version wiederherzustellen.

Req 3.3 - Der Specification Editor sollte dem Nutzer die Möglichkeit bieten zu entscheiden, ob eine neue Version angelegt wird

Der Nutzer soll entscheiden können, ob das anlegen einer neuen Version nötig ist. So soll beispielsweise das verbessern eines Rechtschreibfehlers nicht in der Versionshistorie erscheinen.

Req 4.0 - Der Specification Editor werden dem Nutzer die Möglichkeit bieten einen Export durchzuführen

Anforderungen werden durch den Nutzer mit dem Versionsverlauf exportiert werden können. So soll die Verknüpfung und Wiederverwendbarkeit mit anderen Tools sichergestellt werden.

Req 5.0 - Der Specification Editor wird dem Nutzer die Möglichkeit bieten Anforderungen zu filtern

Der Nutzer erhält so die Möglichkeit bestimmte Anforderungen ein- beziehungsweise auszublenden. So besteht beispielsweise die Möglichkeit bereits archivierte Anforderungen dennoch im Specification Editor anzuzeigen.

Req 6.0 - Der Specification Editor wird dem Nutzer die Möglichkeit bieten Anforderungen zu sortieren

4 Aufbau und Design

Der folgende Abschnitt beschreibt die Herangehensweise an verschiedene Problemstellungen und Designfragen. Hierbei werden verschiedene Möglichkeiten hergeleitet, welche anschließend eingeordnet werden.

4.1 Ansätze zum Speichern von Versionen

Die erste Fragestellung bezieht sich auf das Anlegen einer Versionshistorie. Dabei gibt es im Grundlegenden zwei verschiedene Ansätze, die es abzuwägen gilt. Hierbei kommen zum einen das Speichern der Veränderungen, zum anderen das Speichern einer vollständigen Kopie der alten Anforderungen in Frage.

4.1.1 Art des Speicherns von Änderungen

Im Folgenden werden verschiedene Ansätze der Datenhaltung und des Speicherns der Änderungen erörtert. Dabei gilt vor allem es nach Speicheraufwand und Nutzbarkeit der Daten abzuwägen.

Variante S1: Speichern der Veränderungen

Die erste Möglichkeit ist das Speichern der Veränderungen zur Vorgängerversion. Der Vorteil dieser Methode liegt darin, dass weniger bis keine redundanten Daten gespeichert werden müssen, was für einen geringeren Bedarf an Speicherplatz sorgt. Um diesen Ansatz umzusetzen, könnten die Änderungen beispielsweise als eigenes Attribut der Anforderung als Liste angelegt werden. Es würde sich dabei anbieten für die Änderungen eine eigene Hilfsklasse anzulegen. Abbildung 4.1 stellt einen beispielhaften Aufbau der Basisklasse `Requirement` und der Hilfsklasse `RequirementChange` dar. `Requirement` hält hierbei neben den aktuellen Daten der Anforderungen eine Liste an Änderungen. Somit ist man in der Lage, basierend auf der ursprünglichen Anforderung, die neue Anforderung zu berechnen. Dieses Vorgehen ist ähnlich zu dem in Abschnitt 2.3.1 von *Subversion* und vermeidet vor allem eine redundante Datenhaltung ungeänderter Attribute. Die

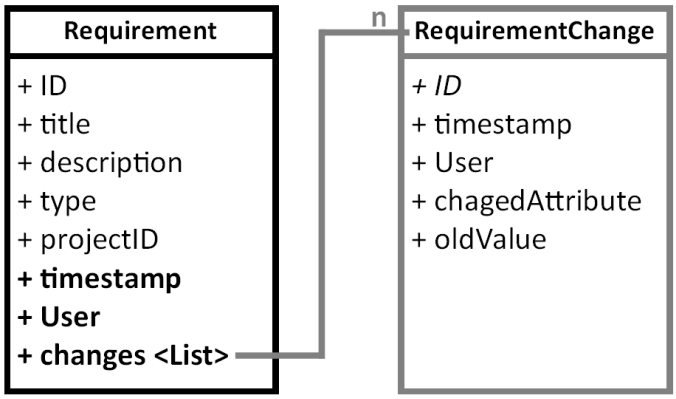


Abbildung 4.1: Speichern der Änderungen

Datenhaltung, insbesondere bei mehreren geänderten Attributen, und die Aufbereitung gestalteten sich dadurch jedoch erheblich komplexer als beim Anlegen einer vollständigen Kopie, was im nächsten Abschnitt genauer ausgeführt wird. Auch das Durchsuchen und Filtern der Daten gestaltet sich aufwändiger, da die Datenbank nicht direkt durchsucht werden kann, sondern für jede Abfrage auch die Attribute durchsucht oder gefiltert werden müssen.

Variante S2.1: Speichern einer Kopie als Attribut

Die zweite Möglichkeit ist das Speichern einer Kopie der Vorgängerversion, welche dann als Attribut in der Anforderung gespeichert wird. Die Änderungen werden hierbei ähnlich wie bei *Variante S1* als Attribut in der eigentlichen Anforderung gespeichert. Anders als bei *Variante S1* werden jedoch nicht die Änderungen im Attribut gespeichert, sondern, wie in Abbildung 4.2 dargestellt, eine vollständige Kopie der alten Anforderung. Dieses Vorgehen hat den Vorteil, dass im Gegensatz zu *Variante S1* keine Hilfsklasse benötigt wird Zudem bleibt aus Gründen der Konsistenz die gesamte alte Version erhalten. Je nach Implementierung gestaltet sich auch das Löschen einfach, da nur diese eine Anforderung gelöscht werden muss und auf das Löschen der Hilfsklassen verzichtet werden kann. Dieser Ansatz bringt jedoch ebenfalls die gleichen Nachteile bei der Datenhaltung, dem Durchsuchen und dem Filtern wie bereits bei *Variante S1* mit sich.

Variante S2.2: Speichern einer Kopie als neue Anforderung

Bei dieser Variante wird, wie von SOPHISTen, 2021 empfohlen, eine Kopie der alten Anforderung angelegt, auf welche dann über eine eindeutige `ID` verlinkt werden kann. Die neu erzeugte Anforderung erhält dann eine neue `ID` und hält die alte `ID` als Attribut. So entsteht eine lineare Kette an Anforderungen, durch welche dann bei Bedarf iteriert werden kann. In Abbildung 4.3 wird diese Ket-

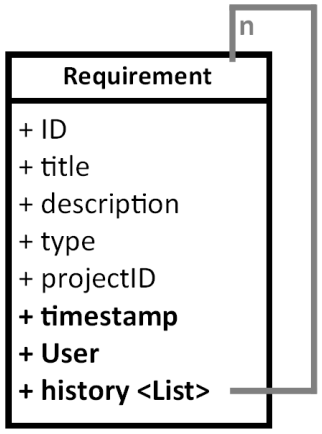


Abbildung 4.2: Speichern einer Kopie als Attribut

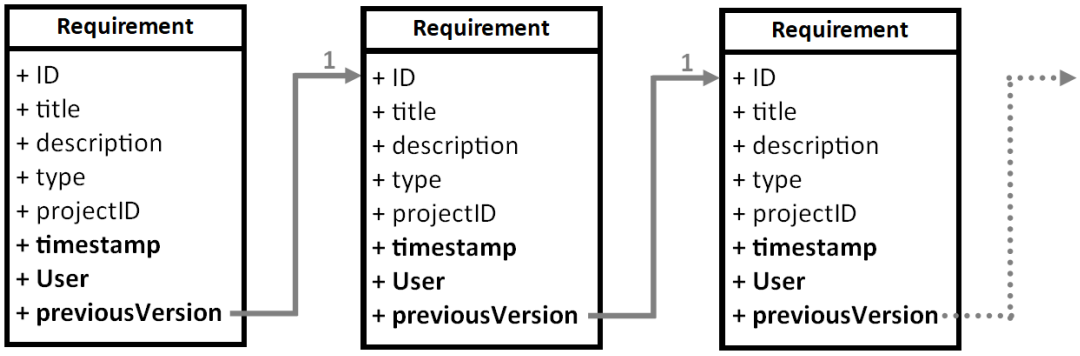


Abbildung 4.3: Speichern einer Kopie als neue Anforderung

te genauer beschrieben. Das Attribut `previousVersion` hält dabei die `ID` der Vorgängerversion, was zu der in Abschnitt 2.2.1 beschriebenen linearen Kette an Versionen führt. Dieses Vorgehen hat jedoch den Nachteil, dass für jede Änderung eine neue Anforderung angelegt wird, was für einen erhöhten Speicherbedarf sorgt. Das Suchen oder Filtern in den Anforderungen gestaltet sich durch die Datenhaltung in einer Tabelle jedoch als äußerst einfach, da sich alle Versionen durch eine einzige Abfrage filtern oder sortieren lassen.

Ergebnis

Wie von SOPHISTen, 2021 beschrieben, eignet sich *Variante S2.2* am Besten für eine Versionierung von Anforderungen. Da beim Versionieren von Anforderungen die Vollständigkeit und Nachverfolgbarkeit von großer Bedeutung ist, bietet es sich an, eine vollständige Kopie der Anforderung anzulegen. Dies stellt zwar einen höheren Bedarf an Speicherplatz dar, da einige Attribute redundant in der Datenbank gehalten werden. Dieser erhöhte Bedarf ist jedoch verglichen mit der

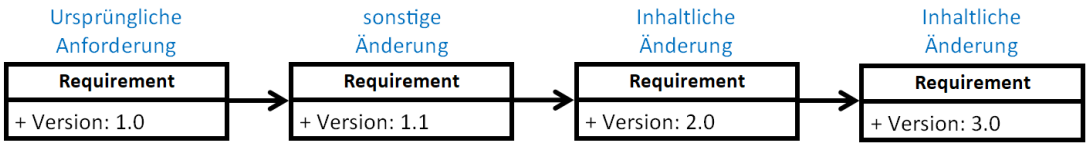


Abbildung 4.4: Speichern einer Versionsnummer mit Inkrementen

alternativen *Variante S1* relativ gering, da bei *Variante S1* ebenfalls für jede Version ein neues Objekt angelegt wird. Dabei handelt es sich zwar nicht um eine vollständige Anforderung, jedoch werden beispielsweise in Darstellung 4.1 ebenfalls bereits mindestens 5 Attribute gespeichert. Der Overhead für eine vollständige Kopie bleibt somit im vertretbaren Rahmen. Die Vorteile einer vollständigen Kopie liegen neben der Vollständigkeit zum einen im einfachen Filtern und Sortieren der einzelnen Attribute der Anforderung, zum anderen lassen sich so auch problemlos und effizient die Vorgängerversionen darstellen und auch wieder zurücksetzten. Da die Version vollständig vorliegt, müssen keine Berechnungen der Änderungen basierend auf anderen Versionen vorgenommen werden. Die Daten liegen als vollständig nutzbare Anforderung in der Datenbank vor, was die Handhabung sehr klar gestaltet und die spätere Wartung erleichtert.

4.1.2 Anlegen einer neuen Version

Dieser Abschnitt befasst sich mit der Fragestellung, wie und wann eine neue Version angelegt werden soll. Dabei gilt es zum einen die Vollständigkeit und Nachverfolgung zu bewahren und zum anderen den Mehraufwand und die Fehlerquote des Nutzers so gering wie möglich zu halten. Wie in Abschnitt 2.2.1 dargelegt, sollte jede Änderung an einer Anforderung dokumentiert werden (Bühne & Hermann, 2015).

Variante Z1: Anlegen von Inkrementen

Um die Art der Änderung festzuhalten und zu differenzieren, gibt es mehrere Möglichkeiten. So kann man die Versionsnummer in Inkremente aufbauen. Wie in Abbildung 4.4 zu sehen, führt eine inhaltliche Änderung zu einer Erhöhung der Versionsnummer. Bei einer geringfügigen Änderung, wie beispielsweise einem Rechtschreibfehler erhöht sich nur das Inkrement. So erhalten alle inhaltlichen Änderungen eine "glatte" Versionsnummer, wodurch eine Differenzierung innerhalb der Versionshistorie zu kleineren unwichtigeren Änderungen leicht möglich ist (Bühne & Hermann, 2015). Dieses Vorgehen bedarf jedoch zum einen das Verwalten der Versionsnummern, zum anderen muss der Nutzer eine Entscheidung treffen, wann wirklich eine neue Version angelegt werden soll oder wann nur das Inkrement erhöht werden soll. Dies stellt eine potentielle Fehlerquelle dar und kann zu einer inkonsistenten Versionshistorie führen, welche es zu vermeiden gilt.

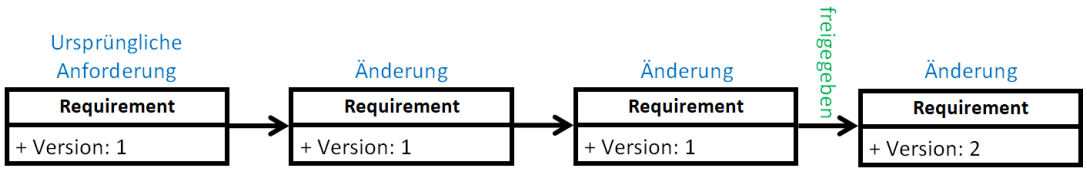


Abbildung 4.5: Anlegen einer neuen Version nach Freigabe

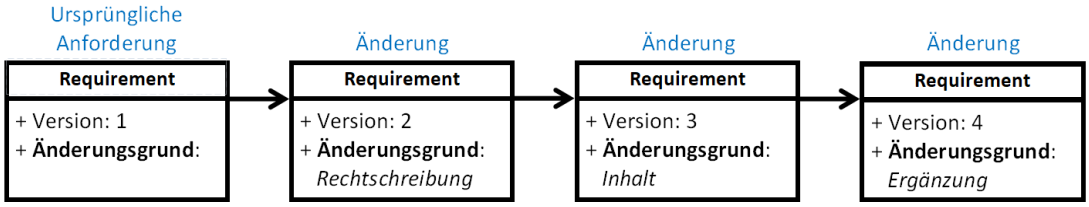


Abbildung 4.6: Anlegen einer neuen Version mit Änderungsgrund

Variante Z2: Freigabe führt zu neuer Version

Alternativ zu der Versionierung mit Inkrementen kann auch mit einem Freigabemachnismus gearbeitet. So werden Änderungen zwar abgespeichert, müssen jedoch erst später freigegeben werden. Erst das Freigeben von Änderungen führt zum Anlegen einer neuen Version. Dieses Vorgehen wird in Abbildung 4.5 veranschaulicht. Diese Variante hat eine deutlich geringere Versionshistorie zur Folge, zieht jedoch den Nachteil mit sich, dass eine zusätzliche Person involviert ist, welche die Änderungen dann erst freigeben muss. Des Weiteren werden so nicht wie in der Literatur gefordert alle Änderungen festgehalten, was die Vollständigkeit und Nachverfolgbarkeit untergräbt.

Variante Z3: Anlegen eines Änderungsgrunds

Eine weitere Variante zum Anlegen einer neuen Version stellt das Anlegen eines zusätzlichen Attributes dar, welches den Änderungsgrund hält. Durch eine Vorauswahl an möglichen Änderungsgründen lässt sich die Versionshistorie wie beim Verwenden von Inkrementen einfach filtern. Des Weiteren kann eine Vorauswahl an möglichen Änderungsgründen den Nutzer bei der Entscheidungsfindung unterstützen, wodurch der Mehraufwand und die Fehlerquote sehr gering bleibt. Jede Änderung führt dann zu einer neuen Version der Anforderung. Dieses Vorgehen ist in Abbildung 4.6 dargestellt und bewahrt, wie in der Literatur gefordert, die Vollständigkeit und Nachverfolgbarkeit der Anforderungen.

Ergebnis

Um eine möglichst einfache Nutzung, eine geringe Fehlerquote und einen möglichst geringen Mehraufwand für den Nutzer zu ermöglichen, bietet sich *Variante*

Z3 an, welche einen Änderungsgrund vorsieht. Ergänzend dazu kann ein optionales Kommentarfeld für eine detaillierte Beschreibung des Änderungsgrundes einen Mehrwert bieten. Wie bereits erörtert führt *Variante Z1* zu einer erhöhten Fehlerquote durch eine ungenaue Abgrenzung und *Variante Z2* sorgt für einen Mehraufwand durch eine zusätzliche Person und untergräbt zudem die Vollständigkeit aller Änderungen, weshalb *Variante Z3* einen guten Kompromiss aus Nutzbarkeit und Vollständigkeit darstellt.

4.2 Kennzahlen zum Auswerten der Versionierung

Wie in Abschnitt 2.2.2 beschrieben, bieten sich mehrere Kennzahlen zum Auswerten der Versionierung an. Denkbar wäre eine Analyse über das gesamte Projekt durchzuführen. Des Weiteren sollen auch einzelne Anforderungen ausgewertet werden können. Diese lassen sich dann mit dem Durchschnitt eines Projekts vergleichen, wodurch sich größere Abweichungen erkennen lassen.

4.2.1 Kennzahlen im Projekt

Folgende Kennzahlen lassen sich in Bezug auf das gesamte Projekt auswerten:

Anfoderungsanzahl

Die wohl am einfachsten auszuwertende Kennzahl stellt die *Anfoderungsanzahl* dar. Hierbei wird die Summe aller aktiven Anforderungen gezählt.

Änderungsrate

Durch das Anlegen einer vollständigen Kopie lässt sich diese Zahl auch über alle Versionen auswerten. Im Verhältnis zur zuvor berechneten *Anfoderungsanzahl* lässt sich so die *Änderungsrate* über das gesamte Projekt berechnen. Die *Änderungsrate* lässt sich auch auf die letzte Woche und den letzten Monat auswerten. Hierbei wird die Anzahl aller Anforderungen in Relation zu den Versionen gesetzt, welche einen `Timestamp` im besagten Zeitraum haben.

Anzahl der Beteiligten im Prozess

Eine weitere in der Literatur genannte Kennzahl stellt die *Anzahl der Beteiligten im Prozess* dar. So wird für jede Version die Zahl der Autoren gezählt. Auch diese Anzahl lässt sich wiederum mit der *Anfoderungsanzahl* ins Verhältnis setzen, was die Anzahl der an einer Anforderung mitwirkenden Personen darstellt.

Anforderungsstatus

Je nach Projekt kann auch der *Anforderungsstatus* ausgewertet werden. Hierfür wäre ein weiteres Attribut `Status` nötig, welches sich dann auch über dessen Summe auswerten lässt. Analog zu den bereits genannten Kennzahlen lässt sich auch hier eine Quote in Relation zu der Gesamtzahl an Anforderungen berechnen.

Fehler im Projekt

Wie bereits in Abschnitt 2.2.2 erläutert, wären die *Fehler im Projekt* eine weitere aussagekräftige Kennzahl. Diese Kennzahl ist jedoch erst gegen Ende eines Projekts von Bedeutung und nur durch erheblichen Mehraufwand des Nutzers auszuwerten, weshalb in diesem Projekt auf das Auswerten der *Fehler im Projekt* verzichtet wird.

4.2.2 Kennzahlen einer Anforderung

Neben Kennzahlen über das gesamte Projekt, lassen sich einige Kennzahlen auch für eine einzelne Anforderung und deren Versionen berechnen.

Versionsanzahl

Die wohl am einfachsten auszuwertende Kennzahl stellt ähnlich zur *Anfoderungsanzahl* im Projekt die *Anzahl der Versionen* dar. Hierbei wird die Summe aller Versionen einer Anforderungen gezählt.

Änderungsrate

So lässt sich die *Änderungsrate* für einzelne Anforderungen als Änderungen pro Tag berechnen. Denkbar ist hier eine Zeitspanne seit der Erstellung oder auch der letzten Woche oder Monate. Dies schafft eine Vergleichbarkeit mit anderen Anforderungen.

Anzahl der Beteiligten im Prozess

Analog zu den Kennzahlen in einem Projekt lassen sich auch die *Anzahl der Beteiligten* über die Zahl der verschiedenen Autoren berechnen. Auch hier kann wieder eine Quote in Bezug auf die Anzahl der Änderungen berechnet werden.

Änderungen pro Code

Eine bisher nicht berücksichtigte Zahl können die *Änderungen pro Code* darstellen. Durch die Verknüpfung der Anforderungen mit den Codes aus dem Codesystem, welches mittels QDAcity erstellt wird, lassen sich auch die Änderungen pro

Code ableiten. Dies bietet einen Mehrwert im Vergleich zu anderen Tools, da sich so abschätzen lässt, wie viele Änderungen pro Themengebiet vorliegen. Daraus lässt sich somit eine Änderungsrate pro Themengebiet ableiten.

4.3 Aufbau der GUI

Um die Versionierung der Anforderungen übersichtlich und einfach bedienbar darzustellen sind mehrere Anpassungen an der Nutzeroberfläche zu betrachten. Zum einen muss das aktuelle GUI um zusätzliche Attribute erweitert werden. Zum anderen bietet es sich an, für die Versionshistorie ein eigenes Fenster anzulegen, welches sämtliche Informationen zu den Versionen einer Anforderungen bietet und Statistiken zur Versionierung darstellt.

4.3.1 Erweiterungen der bestehenden GUI

Wie in Abbildung 2.1 zu Beginn dargestellt, existiert bereits eine Grundlegende Anforderungsverwaltung im Tool QDAcity. In der bestehenden GUI wurde mit fest implementierten Attributen gearbeitet, wodurch eine flexible Erweiterung der Attribute erschwert wird. Um im Laufe der Entwicklung die Anforderungen flexibel erweitern zu können, empfiehlt es sich hier die Anforderung auf einen einheitlichen tabellarischen Aufbau umzustrukturieren. Es bietet sich also an, wie in Abbildung 4.7 zu sehen, für jedes Attribut einen Titel zu vergeben und anschließend den Wert dieses Attributs darzustellen. Dies sorgt für einen einheitlichen Aufbau und eine flexible Erweiterungsmöglichkeit um neue Attribute. Analog zum *Anforderungstyp* lässt sich so die GUI beispielsweise um *Autor*, *Zeitpunkt der letzten Änderung* oder auch einen *Status* erweitern. Wie in Abbildung 4.7 bereits zu sehen, wurden *Autor* und *Zeitpunkt* in die Titelzeile übernommen. So bleiben in der tabellarischen Darstellung nur Attribute, die im Zusammenhang mit inhaltlichen Informationen stehen. Es wäre auch denkbar an dieser Stelle, den *Änderungsgrund* oder einen *Kommentar* darzustellen. Da es sich hierbei jedoch ebenfalls um Attribute bezüglich der Versionierung handelt, welche nicht im direkten Zusammenhang mit dem Inhalt der Anforderung stehen, wurde hierbei ebenfalls darauf verzichtet. Eine genauere Darstellung dieser Attribute wird in Abschnitt 4.3.2 beschrieben.
Neben den Attributen bedarf es auch einer Anpassung der bestehenden Buttons. Wie in 2.2.1 beschrieben, sollen Anforderungen aus Gründen der Verfolgbarkeit und Vollständigkeit nicht von jedem Nutzer gelöscht werden können. Demnach wird eine zusätzlicher Button *Archivieren* benötigt, welcher es dem Nutzer ermöglicht, eine Anforderung zu archivieren anstatt zu löschen. Handelt es sich um einen normalen Nutzer, soll dieser Button anstelle des bestehenden Buttons *Löschen* erscheinen. Für berechtigte Nutzer sollen jedoch beide Möglichkeiten zur Verfügung stehen.

Button/Zustand	Eingeklappt	Details	In Bearbeitung
Versionshistorie	X	X	
Archivieren	X	X	
Löschen	X	X	
Bearbeiten		X	X
Speichern			X
Anzahl	*3*	*4*	*2*

Tabelle 4.1: Darstellung der Buttons nach Zustand

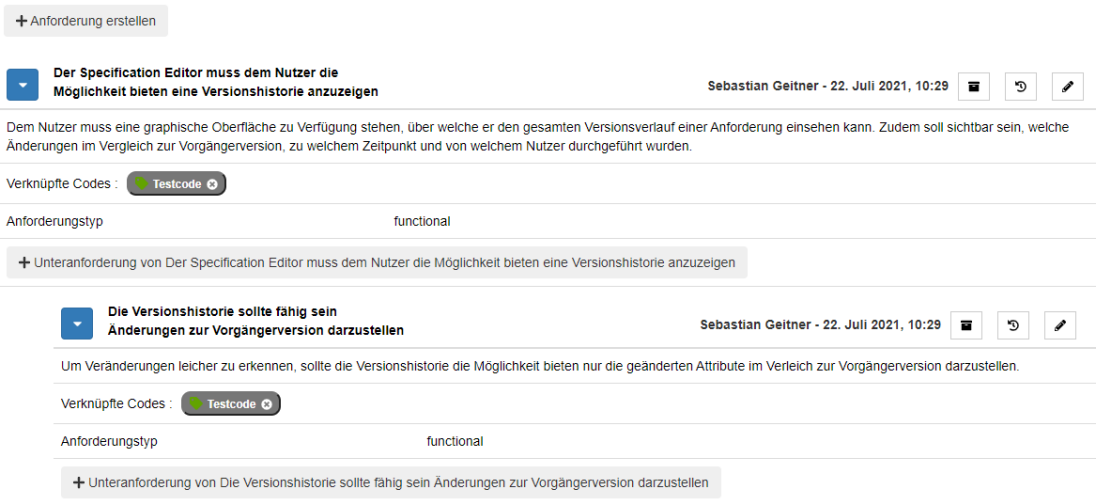


Abbildung 4.7: Screenshot QDAcity Umstrukturierung GUI

Zusätzlich zum *Archivieren*-Button soll ein weiterer Button dem Nutzer die Möglichkeit bieten, die Versionshistorie einer Anforderung einblenden zu können. Eine Darstellung aller Buttons zu jedem Zeitpunkt würde die Titelzeile jedoch schnell unübersichtlich gestalten. Deshalb gilt es abzuwägen, welche Buttons zu welchem Zeitpunkt dargestellt werden sollen. Tabelle 4.1 bietet eine Übersicht, in welchem Zustand die Buttons dargestellt werden sollen. Generell gilt zu erwähnen, dass weniger Buttons für eine bessere Übersichtlichkeit sorgen. Zudem sollte sich die Darstellung konsistent gestalten, was sich beispielsweise in einer einheitlichen Reihenfolge der Buttons widerspiegeln kann.

4.3.2 Ansätze zum Darstellen einer Versionshistorie

Für die Darstellung der Versionshistorie sind mehrere Ansätze denkbar. Um die Versionshistorie möglichst differenziert von der aktuellen Version der Anforderung zu gestalten, bietet sich ein eigenes Fenster an, welches über den zuvor erwähnten Button *Versionshistorie* aufgerufen werden kann. Im Folgenden werden sechs Ansätze zur Darstellung dieser Historie in einem eigenen Fenster vorgestellt.

Abbildung 4.8: Mockup GUI - Darstellen der vollständigen Kopie

Variante VH1: Darstellen der vollständigen Kopie

Analog zu den Anforderungen lässt sich, wie in Abbildung 4.8 zu sehen ist, auch jede Version einer Anforderung mit allen Attributen darstellen. Diese Variante lässt sich am Einfachsten implementieren und bedarf kaum Anpassungen, da beim Speichern der Änderungen einer Anforderung eine vollständige Kopie angelegt wird. Dadurch liegen analog zu den bisherigen Anforderungen für jede Version alle Attribute in der Datenbank vor. Den einzigen Unterschied stellen die Buttons dar. Da weder das Löschen noch das Bearbeiten einer Version möglich sein soll, entfallen sämtliche bisher genutzte Buttons. Um die Funktion eines Reverts einer Anforderung zu realisieren, wird jedoch ein neuer Button *Revert* benötigt. Dieser lässt sich anstelle der bisherigen Buttons implementieren. Der Nachteil dieser Variante ist, dass sich für den Nutzer nur schwer nachvollziehen lässt, welche Attribute sich geändert haben.

Variante VH2: Hervorheben der geänderten Attribute

Um das Problem der schlechten Nachvollziehbarkeit der geänderten Attribute aus *Version VH1* zu beheben, werden in dieser Version zwar analog zu *Version VH1* die vollständigen Versionen der Anforderung dargestellt. Jedoch werden hier zusätzlich die Attribute, welche sich im Vergleich zur Vorgängerversion geändert haben, farblich hervorgehoben. So kann der Nutzer unproblematisch erkennen, welche Attribute sich geändert haben. Wird die Anforderung jedoch im Laufe der Zeit mit zusätzlichen Attributen ergänzt, kann auch diese Version schnell unübersichtlich hinsichtlich der getätigten von Änderungen werden.

Abbildung 4.9: Mockup GUI - Darstellen der Änderungen und eines generischen Titels

Variante VH3: Darstellen nur der geänderten Attribute

Im Gegensatz zu *Variante VH1* und *Variante VH2* werden in dieser Variante, wie in Abbildung 4.10 zu sehen, nur die Attribute dargestellt, welche sich im Vergleich zur Vorgängerversion geändert haben. Dies sorgt für eine sehr gute Übersichtlichkeit, da nur die relevanten Attribute dargestellt werden und auf redundante Informationen verzichtet werden kann. Durch diese Art der reduzierten Darstellung kann es jedoch zum Verlust an nötigen Informationen kommen, da der Nutzer nicht alle Attribute einsehen kann.

Variante VH4: Darstellen der Änderungen der Attribute

Diese Variante beschäftigt sich mit dem Darstellen der Änderungen der Attribute. Hierbei werden wie in *Variante VH3* nur die geänderten Attribute dargestellt. Zusätzlich zu den neuen Werten der Attributen, werden hier jedoch zudem, wie in Abbildung 4.9 zu sehen, die Werte der Vorgängerversion dargestellt. So sieht der Nutzer sowohl den alten, als auch den neuen Stand der Anforderung. Bei vielen Änderungen oder größeren Änderungen, beispielsweise einer Änderung der Beschreibung, kann dies jedoch ebenfalls schnell unübersichtlich werden.

Variante VH5: Darstellen des Änderungsgrunds im Titel

Wie in Abbildung 4.10 zu sehen, lässt sich bei den bisher vorgestellten Varianten der Änderungsgrund und der dazugehörige Kommentar nur in der Detailansicht einsehen. Zudem fehlt in *Variante VH3* der Titel gänzlich, da nur die geänderten Attribute dargestellt werden. Um dem Nutzer einen schnellen Überblick zu geben,

welche Änderung vorgenommern wurde, bietet es sich an, den Änderungsgrund bereits im Titel der Anforderung darzustellen und den Titel analog zu den Attributen darzustellen. So kann der Nutzer auch im eingeklappten Zustand einsehen, was geändert wurde. Dies vermeidet unnötige Schritte wie das Aufklappen einer Version, selbst wenn nur eine Korrektur eines Rechtschreibfehlers vorliegt.

Variante VH6: Generisches Darstellen der geänderten Attribute im Titel

Ähnlich zu *Variante VH5* soll im Titel der Version ersichtlich sein, warum sich die Anforderung geändert hat. Anstatt, dass der Nutzer jedoch einen Änderungsgrund angeben muss, soll hier aus den geänderten Attributen ein geeigneter Titel abgeleitet werden. Dieses Vorgehen wurde ebenfalls bereits in Abbildung 4.9 dargestellt. Ein möglicher Titel könnte dann wie folgt lauten: *Sebastian Geitner hat den Titel geändert*. Diese generische Option gestaltet sich jedoch bei Änderungen an mehreren Attributen als schwierig, wodurch der Titel sehr lang und unübersichtlich werden kann. Zudem wird in der Implementierung ein zusätzlicher Aufwand bei der Lokalisierung für viele Attribute und Sprachen von Nöten sein.

Ergebnis

Nach Abwägen der einzelnen Möglichkeiten, sowie deren Vor- und Nachteile, bietet sich eine hybride Mischung aus mehreren der vorgestellten Varianten an. Standardmäßig soll der Nutzer beim Ausklappen einer Version die geänderten Attribute angezeigt bekommen. So erhält der Nutzer eine schnelle Übersicht, was sich im Vergleich zur Vorgägngerversion geändert hat. Darüber hinaus bietet sich ein zusätzlicher Button an, welcher es dem Nutzer ermöglicht, sich die vollständige Anforderung anzeigen zu lassen. So gehen zum einen keine Informationen verloren, zum anderen werden die alten Werte einzelner Attribute in der aktuellen Version nicht benötigt, da sich diese in der vollständigen Ansicht der Vorgängerversion einblenden lassen. Dies vermeidet eine redundante Darstellung von älteren Versionen eins Attributs. Des Weiteren soll der Änderungsgrund im Titel aus *Variante VH5* übernommen werden. So erhält der Nutzer mit nur geringem Mehraufwand beim Speichern eine schnelle Übersicht bezüglich des Änderungsgrunds. Zudem wird durch einen Änderungsgrund die Option offen gehalten, die Anforderung später nach Änderungsgrund zu filtern und so unwichtige Änderungen wie Rechtschreibfehler ausblenden zu lassen. Dies kann gerade bei komplexen Projekten mit vielen Änderungen einen Mehrwert bieten.
Ergänzend zu der Versionshistorie bietet es sich an, die in Abschnitt 4.2.2 erörterten Kennzahlen ebenfalls im neuen Fenster *Versionshistorie* darzustellen. Auch hier empfiehlt sich die bereits bekannte tabellarische Darstellung. Ein Button *Information* kann dem Nutzer hier genauere Details bezüglich einzelner Metriken

Abbildung 4.10: Mockup GUI - Darstellen der Änderungen zur Vorgängerversion

liefern.

4.3.3 Anzeigen der geänderten Codings

Durch eine Verknüpfung der Anforderung mit den Codes aus dem Codesystem bietet das Tool QDAcity einen Mehrwert durch eine vollständige Verfolgbarkeit zurück zum Anforderungsdokument. Diese Verknüpfung lässt sich nun Nutzen, um die sich geänderten Codes und deren Codings darzustellen, welche sich seit der letzten Änderung der Anforderung ebenfalls geändert haben. Dies kann den Nutzer beim Bearbeiten der Anforderung unterstützen, da dieser sofort die geänderten Codings einsehen kann. Über den `Timestamp` der Versionen lässt sich ermitteln, wann die Anforderung zuletzt vom Nutzer geändert wurde. Dadurch lassen sich die geänderten Codings passend zu dieser Anforderung anzeigen. Analog zur Versionshistorie gibt es auch hier die Möglichkeit, die geänderten Codings in einem extra Fenster darzustellen. Eine weitere Möglichkeit wäre es, die Codings mit dem bekannten tabellarischen Design unterhalb des Änderungsgrunds während der Bearbeitung darzustellen. So ist der Nutzer in der Lage, sich die betroffenen Codings während der Bearbeitung anzeigen zu lassen. Dies dient der Unterstützung des Nutzers bei einer möglichst genauen Bearbeitung der Anforderung. Da bei einer Darstellung in einem extra Fenster die geänderten Codings nicht direkt während der Bearbeitung einsehbar sind, empfiehlt sich hier, entgegen dem Vorgehen der Versionshistorie, eine ein- und ausklappbare Darstellung unterhalb der zu bearbeitenden Anforderung.
Die zuvor vorgestellten Daten zu geänderten Codings lassen sich zudem Nutzen, um dem Nutzer zu visualisieren, dass mit einer Anforderung verknüpfte

Codings geändert oder hinzugefügt wurden. So erhält der Nutzer einen schnellen Überblick, welche Anforderungen möglicherweise, aufgrund von Änderungen an Codings, angepasst werden müssen.

5 Implementierung

Im folgenden Abschnitt wird die Implementierung der Anforderungen genauer erläutert. Zu Beginn werden die mit Umstrukturierung der Datenbank und den nötigen Anpassungen im Backend erläutert. Anschließend wird der Aufbau und die Funktionalität, die dem Nutzer im Frontend zur Verfügung gestellt wird, ausgearbeitet. Um die Funktionalität umfassend geprüft zu haben, werden abschließend die Anpassungen an den bestehenden Testszenarien beschrieben. Die Tabelle 5.1 stellt die Ausgangslage der Datenbank dar. Für jede Anforderung werden die folgenden Attribute abgespeichert, wobei das Attribut `parentID` auf den Wert `0` gesetzt wird, sollte diese Anforderung keine übergeordnete Anforderung besitzen. So lässt sich ein einfacher Einstiegspunkt ermitteln.

5.1 Speichern von Änderungen

5.1.1 Art des Speicherns

Wie in Abschnitt 4.1 dargestellt, gibt es mehrere Arten des Speicherns von neuen Versionen. Nach Abwägen der verschiedenen Varianten wird im Folgenden nun die Implementierung der in Abbildung 4.3 vorgestellten *Variante S2.2* beschriebenen. Hierbei wird eine vollständige Kopie der geänderten Anforderung in der Datenbank zurückgehalten.

Spaltenname	Datentyp	Beschreibung
ID	*Long*	Eindeutige ID der Anforderung
parentID	*Long*	ID der Übergeordneten Anforderung
project	*Long*	ID des Projekts
codeIDs	*List<Long>*	Liste mit der Anforderung verknüpfter Codes
title	*String*	Name der Anforderung
type	*String*	Typ der Anforderung
description	*String*	Beschreibung der Anforderung

Tabelle 5.1: Ausgangslage der Datenbank - `Requirement`

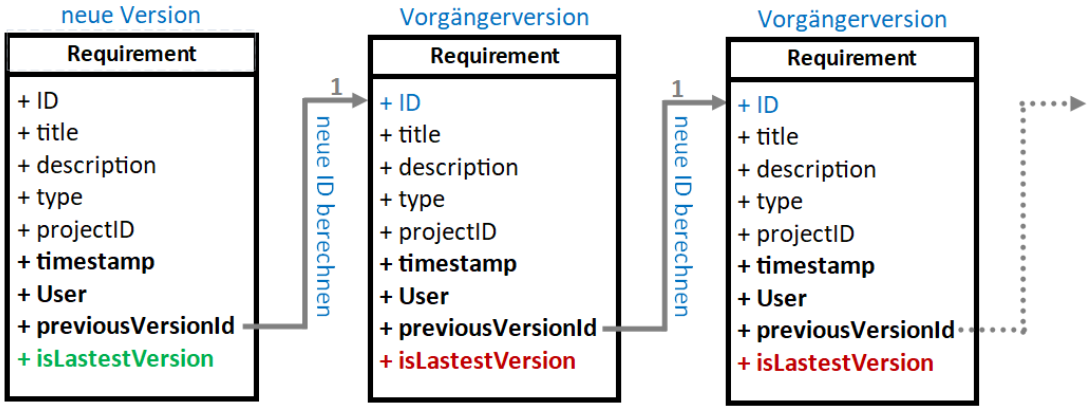


Abbildung 5.1: Speichern einer Kopie mit neuer ID als neue Anforderung

Anders als von SOPHISTen, 2021 beschrieben, wird in diesem Projekt nicht für jede neue Version eine neue `ID` generiert, sondern, wie in Abbildung 5.1 dargestellt, erhält die Vorgängerversion eine neue `ID`, auf welche dann in der neuen Version referenziert wird. Es ändert sich somit nur die `ID` der alten Versionen. Die `ID` der aktiven Anforderung bleibt hierbei unverändert. Diese konzeptionelle Änderung bietet den Vorteil, dass keine Referenzen auf diese Anforderung angepasst werden müssen. So zeigt eine bestehende Referenz auf die `ID` der Anforderung weiterhin auf die aktuelle Version. Ohne diese Anpassung müssten beispielsweise sämtliche Referenzen der `parentID` beim Anlegen einer neuen Version angepasst werden. Eine weitere Anpassung zu dem in Abschnitt 4.1 (V2.2) beschriebenen Vorgehen ist das Hinzuziehen eines weiteren Attributes `isLastestVersion`, welches die Filterung nach der aktuellen Version erleichtert.

Dementsprechend wurden zum Speichern einer neuen Version zwei zusätzliche Attribute angelegt:

- `previousVersionId`, speichert die `ID` der Vorgängerversion als *Long*
- `isLastestVersion`, speichert, ob die Anforderung die aktuelle Version ist als *Boolean*

Im Backend wird bei einer eingehenden Änderung einer Anforderung nun wie folgt vorgegangen. Zuerst wird eine Kopie der alten Anforderung mit einer neuen `ID` angelegt und der Wert `isLastestVersion` auf `false` gesetzt, da mit der eingehenden Änderung eine neuere Version der Anforderung existiert. Anschließend kann die bisherige Anforderung mit den neuen Werten der Änderung aktualisiert werden. Zudem muss vor dem Abspeichern der Änderungen die `previousVersionId` auf die `ID` der zuvor angelegten Kopie geändert werden. Durch dieses Vorgehen wird, dadurch dass jede neue Version einen Verweis auf die Vorgängerversion erhält, eine lineare Kette an Versionen einer Anforderung angelegt. Somit lässt sich später im Frontend eine vollständige Versionshistorie darstellen.

Spaltenname	Datentyp	Beschreibung
author	*String*	Autor der letzten Änderung
timestamp	*Date*	Zeitpunkt der letzten Änderung
previousVersionId	*Long*	ID der Vorgängerversion
isLastestVersion	*Boolean*	Zeigt, ob diese Anforderung die aktuelle Anforderung ist

Tabelle 5.2: Zusätzliche Attribute der Datenbank durch Versionierung

Um weitere Daten zur Änderung zu sammeln, wurden neben diesen Anpassungen zwei weitere Attribute in der Datenbank angelegt:

- `author`, speichert den vollständigen Namen des Nutzers, welcher die Anforderung geändert hat als *String*
- `timestamp`, speichert den Zeitpunkt der Änderung als *Date*

Die Informationen zum Autor werden dem Backend wie ein normales Attribut vom Frontend übergeben. Um den genauen Zeitpunkt zu ermitteln, wir das Attribut `timestamp` erst im Backend generiert und automatisiert abgespeichert. So werden die Zeitpunkte konsistent von der Zeit des Backends abgeleitet und Konflikte mit unterschiedlichen Zeitzonen vermieden.
Die für das Speichern einer neuen Version zusätzlich hinzugefügten Attribute sind in Tabelle 5.2 zusammengefasst aufgeführt.

Löschen einer Anforderung und deren Versionen

Diese Änderungen erfordern jedoch nicht nur beim Speichern einer Anforderung Anpassungen, sondern betreffen auch das Löschen einer Anforderung. Hierbei sollten auch die Vorgängerversionen dieser Anforderung gelöscht werden. Dieses Vorgehen erfordert einen rekursiven Aufruf, da beim Löschen einer Anforderung zuerst geprüft werden muss, ob eine Vorgängerversion existiert, welche dann zuerst gelöscht werden muss. Zudem müssen beim Löschen einer Anforderung auch alle untergeordneten Anforderungen gelöscht werden. Wie in Abbildung 5.2 dargestellt sollen beim Löschen der Anforderung mit `ID` *1* alle weiteren abgebildeten Anforderungen gelöscht werden, da es sich zum einen um untergeordnete Anforderungen handelt und zum anderen um deren alte Versionen, welche ebenfalls gelöscht werden sollen.

Archivieren einer Anforderung und deren Versionen

Analog zum Vorgehen beim Löschen gestaltet sich auch das Archivieren einer Anforderung. Der einzige Unterschied hierbei ist, dass die Anforderung nicht gelöscht wird, sondern ein zusätzliches Attribut `isArchived` auf `true` gesetzt

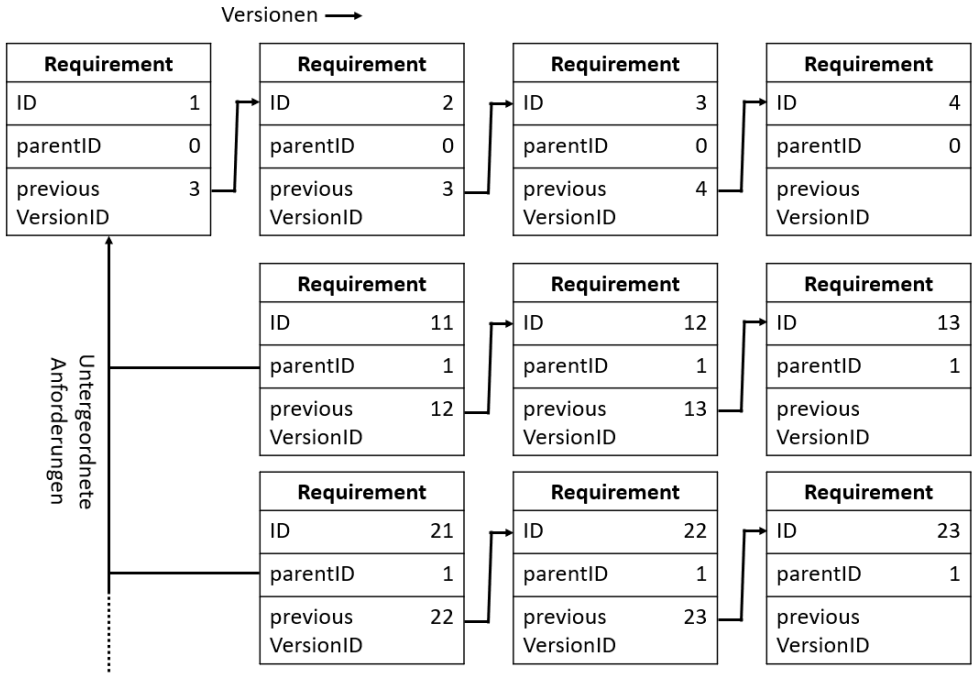


Abbildung 5.2: Löschen aller untergeordneten Anforderungen und deren Vorgängerversionen

wird. Da durch das Archivieren von Anforderungen keine Objekte gelöscht werden und die Datenmenge in der Datenbank stetig ansteigt, wird die Option *Löschen einer Anforderung* aus Gründen der Wartbarkeit Nutzern mit der Rolle *ADMIN* weiterhin zur Verfügung stehen.

5.1.2 Anlegen einer neuen Version

Wie in Abschnitt 4.1.2 hergeleitet, soll für jede Änderung an einer Anforderung eine neue Version angelegt werden. Beim Speichern der Änderungen muss der Nutzer jedoch den Änderungsgrund hinterlegen. Um diese Entscheidung für den Nutzer möglichst trivial zu gestalten, wurde im Frontend eine Klasse `ChangeType` angelegt. Diese Klasse hält eine Auswahl an möglichen Änderungsgründen, welche in Tabelle 5.3 aufgelistet ist.

Die Klasse `ChangeType` hält neben der Vorauswahl auch die Übersetzungen vor, sodass Änderungen an den Gründen für den Entwickler einfach verwaltet werden können.
Zusätzlich zu den Änderungsgründen, die dem Nutzer zur Verfügung gestellt werden, sind folgende automatisierte Änderungstypen für verschiedene Aktionen notwendig, um zusätzlichen Mehraufwand für den Nutzer zu verhindern:

- *parent*, wird automatisch als Grund hinterlegt, wenn der Nutzer per

changeType	Beschreibung
content	Inhaltliche Änderung
spelling	Verbesserung von Rechtschreibung und Grammatik
additions	Ergänzung der Anforderung
restructure	Umstrukturierung der Anforderung
context	Änderung des Zusammenhangs
others	Sonstige Änderungen

Tabelle 5.3: Vorauaswahl an Änderungsgründen

Drag&Drop die übergeordnete Anforderung anpasst. Hierbei werden im Kommentar die *IDs* der alten und der neuen übergeordneten Anforderung als JavaScript Object Notation (JSON) abgelegt. So lässt sich der Name der Anforderung ohne zusätzlichen Aufruf der Backends in der Datenbank speichern und später in der Versionshistorie darstellen.

- *code*, wird automatisch als Grund hinterlegt, wenn der Nutzer per Drag&Drop einen neuen Code mit der Anforderung verknüpft oder einen Code von einer Anforderung entfernt.
- *revert*, wird automatisch als Grund hinterlegt, wenn der Nutzer die Anforderung auf einen alten Stand zurücksetzt.

Wie bereits in Abschnitt 4.1.2 erläutert, soll der Nutzer zusätzlich zum Änderungsgrund einen Kommentar hinterlegen können. Um diese beiden Änderungen möglichst einfach nutzen zu können, wurde die bestehende GUI zum Bearbeiten von Anforderungen, wie in Abbildung 5.3 dargestellt, um eine weitere Zeile mit einem Dropdown-Menü, welches eine Vorauswahl an Änderungsgründen bietet, und einem Eingabefeld, für einen optionalen Kommentar, ergänzt. Diese Felder bleiben solange deaktiviert, bis der Nutzer Änderungen an der Anforderung vornimmt, um ein Abspeichern ohne Änderungen zu verhindern. Da ein Änderungsgrund zwingend erforderlich ist, können die Änderungen erst nach dem Auswählen eines Grunds abgespeichert werden. Ist dies nicht der Fall, erhält der Nutzer einen Warnhinweis, welcher ihn darauf hinweist, dass ein Änderungsgrund zwingend erforderlich ist. Das Abspeichern wird in diesem Fall abgebrochen, sodass der Nutzer die Möglichkeit erhält den fehlenden Grund zu ergänzen.

5.2 Kennzahlen zum Auswerten der Versionierung

Wie in Abschnitt 4.3.2 bereits angemerkt, lassen sich die Kennzahlen einer Anforderung im noch zu implementierenden Fenster *Versionshistorie* darstellen. Hierfür wurde die Methode `calcStatistics(versions)` im Frontend angelegt, welche als Parameter alle Versionen einer Anforderung als *Array* übergeben be-

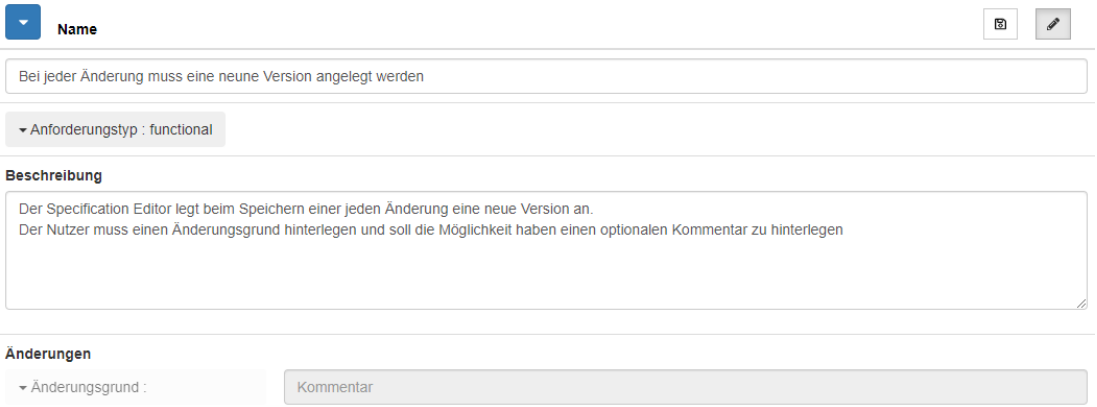


Abbildung 5.3: GUI zum Bearbeiten der Anforderung - Zusätzliche Felder zum Auswählen eines Änderungsgrundes und hinterlegen eines optionalen Kommentars

kommt. Damit lassen sich alle in Abschnitt 4.2.2 beschriebenen Kennzahlen berechnen. So lässt sich über die Länge des *Arrays* die *Versionsanzahl* abbilden. Zum Ermitteln der *Anzahl der Beteiligten im Prozess* wird ein Array angelegt, in welchem die Autoren abgelegt werden. Beim Iterieren durch die einzelnen Versionen wird nun geprüft, ob sich der Autor der Version bereits im Array befindet. Sollte dies nicht der Fall sein, wird dieser dem *Array* hinzugefügt. So lassen sich analog zur *Versionsanzahl* über die Länge des *Arrays* die Anzahl der Autoren und somit die *Anzahl der Beteiligten im Prozess* ermitteln. Um die *Änderungen pro Code* abzuleiten, muss zuerst, analog zum Vorgehen bei den Autoren, die Anzahl der Codes ermittelt werden, welche bereits mit dieser Anforderung verknüpft waren. Dividiert man nun die zuvor ermittelte *Versionsanzahl* durch die Anzahl der verwendeten Codes lassen sich die *Codes pro Version* ableiten. Da davon auszugehen ist, dass jede Version auch mehrere Änderungen halten kann, muss für die Kennzahl *Änderungen pro Code* die Gesamtzahl der Änderungen abgeleitet werden. Hierfür muss ebenfalls durch alle Versionen iteriert werden. Dabei müssen dann alle Attribute mit der Vorgängerversion verglichen werden. Liegt eine Änderung vor, wird ein zuvor angelegter Zähler erhöht. Dividiert man nun die Anzahl der verwendeten Codes durch die im Zähler ermittelte Anzahl an Änderungen, erhält man die *Änderungen pro Code*. Dieser Zähler lässt sich zudem auch als Kennzahl *Änderungsanzahl* darstellen oder mit der Anzahl der Tage ins Verhältnis setzen, welche seit dem Erstellen der Anforderung vergangen sind. Dadurch ergibt sich die *Änderungsrate* pro Tag.

5.3 Design und Funktionen im Frontend

Im folgenden Abschnitt wird die Implementierung und die Umstrukturierung der GUI im Frontend genauer betrachtet.

5.3.1 Erweiterungen der bestehenden GUI

Dynamisches Rendern der Attribute

Wie in Abschnitt 4.3.1 erläutert, soll die GUI so angepasst werden, dass statt dem statischen Rendern der Attribute ein einfaches Erweitern der Attribute möglich ist. So sollen alle Attribute einer Anforderung, die in der Datenbank gespeichert sind, im Frontend dargestellt werden können. Das Backend liefert diese Informationen hierzu im JSON-Format. Es wird also ein *Key* mit dem dazugehörigen *Value* zu jedem Attribut geliefert. Um die Übersetzung der *Keys* zentral zur Verfügung zu stellen, wurde eine neue Klasse `RequirementAttributes` im Frontend angelegt. Diese neue Klasse stellt nun die statische Methode `getLocalizedAttribute(attribute)` zur Verfügung, welche den *Key* der Anforderung entgegennimmt. So kann über ein *switch*-Statement die passenden Lokalisierung zurückgegeben werden. Da nicht alle Attribute der Anforderung wirklich im Frontend dargestellt werden sollen, stellt die Klasse `RequirementAttributes` zusätzlich eine Blacklist zur Verfügung, welche alle *Keys* hält, die beim Darstellen im Frontend nicht berücksichtigt werden sollen. Dies ist beispielweise bei den Attributen `isLastestVersion` oder `previousVersionId` der Fall. So kann durch die Umstrukturierung über die *Keys* iteriert werden, was ein flexibles Rendern aller in der Datenbank gespeicherten Attribute ermöglicht.

Möglichkeit eine Anforderung zu Archivieren

Wie ebenfalls in Abschnitt 4.3.1 erläutert, bedarf es mit der Versionierung auch der Möglichkeit eine Anforderung zu archivieren. Die Möglichkeit des Löschens von Anforderungen soll nur noch Nutzern mit der Rolle *ADMIN* zu Verfügung stehen. So kann die Vollständigkeit und Nachverfolgbarkeit sichergestellt werden. Für die Funktionalität des Archivierens wurde analog zum Entfernen der bestehende Realtimeservice erweitert, sodass die Anforderungen auch bei anderen Nutzern umgehend ausgeblendet werden. Beim Darstellen des *Entfernen* Buttons muss nun zusätzlich die Rolle geprüft werden. Sollte der Nutzer der Rolle *ADMIN* zugeordnet sein, wird der Button wie gehabt dargestellt. Ist dies nicht der Fall, entfällt der *Entfernen* Button.

Umstrukturierung der Klassenstruktur im Frontend

Da, wie in Abschnitt 4.3.2 beschrieben, das bestehende Design der Anforderungen in die Versionshistorie übernommen werden soll, bietet es sich an, einen einfachen

Prototypen einer Anforderung anzulegen. Von diesem Prototypen können dann sämtliche Klassen, die sich mit dem Darstellen von Anforderungen beschäftigen, erben. Dadurch werden Redundanzen im Quellcode vermieden und die Wartung erleichtert. Zudem können so Anpassungen bezüglich des Erscheinungsbilds schnell und einfach in der neuen Klasse `SimpleRequirementElement` durchgeführt werden. Die neue Klasse `SimpleRequirementElement` stellt 3 grundlegende Ansichten zur Verfügung:

- `renderChangedRequirement()`, stellt die sich geänderten Attribute zweier Anforderungen dar
- `renderSmallRequirement()`, stellt eine *eingeklappte* Anforderung ohne Attribute dar
- `renderDetailedRequirement()`, stellt alle Attribute einer Anforderung dar

Die Klassen, welche von `SimpleRequirementElement` erben, müssen grundlegend lediglich folgende 3 Methoden implementieren, auf welche die Methoden der Klasse `SimpleRequirementElement` zugreifen:

- `getRequirement()`, liefert die darzustellende Anforderung
- `getTitle()`, stellt den Titel der Anforderung dar, bei der Versionhistorie wird hier beispielsweise der Änderungsgrund zurückgegeben
- `renderButtons()`, liefert ein Array an Buttons, welche je nach Status dargestellt werden sollen

Die Buttons und der Inhalt der Titelzeile werden somit nach wie vor in der Unterklasse implementiert. Zudem werden alle Daten ebenfalls in der Unterklasse gehalten. Dies ermöglicht eine flexible aber dennoch einheitliche Darstellung einer Anforderung oder Version.

Gesammeltes Abspeichern von geänderten Anforderungen

Um bei mehreren ausstehenden Änderungen nicht für jede Änderung einen eigenen Änderungsgrund hinterlegen zu müssen, wurde ein neues Fenster `UnsavedChangesModal` implementiert. Dort hat der Nutzer die Möglichkeit, alle bisher ungespeicherten Änderungen zentral mit einem Änderungsgrund zu versehen oder auch zu verwerfen. Zum Darstellen der geänderten Anforderungen wird hier die zuvor vorgestellte neue Klasse `SimpleRequirementElement` verwendet.

5.3.2 Darstellen einer Versionshistorie

Wie in Abschnitt 4.3.2 hergeleitet, soll die Versionshistorie in einem eigenen Fenster angezeigt werden, um eine Überschneidung mit den aktuellen Anforderungen

zu vermeiden und die Übersichtlichkeit zu bewahren. Wie bereits in Abschnitt 4.3.1 erwähnt, wird die Darstellung einer Anforderung um einen weiteren Button *Versionshistorie* ergänzt, welcher das Anzeigen des neuen Fensters ermöglicht. Um das neue Fenster darzustellen, wurde eine neue Klasse `VersionHistoryModal` angelegt. Dort werden einerseits die Kennzahlen bezüglich dieser Anforderung dargestellt, andererseits auch sämtliche Versionen der Anforderung. Um eine Version darzustellen, wurde die neue Klasse `VersionHistoryElement` angelegt.

VersionHistoryElement

Zum Darstellen einer Version einer Anforderung kann zunächst von der zuvor erläuterten Klasse `SimpleRequirementElement` geerbt werden. Um jedoch die Änderungen zur Vorgängerversion darzustellen wird initial die darzustellende Version mit deren Vorgängerversion über die Methode `compareVersions` verglichen und die ermittelten Unterschiede in einem *Dictionary* `changesDict` abgelegt. Hierfür wird über die *Keys* der Anforderung iteriert und mit den Werten der Vorgängerversion verglichen. Bei unterschiedlichen Werten wird der Wert und der dazugehörige *Key* im *Dictionary* abgelegt. So kann die Render-Methode zur Laufzeit prüfen, ob der Key im *Dictionary* vorliegt und das entsprechende Attribut bei Änderungen farblich hervorheben. Die Werte der Änderungen werden ebenfalls im *Dictionary* abgespeichert, um die in Abschnitt 4.3.2 vorgestellte *Variante VH3* ohne Mehraufwand darzustellen. Hier wird statt der gesamten Anforderung nur das generierte *Dictionary* `changesDict` dargestellt. Durch den Button *Vollständige Anforderung anzeigen* hat der Nutzer nun die Möglichkeit, zwischen der vollständigen Anforderung, welche in Abbildung 5.5 dargestellt ist, und nur den geänderten Attributen zu wechseln. Diese Ansicht ist in Abbildung 5.4 zu sehen. Beide Ansichten stellen beispielhaft Version 3 in den beiden unterschiedlichen Ansichten dar. Wie zu sehen ist, wird in Abbildung 5.4 nur die geänderte *Beschreibung* dargestellt, wohingegen Abbildung 5.5 auch die nicht geänderten Attribute, wie *verknüpfte Codes*, den *Namen* oder den *Anforderungstyp*, darstellt.
Anstatt wie bei Anforderungen den Titel der Anforderung in der ersten Zeile als Überschrift darzustellen, wird der Änderungsgrund sowie der optionale Kommentar dargestellt. Das bietet dem Nutzer eine schnelle Übersicht zu den Änderungen der jeweiligen Version. Zudem wird darüber hinaus die Versionsnummer errechnet und ebenfalls dargestellt.
Eine weitere Besonderheit ist die Darstellung der initialen Version *1*. Da hier keine Änderungen vorliegen können, wird bei dieser Anforderung statt dem *Änderungsgrund* immer der *Titel* angezeigt. Zudem entfällt die Ansicht der geänderten Attribute. Stattdessen wird immer die vollständige Anforderung dargestellt.

Wiederherstellen einer Vorgängerversion

Das Abspeichern einer vollständigen Kopie kann sich beim Wiederherstellen einer älteren Version einer Anforderung zu Nutze gemacht werden. Eine ältere Version kann über den Button *Anforderung auf diese Version zurücksetzen* ohne Anpassungen im Backend wiederhergestellt werden. Hierfür wird im Frontend die Methode `revertVersion` angelegt. Diese kann nun die bestehende API-Schnittstelle `updateRequirement` verwenden, welche eine neue Version der Anforderung anlegt. Es können alle Daten der alten Version verwendet werden, mit dem Unterschied, dass die `ID` der aktiven Anforderung benutzt werden muss, sowie der Autor und der Änderungsgrund angepasst werden muss. Es wird also eine Kopie der alten Anforderung als Änderung ans Backend geschickt, wo dann eine neue Version angelegt wird. So werden zwar die Attribute der alten Version übernommen, die Versionshistorie bleibt aber erhalten. Wie in den Abbildungen 5.4 und 5.5 zu sehen, erhält jede Version, mit Ausnahme der aktuellen Version, den Button *Anforderung auf diese Version zurücksetzen*. Da das Zurücksetzen auf aktive Anforderung keine Änderungen mit sich bringen würde, wurde hier auf den Button verzichtet.

Darstellen der Kennzahlen einer Anforderung

Die in Abschnitt 5.2 berechneten Kennzahlen können nun ebenfalls tabellarisch im neuen Fenster der Versionshistorie dargestellt werden. Wie in Abbildung 5.4 zu sehen wurde das gleiche Design wie zum Darstellen von Anforderungen oder Versionen verwendet. Beim Ausklappen der Kennzahlen können diese nun anhand der Versionen einer Anforderungen ohne weiteren Aufruf des Backends berechnet werden. Der Button *Information* bietet dem Nutzer zudem eine Beschreibung der jeweiligen Kennzahl, welche dem besseren Verständnis dient.

5.3.3 Anzeigen der geänderten Codings

Um das Anzeigen der geänderten Codings seit der letzten Änderung der Anforderung zu ermöglichen, wurde eine neue API-Schnittstelle `listChangedCodingsForCodeIds` implementiert. Hierbei werden die verknüpften *CodeIds* und der Zeitpunkt der letzten Änderung übergeben. Über eine neue Datenbankabfrage lassen sich alle Codings durchsuchen, die einen `timestamp` haben, welcher nicht weiter zurückliegt als der der letzten Änderung. Zudem lässt sich prüfen, ob das Coding mit einer `CodeID` verknüpft wurde, welche im übergebenen Array *CodeIds* liegt. So liefert diese Schnittstelle alle geänderten Codings seit der letzten Änderung. Zudem wird bei den Codings geprüft, ob diese im Zusammenhang mit den mit der Anforderung verknüpften Codes stehen.
Die in Abbildung 5.3 wird nun um den in Abbildung 5.6 gezeigte Abschnitt erweitert. Hier werden die Codings, welche zuvor von der neuen API-Schnittstelle

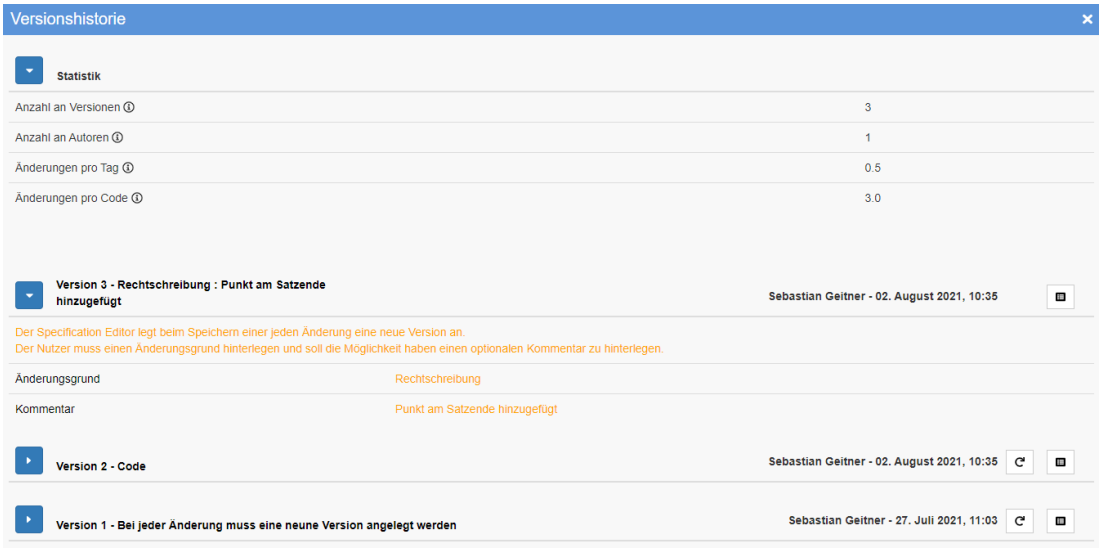


Abbildung 5.4: GUI Versionshistorie - Ansicht der geänderten Attribute von Version 3

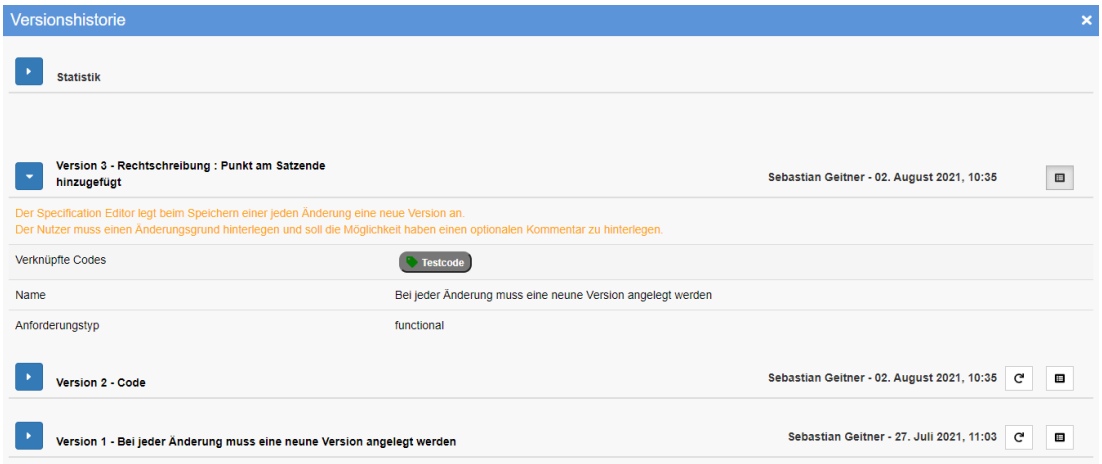


Abbildung 5.5: GUI Versionshistorie - Detailansicht der Version 3

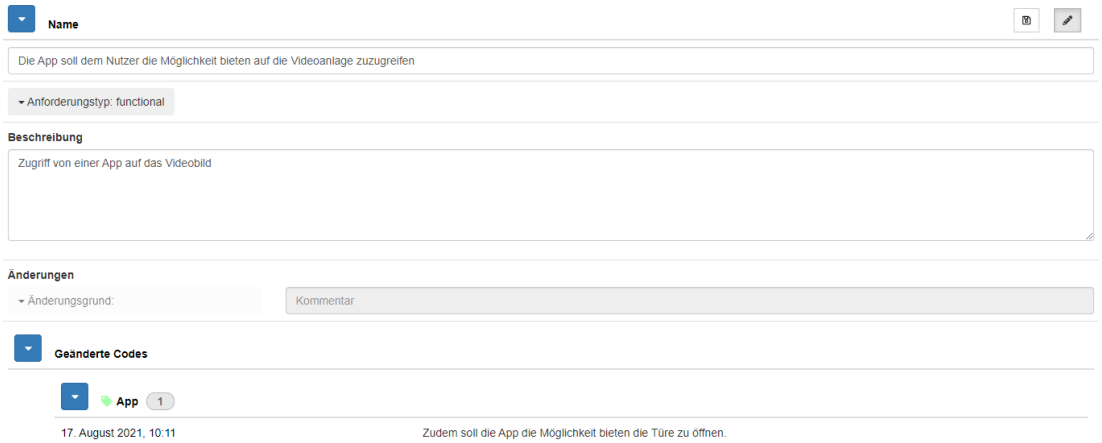


Abbildung 5.6: GUI Bearbeiten einer Anforderung - Geänderte Codings welche mit der Anforderung verknüpft sind

Abbildung 5.7: GUI Übersicht Anforderungen - Fahne kennzeichnet Änderungen an verknüpften Codings

zurückgeliefert wurden, nach deren zugehörigen Codes gruppiert und in ähnlichem Design wie dem einer Anforderung dargestellt. Hierbei gilt es zu berücksichtigen, dass Änderungen von Codes sämtliche Informationen zu dessen Codings liefern sollen. Beim Hinzufügen eines neuen Codes sind dementsprechend alle Codings von Bedeutung, nicht nur jene, welche sich seit der letzten Änderung geändert haben.
Die neue API-Schnittstelle `listChangedCodingsForCodeIds` lässt sich zudem verwenden, um den Nutzer bereits vor dem Bearbeiten darauf aufmerksam zu machen, dass sich Codings, welche mit dieser Anforderung in Verbindung stehen, geändert haben. Dies wird über ein zusätzliches Symbol in der Titelzeile signalisiert, sodass der Nutzer sofort sieht, welche Anforderungen möglicherweise angepasst werden müssen. Dies wurde wie in Abbildung 5.7 über eine rote Fahne umgesetzt. Um einen zusätzlichen Backend-Aufruf für jede darzustellende Anforderung zu verhindern, werden diese Daten bereits beim Aufruf der Methode `getRequirements` aus der Datenbank abgerufen. Zudem wird im Frontend der bestehende Realtimeservice bei Änderungen an Codings genutzt. Auch hieraus lassen sich die Daten zu geänderten Codings abrufen, was dazu führt, dass auch hier auf eine zusätzliche Datenbankabfrage für jede Anforderung verzichtet werden kann.

5.4 Tests

Nach den zuvor vorgestellten Anpassungen im Backend ist es notwendig, deren beabsichtigte Funktionalität zu prüfen. Hierfür wurde die Struktur der Testfälle ausgebaut. Zu Beginn eines Tests wird eine verzweigte baumartige Struktur an Anforderungen angelegt. Hierfür werden 4 unabhängig voneinander existierende *Bäume* an Anforderungen angelegt. Jeder *Ast* erhält hierbei eine unterschiedliche Anzahl an Unteranforderungen. So können realistische Situationen simuliert werden, wobei es beispielsweise beim Löschen einer Anforderung auch ein Löschen von mehreren untergeordneten Anforderungen bedarf.
Die Testmethode `testRequirementUpdate` muss nun neben den Attributen prüfen, ob ein neuer `timestamp` angelegt wurde. Zudem wird nun getestet, ob beim aktualisieren einer Anforderung eine Kopie der alten Version angelegt wird, und dort das Attribut `isLastestVersion` auf `false` gesetzt wird.
Neben dem Entfernen von Anforderungen muss zudem nun analog das Archivieren von Anforderungen geprüft werden. Beim Archiviern von Anforderungen soll sich die Anzahl der gespeicherten Anforderungen nicht ändern. Das Attribut `isArchived` muss jedoch auf `true` gesetzt werden, sodass sich alle archivierten Anforderungen leicht über dieses Attribut herausfiltern lassen.
Da die Wiederverwendung einer Anforderung vollständig über das Frontend abgedeckt wird, genügt hier der erweiterte Test `testRequirementUpdate`.

6 Evaluation

In diesem Abschnitt werden die ausgehenden Anforderungen aus Kapitel 3 mit dem implementierten Stand von QDAcity verglichen und die Ergebnisse eingeordnet. Zudem wurden in einer abschließenden heuristischen Evaluation nach Nielsen (Nielsen, 1995) die Konzepte und die Nutzbarkeit der Anforderung durch Experten geprüft.

6.1 Einordnen der Anforderungen

Im Folgenden wird die Umsetzung der Anforderungen aus Kapitel 3 betrachtet:

Req 1.0

Anforderung *Req 1.0* wurde vollständig erfüllt.
Die in Anforderung *Req 1.0 - Der Specification Editor muss dem Nutzer die Möglichkeit bieten eine Versionshistorie anzuzeigen* wird durch ein neues Fenster umgesetzt, welches über einen zusätzlichen Button *Versionshistorie* aufgerufen werden kann. Die Darstellung der einzelnen Versionen wurde durch eine Vererbung der Klasse `SimpleRequirementElement` realisiert, wodurch sich die Oberfläche konsistent zu den bisherigen Anforderungen gestaltet.

Req 1.1

Anforderung *Req 1.1* wurde vollständig erfüllt.
Beim Ausklappten einer Anforderung werden die in Anforderung *Req 1.1 - Die Versionshistorie sollte Änderungen zur Vorgängerversion darzustellen* geforderten Änderungen in der Versionshistorie übersichtlich dargestellt. Durch den Button *Vollständige Anforderung anzeigen* hat der Nutzer zudem die Möglichkeit die vollständige Anforderung einzusehen. Hierbei werden ebenfalls die geänderten Attribute hervorgehoben.

Req 1.2

Anforderung *Req 1.2* wurde erfüllt.
Wie die beiden zuvor genannten Anforderungen wurde auch Anforderung *Req 1.2 - Die Versionshistorie sollte dem Nutzer die Möglichkeit bieten Kennzahlen zu den Versionen der Anforderungen darzustellen* im neuen Fenster Versionshistorie implementiert. Dabei hat der Nutzer die Möglichkeit sich die in Abschnitt 4.2.2 erläuterten Kennzahlen darstellen zu lassen.

Req 2.0

Anforderung *Req 2.0* wurde erfüllt.
Um Anforderung *Req 2.0 - Die Versionshistorie muss gegen nachträgliche Änderungen geschützt sein* umzusetzen, wurde der Button *Löschen* für Nutzer, welche nicht über die Rolle *ADMIN* verfügen, durch den Button *Archivieren* ersetzt. So bleiben sämtliche Anforderungen in der Datenbank erhalten. Zudem wird bei jedem Speichern einer Anforderung eine neun Version angelegt. Das Bearbeiten von Versionen ist nicht mehr möglich. Auch beim Wiederherstellen einer älteren Version wird eine Kopie dieser angelegt, sodass die Versionshistorie erhalten bleibt.

Req 2.1

Anforderung *Req 2.1* wurde vollständig erfüllt.
Wie bereits in der vorherigen Einordnung erwähnt, bleibt der Button *ADMIN* erhalten, um Anforderung *Req 2.1 - Die Versionshistorie soll einem Administrator die Möglichkeit geben Anforderungen zu löschen* zu erfüllen. So ist ein Löschen von fälschlich angelegten Anforderungen dennoch durch autorisierte Nutzer möglich.

Req 2.2

Anforderung *Req 2.2* wurde vollständig erfüllt.
Wie in Anforderung *Req 2.2 - Der Specification Editor sollte dem Nutzer die Möglichkeit bieten eine Anforderung zu archivieren* gefordert, bietet der bereits vorher erwähnte Button *Archivieren* jedem Nutzer die Möglichkeit eine Anforderung aus dem Specification Editor zu entfernen, ohne dabei die Vollständigkeit und Nachverfolgbarkeit zu untergraben.

Req 3.0

Anforderung *Req 3.0* wurde vollständig erfüllt.
Jedes Speichern einer Änderung sorgt für das Anlegen einer neuen Version. Dieses Vorgehen wird in Abschnitt 5.1.2 erläutert und erfüllt somit die Forderungen aus

Anforderung *Req 3.0 - Anforderungen müssen dem Nutzer die Möglichkeit bieten eine neue Version anzulegen.*

Req 3.1

Anforderung *Req 3.1* wurde vollständig erfüllt.
Auch die Implementierung der Anforderung *Req 3.1 - Anforderungen müssen dem Nutzer die Möglichkeit bieten beim Abspeichern einen Kommentar zum Änderungsgrund anzulegen* wurde in Abschnitt 5.1.2 beschrieben.

Req 3.2

Anforderung *Req 3.2* wurde vollständig erfüllt.
Die Umsetzung der Anforderung *Req 3.2 - Die Versionshistorie sollte dem Nutzer die Möglichkeit bieten den Stand einer älteren Version wiederherzustellen* wurde vollständig im Frontend durchgeführt. Hierbei wird wie in Abschnitt 5.3.2 im neuen Fenster der Versionshistorie ein Button *Anforderung auf diese Version zurücksetzen* implementiert. Hier wird eine Kopie dieser Version als neue Anforderung angelegt. Hierbei kann der alte Stand wiederverwendet werden ohne die Vollständigkeit und Nachverfolgbarkeit zu verletzen.

Req 3.3

Anforderung *Req 3.3* wurde verworfen.
Anforderung *Req 3.3 - Der Specification Editor sollte dem Nutzer die Möglichkeit bieten zu entscheiden, ob eine neue Version angelegt wird* wurde, wie in Abschnitt 4.1.2 erläutert, nicht wie gefordert umgesetzt. Aus Gründen der Vollständigkeit und Nachverfolgbarkeit wird bei jeder Änderung eine neue Version angelegt. Das gewünschte Ausblenden von unnötigen Änderungen in der Versionshistorie lässt sich dennoch durch das Filtern nach Änderungsgründen realisieren. Somit wird der Grundgedanke dieser Anforderung trotz anderer Umsetzung dennoch erfüllt.

Req 4.0, 5.0, 6.0

Anforderungen *Req 4.0, 5.0, 6.0* wurden vorbereitet.
Die Anforderungen 4.0 bis 6.0 werden im Abschnitt 7 behandelt. Aufgrund der neuen Struktur der Datenbank und dem Anlegen einer vollständigen Kopie einer Version, wurden bereits alle nötigen Vorbereitungen für diese zusätzlichen Funktionen getroffen.

6.2 Nutzerevaluation der GUI

Um die in der Arbeit ermittelten Konzepte und die Benutzbarkeit der GUI einzuordnen, wurde abschließend eine Evaluation durch Experten durchgeführt.

6.2.1 Vorgehen

Die folgende Nutzerevaluation basiert auf einer heuristischen Evaluation nach Nielsen (Nielsen, 1995). Hierbei begutachten und testen Domainexperten die Nutzeroberfläche.
Anderes als bei Nielsen wird die folgende Evaluation nur von zwei Experten durchgeführt. Die Experten werden im Folgenden als Evaluatoren bezeichnet. Bei jeder Evaluation steht ein Observer zur Verfügung, welcher die Evaluation leitet und für weitere Fragen zur Verfügung steht. Nach einer Einführung des Observers ins Projekt, werden den Evaluatoren drei spezifische Aufgaben gestellt.
Um die Nutzerevaluation durchzuführen wurde ein fiktives Szenario entwickelt, welches die Entwicklung einer Türsprechanlage beschreibt. Zur Durchführung der Evaluation wurde im Tool QDAcity ein Projekt mit mehreren zum Teil bereits vor codierten Dokumenten angelegt. Zudem wurden im Specification-Editor bereits alle nötigen Anforderungen angelegt und mit den betreffenden Codes verknüpft. Dem Evaluator wurden nun folgende drei Aufgaben gestellt:

- Identifikation von betroffenen Anforderungen
- Erstellen einer neuen Version
- Auswerten der Versionshistorie

Nach Bearbeitung jeder Aufgabe wurde der Evaluator nach folgenden Themen befragt:

- Probleme
- Einsetzbarkeit in der Praxis
- Verbesserungen
- Vergleich zum alltäglich genutzten Tool/Vorgehen

Abschließend sorgt ein Fragebogen nach Ssemugabi und De Villiers, 2007 für eine Vergleichbarkeit der Ergebnisse. Der vollständige Auswertungsbogen befindet sich im Anhang C dieser Arbeit.

6.2.2 Auswertung

Das Tool, insbesondere der Specification-Editor wurde im Allgemeinen als sehr Benutzerfreundlich empfunden. Für den Alltagseinsatz würde jedoch die im Kapi-

tel 7 beschriebene Filterfunktion für einen Mehrwert sorgen. Im Folgenden werden die Aufgabenstellungen abstrahiert ausgewertet.

Identifikation von betroffenen Anforderungen

Bei der Identifikation der betroffenen Anforderungen stellte sich die Markierung der betroffenen Anforderungen, mit der in Abbildung 5.7 dargestellten Fahne, als sehr hilfreich bezüglich der Identifikation dar. Für ungeübte Nutzer kann sich diese Fahre jedoch als zu unauffällig darstellen. Nach dem ersten Wahrnehmen dieser, stellt sich die Darstellung dennoch als zutreffend und hilfreich heraus. Im alltäglichen Einsatz können so die in Stichpunkten vorliegenden Ausgangsdaten codiert und später in bestehende Anforderungen eingearbeitet werden. Durch das markieren der betroffenen Anforderungen mit einer Fahne kann so eine Halbautomatisierung erreicht werden. Zudem ersetzt die Verknüpfung der Anforderung mit Codes eine Kategorisierung der Anforderung durch zusätzlich zu verwaltende Attribute.

Erstellen einer neuen Version

Beim Einarbeiten der nötigen Änderungen war bei der ersten Änderung nicht klar, wann eine neue Version einer Anforderung angelegt wird. Nach Speichern der Änderung konnte gut nachvollzogen werden, dass jedes Speichern einer Änderung zum Anlegen einer neuen Version führt. Hier wäre eine Feedback an den Nutzer hilfreich, wann eine neue Version angelegt wird.
Das automatische Abspeichern des Autors und des Änderungszeitpunkts, sorgt für geringeren Arbeitsaufwand des Nutzers, da diese Daten nicht händisch im Kommentar erfasst werden müssen.
Die vorgegebenen Änderungsgründe decken alle benötigten Anwendungsfälle ab. Die Gründe *Zusammenhang* und *Umstrukturierung* werden in den meisten Fällen nicht benötigt und könnten somit entfernt werden, um Unklarheiten zu vermeiden. Es wäre zudem denkbar die Standardfälle auf auf *Inhaltliche Änderung* für "Breaking Changes" und *Redaktionelle Änderung* für kleinere Änderungen wie Rechtschreibfehler zu reduzieren. Zudem könnten individuelle Kategorien für jedes Projekt dem Nutzer einen Mehrwert bieten.
Der Abschnitt *geänderte Codings*, welcher in Abbildung 5.6 dargestellt wird, unterstützt den Nutzer erheblich in der Ausarbeitung der Anforderung, da betroffene Textabschnitte direkt während dem Bearbeiten einsehbar sind. Diese Abschnitt könnte jedoch farblich noch von den Anforderungen abgegrenzt werden. Zudem wurde die farbliche Kennzeichnung von bearbeiteten aber noch ungespeicherten Anforderungen als sehr nützlich eingeordnet, da so Fehler durch das Vergessen des Speichervorgangs vermieden werden können.

Auswerten der Versionshistorie

Bei der Auswertung der Versionshistorie traten keinerlei Probleme auf, da diese sehr übersichtlich und einheitlich gestaltet wurde.
Auch das Zurücksetzen einer Anforderung war ohne Vorkenntnisse problemlos möglich. Ein zusätzlicher Dialog, der zur Bestätigung einer Aktion führt, könnte hier ein ungewolltes Zurücksetzten vermeiden. Ein verpflichtender Kommentar des Nutzers könnte den Grund für das zurücksetzen noch genauer erfassen.
Alle nötigen Metriken konnten ebenfalls eingesehen werden. In beiden Evaluationen wurde die Metrik *Anzahl an Autoren* als sehr hilfreich empfunden. Da die Metrik *Änderungen pro Code* in anderen Anwendungen nicht genutzt wird, wurde hier eine Einordnung dieser Werte als schwierig angesehen. Eine Auswertung dieser findet in der alltäglichen Nutzung jedoch selten statt. Hilfreich wäre hier eine Funktion die Metriken über das gesamte Projekt oder einzelne Revisionen auszuwerten. Dies würde die Möglichkeit bieten eine Auswertung einer Entwicklungsiteration zu starten.

6.2.3 Einordnung der Ergebnisse

Aus den Erkenntnissen der Evaluation wurden nach jeder Aktion an der Datenbank Banner in das Tool integriert, welches dem Nutzer mehr Feedback zu den betreffenden Aktionen gibt. Zudem wurde eine Rückfrage bei kritischen Aktionen implementiert, was zum vermeiden von Fehlern führt. So ist es nicht mehr möglich eine Anforderung versehentlich zu archivieren oder zurücksetzen.
Die Versionierung der Anforderungen eignet sich nach Einschätzungen der Experten durchaus für den Praxiseinsatz. Gerade die Verknüpfung mit den Codes aus dem Codesystem können einen Mehrwert im Vergleich zu anderen Tools darstellen. Für den Praxiseinsatz wird jedoch noch die Möglichkeit benötigt, für jedes Projekt individuelle zusätzliche Attribute hinzuzufügen, um so den Detailgrad verfeinern zu können. Zudem könnte die Anwendung noch durch Verweise auf andere Attribute und die in Abschnitt 2.2.1 beschriebenen Varianten ergänzt werden.

7 Ausblick

Das folgende Kapitel soll einen Ausblick des Tools geben, sowie über weitere nützliche Features informieren.

7.1 Nützliche Features

Im Laufe der Arbeit haben sich weitere Features als nützlich erwiesen, welche der Anwendung einen Mehrwert bieten können. Dieser Abschnitt gibt einen Ausblick über noch zu implementierende Features.

Varianten einer Anforderung

Wie in Abschnitt 2.2.1 vorgestellt, würde es einen Mehrwert bieten, wenn eine Anforderung in mehreren Varianten vorliegen kann (Ebert, 2019). Diese Varianten gilt es jedoch bei der Versionierung ebenfalls zu beachten. So müsste zum Beispiel das Anpassen einen Rechtschreibfehlers in sämtlichen Varianten für Änderungen sorgen. Dies würde somit auch für jede Variante zu einer neuen Version führen.

Filtern und Sortieren von Anforderungen

Um die Übersichtlichkeit gerade bei einer großen Anzahl an Anforderungen zu bewahren, bietet es sich an eine Möglichkeit des Filtern und Sortierens zu schaffen. So könnten beispielsweise bereits archivierte Anforderungen wieder eingeblendet werden. Denkbar wäre auch das Wiederherstellen dieser zu ermöglichen. Durch das in Abschnitt 5.1.1 beschriebene Anlegen einer vollständigen Kopie, gestaltet sich das Filtern und Sortieren der Anforderungen kaum aufwendiger als ohne Versionierung.

Export von Anforderungen

Da die Anforderungen den letzten Schritt des Tools QDAcity darstellt, kann eine Exportfunktion die Nutzung der generierten Anforderungen in weiteren Tools ermöglichen. Hierbei gilt es jedoch auch im Sinne der Vollständigkeit und Nachver-

folgbarkeit sicherzustellen, dass nicht nur die aktiven Anforderungen exportiert werden, sondern auch deren Versionen. Denkbar wäre dann auch eine Wiederverwendung der Anforderungen in anderen Projekten im Tool QDAcity. Zum Export der Anforderungen bietet sich hierbei der ReqIF-Standard[1] an. Es ist zudem die Unterstützung anderer gängiger Formate wie JSON oder Comma separated value (CSV) denkbar.

Satzschablone für Anforderungen

Die Möglichkeit eine Satzschablone zu verwenden sorgt für das Vermeiden von Fehlern und eine einheitliche Struktur der Anforderungen (SOPHISTen, 2021). Der Nutzer könnte durch eine vorgefertigte Satzschablone beim Anlegen und Bearbeiten der Anforderungen unterstützt werden.

Unveränderbarer Stand der Anforderungen

Laut SOPHISTen, 2021 wäre es hilfreich, wenn das Tool die Möglichkeit bieten würde einen unveränderlichen Stand festzulegen. So könnte in agilen Vorgehensweisen vor jedem Sprint die Anforderungen für die Entwickler eingefroren werden.

Individuelle Attribute

Laut der Nutzerevaluation wäre es hilfreich, wenn der Nutzer die Möglichkeit hätte für jedes Projekt zusätzliche individuelle Attribute anzulegen. So kann jedes Projekt abhängig vom Kunden individuell angepasst und erweitert werden.

Verweise auf andere Projekte

Anhand der Nutzerevaluation könnten durch verweise auf andere Projekte eine übersichtlichere Struktur geschaffen werden. So wäre es denkbar anstatt eine gesamten Anforderung eine Gruppe anzulegen, welche dann anstatt von einem Set an Anforderungen einen Verweis auf ein anderes Projekt hält. So werden einzelne Funktionalitäten abstrahiert und aus dem Projekt ausgegliedert.

[1] https://www.omg.org/spec/ReqIF

8 Fazit

Basierend auf verschiedenen Ansätzen der Literatur konnte in dieser Arbeit ein umfassendes Konzept zur Versionierung von Anforderungen entwickelt werden. Dabei konnten konkrete Forderungen der Literatur bezüglich der Vollständigkeit und Nachverfolgbarkeit umgesetzt werden. Dies spiegelt sich vor allem in der Umsetzung des Anlegens einer neuen Version wider. Anders als ursprünglich angedacht, wurde dem Nutzer keine Möglichkeit gegeben, zu entscheiden ob eine neue Version einer Anforderung angelegt werden soll. Bei jeder Änderung wird nun eine neue Version der Anforderung angelegt. Dies nimmt zum einen dem Nutzer die Entscheidung ab, wann eine Änderung zu einer neuen Version führt und vermeidet somit Fehlentscheidungen. Zum anderen werden dadurch alle Änderungen an einer Anforderung dokumentiert, was die Nachverfolgbarkeit und die Vollständigkeit sicherstellt. Um jedoch eine unübersichtliche Versionshistorie durch eine Vielzahl an Änderungen zu vermeiden, muss bei jedem Speichern ein Änderungsgrund hinterlegt werden. Hierdurch besteht die Möglichkeit, die Versionshistorie zu einem späteren Zeitpunkt filtern zu können. Die Vorbereitung dieser Funktion konnte bereits in dieser Arbeit getroffen werden. Die vollständige Implementierung wurde hingegen noch zurückgestellt.
Neben der Fragestellung, wann eine neue Version einer Anforderung angelegt werden soll, wurde sich auch mit der Art des Speicherns der neuen Version beschäftigt. Nach dem Abwägen verschiedener bekannter Ansätze wie *GIT* und *Subversion* und der Auswertung bekannter Literatur zur Versionsverwaltung wurde sich für das Anlegen einer vollständigen Kopie der Anforderung entschieden. Dieser Ansatz bietet zum einen Vorteile beim Filtern der Anforderungen, da in der Datenbank schnell alle alten Versionen durchsucht werden können. Zum anderen wird das Darstellen und Zurücksetzen vorheriger Versionen erleichtert, da die Versionen vollständig vorliegen und keine Berechnungen, basierend auf alten Änderungen, notwendig sind. Auf diese Weise lassen sich Änderungen zur Vorgängerversion einfach vergleichen und übersichtlich darstellen.
Schlussendlich haben diese Designentscheidungen auch den Vorteil, dass sich die Berechnung von Kennzahlen bezüglich der Versionierung als unproblematisch gestaltet, da, wie bereits erwähnt, alle Daten vollständig in der Datenbank vorliegen. Eine Übersicht über die implementierten Kennzahlen findet sich in Abschnitt

4.2.
Um die neu generierten Informationen übersichtlich darzustellen wurde das in Abbildung 5.4 dargestellte Fenster *Versionshistorie* implementiert. Neben den verschiedenen Kennzahlen wird dem Nutzer hier eine lineare Versionshistorie geboten, welche neben der vollständigen Version auch die Änderungen zur Vorgängerversion darstellt. Zudem hat der Nutzer die Möglichkeit, die Anforderung auf eine alte Version zurückzusetzen. Um die Vollständigkeit zu gewährleisten, wird hier eine Kopie der alten Version erstellt und als aktive Version eingefügt.
Aus der Verknüpfung der Anforderungen mit dem bestehenden Codesystem im Tool QDAcity konnte abschließend eine Mehrwert der neuen Versionierung gebildet werden. Durch die Dokumentation der Änderungszeitpunkte von Codings und Anforderungen lassen sich die möglicherweise noch zu ändernden Anforderungen in der Nutzeroberfläche kennzeichnen. Folglich kann ein Nutzer ohne Mehraufwand die von Änderungen betroffenen Anforderungen identifizieren. Zudem lassen sich beim Bearbeiten einer Anforderung die verknüpften Codings, bei welchen einer Änderung vorliegt, darstellen, wodurch ein Nutzer die Änderungen direkt während des Bearbeitens einer Anforderung einsehen kann. Dieses Vorgehen wurde in einer abschließenden Evaluation ebenfalls als nützlich eingeordnet.

Appendices

A Evaluation RE-Experte 1

Identifikation von betroffenen Anforderungen

Es bestand das Problem, dass beim ersten Anlegen des Codings, unklar war, wie ein Coding angelegt werden kann. Markieren des Textes und anschließendes Auswählen des Codings im Codesystem war nicht selbsterklärend. Nach erstem Verständnis keine weiteren Probleme.
Die Fahne gestaltet sich bei erster Nutzung als zu unauffällig. Ist jedoch im Praxiseinsatz sehr hilfreich.
Prinzipiell sehr hilfreich im Praxiseinsatz. Anforderungsdokumente liegen oft in Stichpunkten und einem Fragebogen vor.
Verschiedene Aufgaben sind sehr gut in einer Anwendung gebündelt. Anzahl der Codings im Codesystem sehr hilfreich für eine erste grobe Einschätzung. Fahnen sehr hilfreich bei der Identifikation er betroffenen Anforderungen.
Im alltäglichen Einsatz werden aus einer Vorlage oder Stichpunkten richtige Anforderungen entwickelt und durch Attribute kategorisiert, um ein Filtern zu ermöglichen. Die Filterung kann durch Codierung erleichtert werden.

Erstellen einer neuen Version

Beim Speichern von Änderungen war anfangs unklar, wann eine neue Version angelegt wird. Die Versionierung ist jedoch sehr hilfreich, da Autor und Änderungszeitpunkt automatisiert festgehalten werden.
Geänderte Codes optisch besser von Anforderungen abgrenzen. Es könnte sich bei erster Betrachtung auch um eine weitere Anforderung handeln. Die Änderungsgründe „Zusammenhang“ und „Umstrukturierung“ werden nicht benötigt und sorgen für Unklarheiten. Der Änderungsgrund „sonstiges“ könnte möglicherweise missbraucht werden. Dies könnte vermieden werden, wenn beim Wählen des Grundes „sonstiges“ das Kommentarfeld zwingend erforderlich wäre. So muss der Nutzer seinen Grund genauer spezifizieren. Ein Feedback, wenn eine Änderung oder eine neue Version angelegt wird, wäre förderlich und würde dem Nutzer mehr Sicherheit geben. Es wäre gut, wenn ungespeicherte Änderungen leicht verworfen werden könnten.
Das Farbliche hervorheben von ungespeicherten Änderungen vermeidet Fehler und hält den Nutzer über den aktuellen Stand informiert. Der Abschnitt „Geänderte Codings“ ist sehr hilfreich beim Bearbeiten von Anforderungen, da ein Wechseln zum Anfoderungsdokument nicht nötig ist.
Im alltäglichen Vorgehen muss der Nutzername und das Änderungsdatum manuell hinterlegt werden. Dieses Vorgehen wurde hier automatisiert.

Auswerten der Versionshistorie

Das automatisierte Erstellen einer Versionshistorie ist in der Praxis sehr hilfreich. Alle nötigen Metriken vorhanden.
Vor dem Durchführen eines Reverts sollte der Nutzer zur Sicherheit seine Aktion nochmals bestätigen müssen. Dies kann über einen Dialog „Wollen sie die Anforderung auf einen alten Stand zurücksetzten? JA/Nein“ realisiert werden.
Das Zurücksetzen der auf eine ältere Version gestaltet sich sehr einfach und intuitiv. Die Versionshistorie ist sehr übersichtlich gestaltet. Die Möglichkeit die detaillierte Version einzusehen kann im Alltag hilfreich sein.
Die Statistik wird im Alltag eher über alle Anforderungen ausgewertet. Im Vergleich mit den Kennzahlen verschiedener Anforderungen kann die Statistik hilfreich sein, um häufig geänderte Anforderungen zu identifizieren. Das Wiederherstellen von älteren Versionen wird aktuell manuell durchgeführt, sodass das hier eine Erleichterung im Alltag erzielt werden kann.

sonstige Anmerkungen

Für den Einsatz im Alltag wäre eine Filterfunktion nötig.
Das Anzeigen und Wiederherstellen von archivierten Anforderungen wäre hilfreich.
Um fehlerhafte Aktionen zu verhindern sollten Aktionen wie Archivieren oder Wiederherstellen vom Nutzer vor dem Abschluss zusätzlich bestätigt werden müssen.

B Evaluation RE-Experte 2

Identifikation von betroffenen Anforderungen

Es bestand das Problem, dass beim ersten Anlegen des Codings, unklar war, wie ein Coding angelegt werden kann. Markieren des Textes und anschließendes Auswählen des Codings im Codesystem war nicht selbsterklärend. Es wurde versucht ein Coding per Drag&Drop anzulgen. Nach erstem Verständnis keine weiteren Probleme.
Fahne zu unauffällig bei erster Benutzung.
Prinzipiell sehr hilfreich im Praxiseinsatz. Anforderungsdokumente liegen oft in schon als Doors-Export des Kunden vor.
Fahne zur Markierung von Änderungen sehr hilfreich. Dies sorgt für eine Halbautomatisierung.
Je nach Kunde und Projekt wird der Schritt des Codierens in der Praxis teils übersprungen. Keines der bekannten Tools unterstützt das Hervorheben der zu ändernden Anforderungen. Dieser Schritt muss oft händisch erledigt werden.

Erstellen einer neuen Version

Geänderte Codings zu schwer einzusehen / zu unauffällig. In der Bearbeitungsansicht hebt sich der Abschnitt „geänderte Codings“ zu wenig ab, sodass dieser leicht übersehen werden kann.
Rechtschreibung könnte zur besseren Klarheit in Redaktionell umbenannt werden
Aufteilung in 2 Änderungsgründe genügt oftmals:

- Größere Änderung / Inhaltliche Änderung (Breaking Change)
- Redaktionelle Änderung / Rechtschreibung

Bei mehr als 2 Änderungsgründen wären Beispiele sehr sinnvoll, um Fehler zu vermeiden. Einteilung der Kategorien sind jedoch oft vom Projekt abhängig.
Anpassbare Kategorien inklusive deren Verpflichtungen wären hilfreich. Vergleichbare Tools bieten die Möglichkeiten der individuellen Kategorisierung. Zudem kann zu jeder Kategorie angegeben werden, ob diese verpflichtend ist. In manchen Projekten muss der Ursprung der Änderung angegeben werden (Kunde/Entwicklungsteam)

Auswerten der Versionshistorie

Änderungen pro Tag könnten sich auch auf „Tags“ beziehen. Unklar, ob es sich dabei um einen Zeitraum handelt. Auswertungen über das gesamte Projekt/ über eine gesamte Revision an Anforderungen würden einen Mehrwert bieten. (Auswertung einer Entwicklungsiteration)
Ein Kommentar bei einem Revert würde den Grund noch genauer erfassen.

Möglichkeit die Statistiken über bestimmte Kategorien zu erstellen (Rechtschreibung uninteressant)
Die Aussage der Metrik *Änderungen pro Code* ist unklar.
Anzahl der Autoren eine sehr sinnvolle Metrik. Zurücksetzen sehr einfach möglich.
Möglichkeit Kennzahlen über ein gesamtes Projekt auszuwerten.

sonstige Anmerkungen

- Banner bei Drag&Drop von Anforderungen hinzufügen (Einheitliches Userfeedback)
- Name der geänderten Anforderung in den Banner einbauen
- Zusätzliche individuelle Attribute (Custom Fields)
- Drag zur Root besser darstellen (Entfernen einer Übergeordneten Anforderung)
- Sortieren/Gruppieren und Zwischenüberschriften
- Verweise auf andere Projekte
- Verschiedene Ansichten für verschiedene Nutzerrollen
- Änderungen verwerfen Funktion

C Auswertung Fragebogen

Die Fragestellungen im Fragebogen konnten auf eine Skala von 1 bis 5 bewertet werden.

- 1: trifft nicht zu
- 5: trifft voll zu

C.1 Generelle Benutzbarkeit

Die Anwendung informiert die Nutzer durch konstruktives, angemessenes und rechtzeitiges Feedback über die aktuellen Status/Zustand

3,5 *(RE1: 3 - RE2: 4)*

Das System reagiert auf die vom Benutzer durchgeführten Aktionen. Es gibt keine überraschenden Aktionen der Website oder langwierige Dateneingaben.

5 *(RE1: 5 - RE2: 5)*

Der Sprachgebrauch, z. B. Begriffe, Phrasen, Symbole und Konzepte, ähnelt dem der Benutzer in ihrer täglichen Umgebung.

4,5 *(RE1: 5 - RE2: 4)*

Die Verwendung von Metaphern entspricht den Objekten/Konzepten der realen Welt, d. h. es werden verständliche und aussagekräftige symbolische Darstellungen verwendet, um sicherzustellen, dass die verwendeten Symbole, Icons und Namen im Zusammenhang mit der ausgeführten Aufgabe intuitiv sind.

4 *(RE1: 4 - RE2: 4)*

Die Informationen sind in einer natürlichen und logischen Reihenfolge angeordnet.

4 *(RE1: 4 - RE2: 4)*

Der Nutzer hat die Kontrolle über das System

5 *(RE1: 5 - RE2: 5)*

Aktionen können rückgängig gemacht werden

4,5 *(RE1: 4 - RE2: 5)*

Der Nutzer kann das tun, was er tun möchte

5 *(RE1: 5 - RE2: 5)*

Die gleichen Begriffe, Wörter, Symbole, Situationen oder Handlungen beziehen sich auf die gleiche Sache.

5 *(RE1: 5 - RE2: 5)*

Gemeinsame Plattformstandards (im Vergleich zu anderen domainspezifischen Tools) werden eingehalten

4,5 *(RE1: 5 - RE2: 4)*

Das System ist so konzipiert, dass die Benutzer nicht ohne weiteres schwerwiegende Fehler machen können.

4 *(RE1: 4 - RE2: 4)*

Wenn ein Benutzer einen Fehler macht, gibt die Anwendung eine entsprechende Fehlermeldung aus. Die Fehler lassen sich auflösen.

5 *(RE1: 4 - RE2: 5)*

Die zu bearbeitenden Objekte, die Auswahlmöglichkeiten und die auszuführenden Aktionen sind sichtbar.

5 *(RE1: 5 - RE2: 5)*

Der Benutzer braucht sich nicht von einem Teil des Dialogs an Informationen eines anderen Dialogs zu erinnern

5 *(RE1: 5 - RE2: 5)*

Die Anzeigen sind einfach und mehrseitige Anzeigen sind auf ein Minimum reduziert.

5 *(RE1: 5 - RE2: 5)*

Die Website ist für verschiedene Benutzerebenen geeignet, vom Anfänger bis zum Experten.

3,5 *(RE1: 3 - RE2: 4)*

Die Benutzeroberfläche lässt schnelle und Effektive Aktionen zu, um den Nutzer schnell an sein Ziel zu bringen

4 *(RE1: 4 - RE2: 4)*

Die Dialoge auf der Website enthalten keine irrelevanten oder selten benötigten Informationen, die den Benutzer bei der Ausführung seiner Aufgaben ablenken könnten.

5 *(RE1: 5 - RE2: 5)*

Fehlermeldungen werden in einfacher Sprache ausgedrückt.

5 *(RE1: 5 - RE2: 5)*

Fehlermeldungen definieren Probleme genau und geben schnelle, einfache, konstruktive, spezifische Anweisungen zur Behebung.

5 *(RE1: 5 - RE2: 5)*

Wenn ein getippter Befehl zu einem Fehler führt, muss der Benutzer nicht den gesamten Befehl neu eingeben, sondern nur den fehlerhaften Teil reparieren.

4 *(RE1: 4 - RE2: 4)*

C.2 RE-spezifische Benutzbarkeit

Der Nutzer wird beim Finden zu ändernder Anforderungen unterstützt

4,5 *(RE1: 4 - RE2: 5)*

Der Nutzer kann einfach von der Anforderung zum betroffenen Teil im Anforderungsdokument navigieren.

5 *(RE1: 5 - RE2: 5)*

Der Nutzer kann während dem Bearbeiten einer Anforderung alle zu berücksichtigenden Informationen einsehen

4,5 *(RE1: 5 - RE2: 4)*

Die vorgeschlagenen Änderungsgründe decken alle benötigten Fälle ab.

Ja, aber zu viele nicht benötigte Gründe

Literaturverzeichnis

Blischak, J. D., Davenport, E. R. & Wilson, G. (2016). A Quick Introduction to Version Control with Git and GitHub. *PLOS Computational Biology*, *12*(1), 1–18. https://doi.org/10.1371/journal.pcbi.1004668

Bühne, S. & Hermann, A. (2015). *Handbuch Requirements Management nach IREB Standard* (1.0.1). International Requirements Engineering Board.

Chacon, S. & Straub, B. (2014). *Pro Git*. Apress.

Dick, J., Hull, E. & Jackson, K. (2017). *Requirements Engineering* -. Springer.

Ebert, C. (2019). *Systematisches Requirements Engineering - Anforderungen ermitteln, dokumentieren, analysieren und verwalten*. dpunkt.verlag.

ISO/IEC/IEEE International Standard - Systems and software engineering–Measurement process. (2017). *ISO/IEC/IEEE 15939:2017(E)*, 1–49. https://doi.org/10.1109/IEEESTD.2017.7907158

Kaufmann, A. & Riehle, D. (2017). The QDAcity-RE method for structural domain modeling using qualitative data analysis. *Requirements Engineering*, *24*, 85–102. https://doi.org/10.1007/s00766-017-0284-8

Nielsen, J. (1995). How to conduct a heuristic evaluation. *Nielsen Norman Group*, *1*, 1–8.

Rupp, C. (2021). *Requirements-Engineering und -Management*. Die SOPHISTen.

Sarkan, H. M., Ahmad, T. P. S. & Bakar, A. A. (2011). Using JIRA and Redmine in requirement development for agile methodology. *2011 Malaysian Conference in Software Engineering*, 408–413. https://doi.org/10.1109/MySEC.2011.6140707

SOPHISTen, D. (2021). *Die RE-Fibel*. Die SOPHISTen.

Ssemugabi, S. & De Villiers, R. (2007). A comparative study of two usability evaluation methods using a web-based e-learning application. *Proceedings of the 2007 annual research conference of the South African institute of computer scientists and information technologists on IT research in developing countries*, 132–142.