**Hochschule**
**Bonn-Rhein-Sieg**
*University*
*of Applied Sciences*

Fachbereich Informatik
Department of Computer Sciences

Forschungsgruppe TIM
(Technologie- und Innovationsmanagement)

Prof. Dr.
**Karl W. Neunast**

E-Mail: Karl.Neunast@h-brs.de

# Abschlussbericht

Forschungsprojekt - SIWOB

(Projekt-Nr. FF-FP0345)

# Inhaltsverzeichnis

# Abbildungsverzeichnis

# Tabellenverzeichnis

# Abkürzungsverzeichnis

| | |
|---|---|
| AGPL | Affero General Public License |
| ANSI | American National Standards Institute |
| ANTLR | Another Tool for Language Recognition |
| API | Application Programming Interface |
| B2B | Business to Business |
| BC | Binding Components |
| BPEL | Business Process Execution Language |
| BPM | Business Process Management |
| BS | British Standard |
| BSI | Bundesamt für Sicherheit in der Informationstechnik |
| CDI | Contexts and Dependency Injection |
| CPU | Central Processing Unit |
| CRUD | Create-Read-Update-Delete |
| CSS | Cascading Style Sheets |
| DGUV | Deutsche Gesetzlichen Unfallversicherung |
| EAI | Enterprise Architecture Integration |
| EAV | Entity-Attribute-Value |
| EJB | Enterprise Java Bean |
| EN | Europäische Norm |
| ESB | Enterprise Service Bus |
| FTP | File Transfer Protocol |
| GPL | General Public License |
| GUI | Graphical User Interface |
| HTML | Hypertext Markup Language |
| HTTP | Hyper Text Transport Protocol |
| ID | Identifikationsnummer |
| IEC | International Electrotechnical Commission |
| IEEE | Institute of Electrical and Electronics Engineers |
| IFA | Institut für Arbeitsschutz |
| ISO | Internationale Organisation für Normung |
| Java EE | Java Enterprise Edition |
| JBI | Java Business Integration |
| Java EE | Java Enterprise Edition |
| JCA | Java Connection Architecture |
| JMS | Java Messaging Service |
| JSF | Java Server Faces |
| JSP | Java Server Pages |
| JSR | Java Specification Request |
| LOC | Lines of Code |
| MD5 | Message-Digest Algorithm |
| MOM | Message-oriented-Middleware |

| | |
|---|---|
| MRL | Maschinenrichtlinie |
| MVC | Model View Controller |
| NMS | Normalized Message Service |
| NTFS | New Technology File System |
| QUIM | Quality in Use Integrated Measurement |
| Regex | Regular expression |
| RIA | Rich Internet Application |
| RPC | Remote Procedure Calls |
| RTES | Real Time Embedded System |
| RWD | Responsive Web Design |
| SE | Service Engines |
| SHA | Secure Hash Algorithm |
| SIL | Security Integrity Level |
| SIWOB | Safety Inspection Workbench |
| SOA | Serviceorientierte Architektur |
| SOAP | Simple Object Access Protocol |
| SPI | Service Provider Interface |
| SPS | Speicherprogrammierbare Steuerung |
| SPSS | Speicherprogrammierbare Steuerungs-Software |
| UI | User Interface |
| URI | Uniform Resource Identification |
| URL | Uniform Resource Locator |
| VE-Nummer | Vorgangserfassungsnummer |
| VFS | Virtual File System |
| W3C | World Wide Web Consortium |
| WAN | Wide Area Network |
| WSDL | Web Services Description Language |
| WYSIWYG | What You See Is What You Get |
| XHTML | Extensible Hypertext Markup Language |
| XML | Extensible Markup Language |
| XSD | XML Schema Definition |
| XSL | Extensible Stylesheet Language |
| XSLT | Extensible Stylesheet Language Transformation |

# 1 Titel und Laufzeit des Vorhabens

| | |
|---|---|
| **Titel** | Safety Inspection Workbench (SIWOB) |
| **Laufzeit** | 01. März 2013 - 31. Dezember 2014 |
| **Berichtszeitraum** | 01. März 2013 - 31. Dezember 2014 |

# 2 Problemstellung

Hersteller von Maschinen und von Sicherheitsbauteilen für Maschinen dürfen ihre Produkte in Europa nur dann in Verkehr bringen, wenn diese strengen Sicherheits- und Gesundheitsanforderungen entsprechen. Die EU-Maschinenrichtlinie (MRL) 2006/42/EG, als rechtliche Vorgabe, ist für jeden Hersteller bindend und konsequent einzuhalten. Die MRL fordert sichere Maschinen, wobei der Aspekt der Sicherheit und wie dieser eingehalten werden kann, durch Normen genauer spezifiziert wird. Sollte eine Maschine oder ein Sicherheitsbauteil von einer Norm abweichen, so muss eine Baumusterprüfung durch eine notifizierte Prüfstelle durchgeführt werden. Prüfstellen, wie das Institut für Arbeitsschutz (IFA) der Deutschen Gesetzlichen Unfallversicherung (DGUV), werden deshalb regelmäßig mit der Prüfung von solchen Maschinen inkl. deren Software, betraut. Bei erfolgreicher Prüfung eines sogenannten Baumusters einer Maschine wird dem Hersteller eine Baumusterprüfbescheinigung durch das IFA ausgestellt.

Ein derartiger Prüfprozess kann von Herstellern durch die Vorlage der technischen Unterlagen eines fertigen Baumusters initiiert und mit der Ausstellung der Baumusterprüfbescheinigung nach bis zu drei Jahren Prüfdauer abgeschlossen werden. In der Praxis hat sich gezeigt, dass eine frühzeitige Zusammenarbeit von Prüfer und Hersteller zu positiven Effekten auf beiden Seiten führen kann. So ermöglicht das IFA der DGUV eine entwicklungsbegleitende Prüfung von Baumustern.

Maschinen werden fast ausnahmslos durch eingebettete Systeme gesteuert, die ihrerseits insbesondere sicherheitsrelevante Funktionen mittels Software kontrollieren. Neben der Sicherheitsprüfung von Hardware gewinnt die Untersuchung der Sicherheit von Software zunehmend an Bedeutung als Bestandteil von Baumusterprüfungen. Obwohl dafür zahlreiche Vorgehensmodelle existieren zeigen sich in der Praxis einige Verbesserungspotenziale bei der Organisation und Kommunikation von Software-Prüfungen.

Neben dem eigentlichen Baumuster benötigen Prüfer für einen solchen Vorgang vor allem die Spezifikation der Sicherheitsfunktionen sowie die Quelltexte der verwendeten Software. Neben der Überprüfung der Spezifikation hinsichtlich fehlender oder mangelhafter Sicherheitsfunktionen ist die Prüfung des Quelltextes gegen die Spezifikation eine der wesentlichen Aufgaben bei der Software-Prüfung.

Eine Prüfung kann nur dann erfolgreich durchgeführt werden, wenn Prüfinstituten geeignete Unterlagen zu Spezifikationen und Quelltexten etc. zur Verfügung gestellt werden. Im Wesentlichen handelt es sich hierbei um einen Verständnisprozess. Prüfer lesen vorhandene Unterlagen und müssen die beschriebenen Konzepte verstehen, bevor eine Prüfung der Sicherheitsfunktionen wirklich stattfinden kann.

Hinsichtlich der Struktur, des Formats und des Umfangs eingereichter Dokumente haben Hersteller relativ große Entscheidungsfreiheiten. Einerseits wird Herstellern dadurch die freie Wahl etwa von Modellierungswerkzeugen ermöglicht. Andererseits zeigt die Praxis, dass der überwiegende Teil eingereichter Dokumente ohne anwendungsgerechte Werkzeuge erstellt wurde (bspw. Word-Dokumente zur Spezifikation) und die inhaltliche Qualität nicht zuletzt aus diesem Grund leidet. Folglich müssen sich Prüfer neben ihrer eigentlichen Tätigkeit regelmäßig mit der Einarbeitung in die Struktur und die Darstellungsform eingereichter Spezifikationen oder mit der Handhabung bestimmter Dokumentformate beschäftigen. Die dafür notwendige Bearbeitungszeit könnte merklich verkürzt werden, wenn zu prüfende Unterlagen stärker strukturiert, in einheitlichem Aufbau und dem richtigen Umfang bei den Prüfern eingehen würden.

Neben diesem einheitlichen Aufbau von eingereichten Unterlagen mangelt es oft an Verknüpfungsmöglichkeiten zwischen den verschiedenen Dokumentarten. Beispielsweise beschreiben informale Spezifikationen das Systemverhalten in natürlicher Sprache, geben aber oft nur unzureichende Hinweise zum Auffinden der beschriebenen Funktionen im Quelltext. Würden eingereichte informale Spezifikationen schnell auffindbare Verknüpfungen zu Quelltexten oder Flussdiagrammen bieten, können Prüfer bei Ihren Aufgaben entlastet werden.

Im Verlauf eines Prüfvorgangs kommt es vor, dass Spezifikationen und Quelltexte in mehreren Iterationen zwischen Prüfer und Hersteller ausgetauscht werden müssen, bis eine zulassungsfähige Software entstanden ist. Um Veränderungen in Spezifikationen oder Quelltexten zwischen zwei oder mehr Iterationsschritten nachvollziehen zu können, kommt in Prüfinstituten oftmals ein manueller oder Tool-gestützter Dokumentvergleich zum Einsatz. Für einen derartigen Arbeitsschritt müssen in jedem Fall alte und neue Dokumente im Ablagesystem gesucht und geöffnet werden. Eine Unterstützung von Prüfern durch eine vollständig automatische Hervorhebung von Änderungen in aktuellen Dokumentversionen fehlt oftmals.

Ein weiterer wesentlicher Erfolgsfaktor bei Prüfvorgängen ist eine einheitliche Kommunikation zwischen Herstellern bzw. Entwicklern von Software auf der einen und Prüfern auf der anderen Seite. Da Hersteller in der Regel auf Anmerkungen und Kommentare durch Änderungen an den Unterlagen reagieren, könnte ein Mechanismus zur Hervorhebung von Veränderungen, wie der oben Genannte zur Optimierung der Kommunikation auf beiden Seiten beitragen.

Bei der Speicherung der Unterlagen zu aktiven und abgeschlossenen Prüfprozessen werden von Prüfinstituten oftmals statische Ordnungssysteme auf Ebene von Dateisystemen genutzt. Ein derartiges Vorgehen bietet zwar eine hohe Kompatibilität zu verschiedensten Plattformen, lässt aber Möglichkeiten im Umgang mit elektronischen Dokumenten ungenutzt. Wird beispielsweise für jeden Prüfprozess ein eigenes Verzeichnis im Dateisystem angelegt und anschließend von einem Prüfer mit einer (frei wählbaren) Struktur von Unterverzeichnissen und Dokumenten gefüllt, so ist die damit festgelegte Struktur für genau diesen Vorgang gültig. Ein weiterer Prüfer könnte für ein anderes Projekt eine völlig andere Struktur anlegen, die ihm sinnvoller erscheint. Falls ein Prüfer seine Dokumentstruktur ändern möchte oder einen Prüfvorgang von einem Kollegen übernehmen muss (bspw. wegen Abteilungswechsel, Ruhestand, Krankheit etc.) sind damit umfangreiche Arbeiten im Dateisystem verbunden. Verbesserungen am Abla-

gesystem kommen darüber hinaus bestenfalls neuen Projekten, nicht aber aktiven oder abgeschlossenen Vorgängen zugute. Des Weiteren ist es bei einer Dokumentverwaltung auf Dateisystemebene kaum möglich, Verknüpfungen oder semantische Zusammenhänge zwischen verschiedenen Dokumenten oder gar Abschnitten in Dokumenten darzustellen und nutzbar zu machen.

Verbesserungspotenziale bei Software-Prüfungen bestehen also zum einen in einer gezielten Unterstützung der Hersteller bei Aufbau, Art und Umfang eingereichter Unterlagen und zum anderen in der Nutzung von semantischen Zusammenhängen innerhalb solcher Unterlagen, um dadurch Informationen schneller und gezielter auffinden zu können. Darüber hinaus existieren für Prüfer wiederkehrende Aufgaben, die sich teilweise oder überwiegend durch ein IT-System abbilden lassen könnten und bisher häufig manuell erledigt werden.

# 3 Forschungszweck/-ziel

Das Hauptziel des Projekts ist die Erforschung einer Dokumentationsinfrastruktur, die in der Lage ist, Prüfprozesse abzubilden und beteiligte Akteure mittels geeigneter Werkzeuge bei ihren Aktivitäten zu unterstützen. Um die Machbarkeit eines solchen Ansatzes zu zeigen, wird im Rahmen des Forschungsvorhabens zunächst der Teilprozess der Software-Prüfung innerhalb des Baumusterprüfprozesses betrachtet.

Im Verlauf des Projekts sollen daher vor allem neue Erkenntnisse mit Relevanz für Prüfvorgänge von Software gewonnen werden. Diese sollen sich zum einen für Hersteller eignen, Prüfprozesse durch die Bereitstellung von gezielteren Informationen zu unterstützen und zum anderen Prüfern und Prüfinstituten durch neue Ansätze zur Verbesserung von Prüfprozessen zugutekommen.

Im Zentrum der Forschungsaktivitäten steht zunächst die Entwicklung einer Dokumentarchitektur, die alle für eine Prüfung von Software benötigten und erzeugten Dokumente aufnehmen und in einer geeigneten Struktur zusammengefasst abbilden kann. Ausgehend von der erforschten Dokumentarchitektur werden Formate und Tools entwickelt, die diesen speziellen Aufbau nutzen, um erweiterte Informationen zu ermitteln, deren manuelle Bestimmung und Pflege großen Aufwand bedeuten würde. Beispielsweise könnten Spezifikationen um bidirektionale Verknüpfungen zu den beschriebenen Funktionen in Quelltexten erweitert werden, die somit einen schnellen Abgleich zwischen Spezifikation und Realisierung möglich machen.

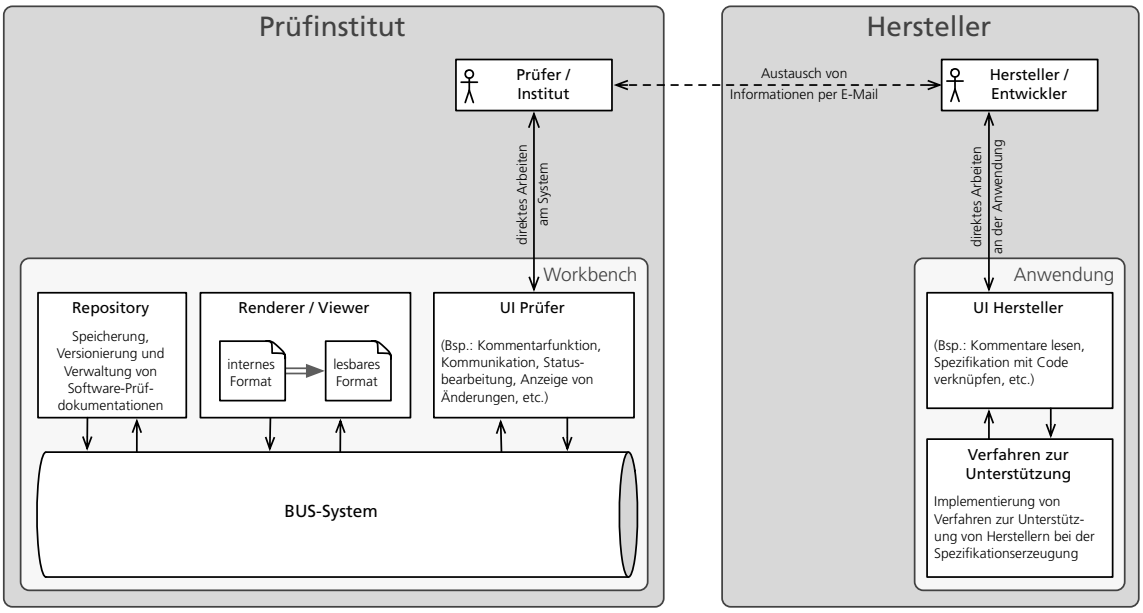



Abb. 1: Aufbau der Workbench mit einzelnen Komponenten

Entwickelte Tools werden innerhalb eines erweiterbaren BUS-Systems in Form einer Intranet-Applikation zur Verfügung gestellt (siehe Abbildung 1). Das BUS-System stellt dabei die zentrale Infrastruktur dar, an die Komponenten gekoppelt und mit deren Hilfe Informationen zwischen den Komponenten ausgetauscht werden können. Auf diese Weise werden Hersteller bei der Erzeugung von prüfungsgerechten

Spezifikationen unterstützt, müssen vertrauliche Informationen aber nicht auf öffentlich zugänglichen IT-Infrastrukturen speichern. Prüfinstitute können ihre Prüfvorgänge durch den Zugriff über ein internes BUS-System flexibel abrufen und profitieren von den erweiterten Anzeige- und Darstellungsmöglichkeiten von Informationszusammenhängen innerhalb oder zwischen einzelnen Prüfvorgängen.

Ein derartiges BUS-System ist im Prinzip bedingt skalierter und könnte daher auch zur Abbildung umfangreicherer Aufgaben, wie der Baumusterprüfung ausgebaut werden. Im Rahmen dieses Forschungsvorhabens soll jedoch zunächst die Machbarkeit eines derartigen Ansatzes anhand des ausgewählten Falls der Software-Prüfung gezeigt werden. Während einer Erhebungsphase wird dafür insbesondere untersucht, welche Art von Software, Embedded-Software, Speicherprogrammierbare Steuerungs-Software (SPSS) etc., bei Sicherheitsprüfungen besondere praktische Relevanz hat, um weitere Forschungsaktivitäten im Anschluss auf diese zu fokussieren.

## Identifikation von Workflows der Software-Prüfung

Das erste wesentliche Teilziel im Rahmen des Forschungsvorhabens ist die Identifikation von "Good Practice" Workflows zur Prüfung von Software. Basierend auf einer empirischen Erhebung der jeweiligen Anforderungen an Prüfverfahren bei Herstellern und Prüfern, sowie der Untersuchung von relevanten Normen, werden empfehlenswerte Workflows abgeleitet, die einen Prüfprozess von Software ganzheitlich beschreiben. Besonderer Fokus wird für dieses Teilziel auf die Identifikation wiederkehrender Aufgaben bei Software-Prüfungen gelegt, die einerseits manuellen Aufwand erfordern und andererseits durch IT-Systeme unterstützt werden können.

Im Zusammenhang mit der Identifikation von Workflows soll außerdem die Akzeptanz durch alle beteiligten Akteure berücksichtigt werden. Das Gesamtziel kann nur dann erreicht werden, wenn die Akzeptanz durch die Beteiligten schon bei den grundlegenden Entwicklungen vorhanden ist.

## Dokumentformat zur Abbildung von Prüfdokumentationen

Die Erforschung eines konsistenten und qualitätsgesicherten Dokumentformats zur Abbildung aller verwendeten und erzeugten Informationen ist ein weiteres zentrales Teilziel des Vorhabens.

Auf der Basis von zu entwickelnden theoretischen Modellen (Dokumentarchitektur) und deren Validierung wird ein konkretes Format entwickelt, das den o.g. Anforderungen entspricht und darüber hinaus den Anforderungen der Ziele der Informationssicherheit gerecht wird.

Das entwickelte Format kann mittels geeigneter Tools z.B. von Prüfinstituten dafür eingesetzt werden, Dokumente im Rahmen der Software-Prüfung zu verwalten und dabei die o.g. Vorteile der zugrundeliegenden Dokumentarchitektur nutzbar zu machen.

## BUS-System als Grundlage einer Infrastruktur

Die Entwicklung eines BUS-Systems, das in der Lage ist, Instanzen des erforschten Dokumentformats zu verwalten, ist ein praktisches Teilziel des Forschungsvorhabens. Zur Erreichung dieses Ziels werden verschiedene Lösungsansätze evaluiert. Ein derartiges BUS-System dient als Grundlage für eine Infrastruktur

zur IT-Unterstützung von Prüfvorgängen in Prüfinstituten und erfordert insbesondere die Auseinandersetzung mit Problemstellungen der Daten-, Identitäts- und Integritätssicherung.

Das entwickelte BUS-System soll primär in der Lage sein, die Verwaltung der für eine Prüfung relevanten Informationen und Dokumente zu unterstützen. Durch die Fähigkeit zur Integration zusätzlicher Tools soll das System skalier- und erweiterbar sein.

Eine wesentliche Komponente des BUS-Systems ist ein Repository zur sicheren Speicherung der Prüfinformationen. Diese Komponente ermöglicht, neben der reinen Speicherung von Dokumenten in Form des oben beschriebenen Datenformats, insbesondere die Speicherung und Abfrage von verschiedenen Dokumentversionen.

Das BUS-System kann insgesamt als Grundlage für die Errichtung der Dokumentationsinfrastruktur angesehen werden. Alle an einem Prüfvorgang beteiligten Dokumente werden über dieses BUS-System transportiert und Prozesse dadurch abgebildet.

## Werkzeuge für Prüfer und Hersteller

Die Entwicklung von Werkzeugen zur Verknüpfung mit dem BUS-System basiert auf den Ergebnissen des ersten Teilziels. Insbesondere sollen, von den zu entwickelnden Tools, Aufgaben unterstützt werden, die bei manueller Durchführung zu einem hohen Zeitaufwand führen.

Als Grundlage zur Implementierung derartiger Tools werden zunächst Verfahren entwickelt, die eine IT-Unterstützung wiederkehrender Aufgaben möglich machen. Beispielsweise könnte ein Verfahren zur Verknüpfung der Elemente von Spezifikationen mit Klassen oder Methoden in korrespondierenden Quelltexten den Aufwand, im Vergleich zur manuellen Suche während der Prüfung, stark reduzieren. Genauso könnte eine automatische Visualisierung der Änderungen zwischen verschiedenen Versionen von Quelltexten oder Spezifikationen Unterstützung für Prüfer bieten.

Für Hersteller ist vorgesehen, ein prototypisches Tool zu entwickeln, mit dem die Dokumentation zu einer Software zusammengestellt und bereits in das erforschte Datenformat inkl. aller semantischen Erweiterungen überführt werden kann, sodass Hersteller in frühen Phasen oder bereits vor dem Beginn von Prüfungen zielgerichtet relevante Informationen zusammenstellen und als Paket, beispielsweise per E-Mail, an Prüfer übermitteln können.

# 4 Methodik

## 4.1 Vorgehensweise

Der im Folgenden vorgestellte, in drei Phasen unterteilte, Lösungsansatz zur Erreichung der Zielsetzung aus Kapitel 3 basiert auf der Erforschung eines idealtypischen Dokumentformats zur Repräsentation von Prüfdokumentationen und der prototypischen Entwicklung einer Dokumentationsinfrastruktur. Abbildung 2 zeigt eine Übersicht über die geplanten Projektphasen.

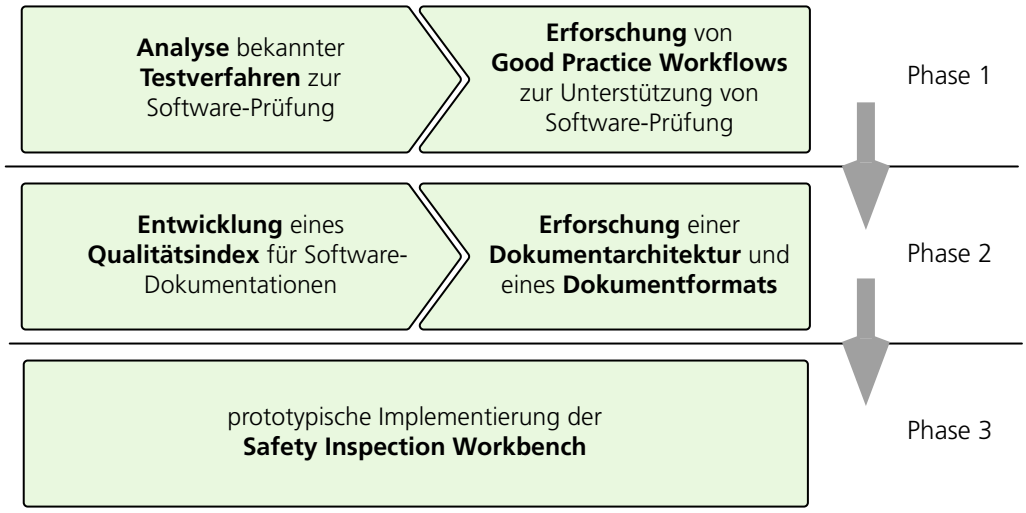



Abb. 2: Darstellung der Projektphasen

### Erste Phase

Die erste Phase des Forschungsprozesses gliedert sich in zwei Teilschritte auf. Zunächst werden dabei etablierte Testverfahren zur Prüfung von Software analysiert. Die für diesen Schritt erhobenen Informationen stammen im wesentlichen aus zwei Quellen:

- Verschiedene Normen, die Prüfprozesse teilweise oder vollständig beschreiben: Neben der IEC 61508-3 werden die IEC 61511, die IEC 62061 oder die DIN ISO/IEC 12119 und weitere relevante Normen bei der Erhebung berücksichtigt.
- Primärerhebungen in Prüfinstituten, wie dem IFA der DGUV und bei Herstellern von Produkten mit Relevanz für den Arbeitsschutz: Die Erhebung auf Seite der Prüfer soll den Grad der Akzeptanz verschiedener Arten von Software-Dokumentationen, sowie verschiedener etablierter Testverfahren aus Prüfsicht zeigen. Die Erhebung bei Herstellern soll vor allem typische Vorgehensweisen bei der Erstellung entsprechender Dokumentationen erfassen und gleichzeitig Anforderungen an mögliche Werkzeuge zur Vereinheitlichung der Vorgehensweisen zeigen. Alle anhand dieser Primär- und Sekundärerhebungen gewonnenen Informationen werden analysiert und die Ergebnisse in Form einer Methodensammlung gebündelt.

Im Rahmen der Auswertung und Beurteilung der Ergebnisse des ersten Teilschritts findet im Anschluss eine Erforschung von Good Practice Workflows zur Prüfung von Software statt. Hierbei werden konkrete

Workflows entwickelt, die einen Prüfprozess jeweils aus Sicht eines Prüfers und eines Herstellers modellieren und damit ganzheitlich betrachten. Insbesondere häufig wiederkehrende Teilaufgaben, die sich durch IT-Systeme teilweise oder vollständig automatisieren lassen, sollen in diesem Schritt identifiziert und dokumentiert werden. Beispielsweise ist die manuelle Zuordnung von einzelnen Spezifikationselementen zu konkreten Funktionen im Quelltext eine Aufgabe, die durch IT-Unterstützung vereinfacht werden könnte.

In dieser ersten Phase des Projekts finden also theoretische und praktische Untersuchungen zur Bildung einer Forschungsgrundlage statt.

## Zweite Phase

Ausgehend von den Ergebnissen der ersten Projektphase wird ein Qualitätsindex entwickelt, der Aussagen über die Güte von Formaten zur Dokumentation von Software-Prüfvorgängen erlaubt. Bei der Bewertung von Formaten sollen Vollständigkeit, Lesbarkeit, Art und Umfang der Meta-Informationen etc. berücksichtigt werden. Der erarbeitete Qualitätsindex wird benötigt, um entwickelte Datenformate hinsichtlich ihrer Tauglichkeit für den Einsatzzweck "Software-Prüfung" zu überprüfen. Der Qualitätsindex dient somit in erster Linie als Maßnahme zur Qualitätssicherung innerhalb des Projekts. Darüber hinaus kann der Index auch eingesetzt werden, um in aufbauenden Projekten proprietäre Dokumentformate hinsichtlich ihrer Importierbarkeit in SIWOB zu untersuchen.

Mit der Erforschung einer idealtypischen Dokumentarchitektur zur Abbildung aller benötigten und erzeugten Informationen eines Prüfprozesses findet der zentrale Forschungsschritt des Projekts statt. Eine solche Dokumentarchitektur beschreibt die Struktur und Art sämtlicher Informationen, die in einem Software-Prüfprozess benötigt werden. Darüber hinaus werden auch alle Meta-Informationen und Relationen zwischen Informationen identifiziert und beschrieben. Die Dokumentarchitektur muss in ihrem Aufbau flexibel genug sein, um auch Sonderfälle und Änderungen in Prozessen unterstützen zu können.

Ausgehend von der Dokumentarchitektur als Modell wird ein konkretes Datenformat entwickelt. Dieses ist als Instanz der Dokumentarchitektur ein in sich konsistentes und im weiteren Projektverlauf praktisch genutztes Format. Die Entwicklung des Formats findet iterativ anhand des entwickelten Qualitätsindex statt, sodass im Ergebnis der zweiten Phase des Forschungsvorhabens ein qualitätsgesichertes Datenformat zur Nutzung in SIWOB vorliegt.

## Dritte Phase

Um eine adäquate Verwaltung und Darstellung aller benötigten Informationen innerhalb eines Software-Prüfvorgangs durch ein IT-System zu ermöglichen, wird in der dritten Phase des Vorhabens eine geeignete technische Dokumentationsinfrastruktur konzipiert und prototypisch errichtet. Diese soll basierend auf aktuellen Techniken aus den Bereichen Web Services und BUS-Systemen entworfen und für den Umgang mit dem entwickelten Dokumentformat konfiguriert werden. Auf diese Weise wird eine Kapselung der einzelnen Teilkomponenten des Gesamtsystems ermöglicht, die zu einem hohen Grad von Skalierbarkeit, Flexibilität und Plattformunabhängigkeit führt. Das entwickelte BUS-System ist als technischer Kern des

SIWOB-Konzeptes zu verstehen, an den im Rahmen des Vorhabens erste Tools gekoppelt werden und der für Erweiterungen auch nach dem Projektende zur Verfügung steht. Mit der Erstellung dieses Systems geht die Entwicklung und Umsetzung eines geeigneten Sicherheitskonzepts einher.

Zu den Implementierungsarbeiten gehört ebenfalls die Entwicklung der in der Zielsetzung vorgestellten Tools für Hersteller und Prüfer. Tools, die für die Verwendung durch Prüfer entwickelt werden, lassen sich direkt mit dem oben beschriebenen BUS-System koppeln und werden bei der Realisierung der identifizierten Workflows entsprechend ihrer Funktion im Prüfprozess eingebunden. Zu diesen Tools gehört zum einen ein Repository, das u.A. Prüfdaten sicher speichert, sowie Unterstützungswerkzeuge, wie ein Tool zur Verknüpfung von Spezifikationselementen mit Funktionen in Quelltexten.

Das für Hersteller entwickelte Tool zur Unterstützung bei der Spezifikationserstellung erhält keine direkte Kommunikationsmöglichkeit mit dem BUS-System. Daten, die mit Hilfe dieses Tools erzeugt werden, können Prüfern als Pakete z.B. via Email übermittelt werden.

## 4.2 Forschungskooperation und Qualitätssicherung

Zwischen dem Institut für Sicherheitsforschung der Hochschule Bonn-Rhein-Sieg und dem IFA der DGUV besteht eine enge Zusammenarbeit. Das IFA nimmt als akkreditierte Prüfstelle, die sich regelmäßig mit der Prüfung von Software beschäftigt, eine beratende Funktion ein.

Im Rahmen der Durchführung sind dauerhafte Maßnahmen zur Qualitätssicherung geplant:

- Einrichtung eines wissenschaftlichen Begleitkreises zur regelmäßigen Diskussion der gewonnenen Erkenntnisse. Diskussion der angestrebten Entwicklungen zur zielgerechten Steuerung der wissenschaftlichen und praktischen Relevanz der Forschungsergebnisse.
- Abarbeitung des in Abschnitt 4.4 abgebildeten Arbeitsplans.
- Festlegung und Überprüfung von angemessenen Qualitätszielen zu jeder Aktivität des Arbeitsplans.
- Kritische Prüfung der erreichten Qualitätsziele zu jedem Meilenstein des Arbeitsplans.
- Unregelmäßige Fachgespräche mit Vertretern des IFA, des Begleitkreises oder Herstellern zur Wahrung der wissenschaftlichen und praxisrelevanten Ausrichtung des Vorhabens.
- Bei den Implementierungsaktivitäten: Wahl eines geeigneten Vorgehensmodells (Bspw. V-Modell) zur Sicherstellung der Ergebnisqualität und zur Ermöglichung von Folgeaktivitäten durch strukturierte, dokumentierte und nachvollziehbare Implementierungskonzepte.

Die geplante Erhebung von Anforderungen und Bedürfnissen, sowie die Befragung zur Akzeptanz verschiedener Ansätze im Rahmen der Analysephase, ermöglicht eine direkte Einbeziehung von Prüfern und Herstellern. Darüber hinaus findet eine indirekte Einbeziehung dieser Akteure durch zielgruppengerechte Forschung anhand von praxisnahen Fallbeispielen statt.

Durch die Einrichtung eines wissenschaftlichen Begleitkreises mit Beginn des Projekts werden direkt und indirekt beteiligte Parteien zur Teilnahme am Forschungsvorhaben eingeladen. Dafür ist geplant, Vertre-

ter des IFA, interessierte Hersteller von Maschinen und Sicherheitsbauteilen, Repräsentanten betroffener UV-Träger sowie Wissenschaftler mit geeigneter Expertise zur Teilnahme am wissenschaftlichen Begleitkreis einzuladen.

Dem Begleitkreis werden regelmäßig Informationen aus dem laufenden Projekt (z.B. per Email-Verteiler) zur Verfügung gestellt. Um das Fachwissen und mögliche inhaltliche Ideen der Mitglieder des Begleitkreises für das Projekt nutzbar machen zu können, werden im Verlauf des Forschungsvorhabens mehrere persönliche Treffen anberaumt, in denen abgeschlossene Teilergebnisse und geplante Folgeschritte diskutiert werden können. Die Teilnehmer des Forschungsbegleitkreises haben somit die Möglichkeit Empfehlungen auszusprechen.

## 4.3 Zeitplan

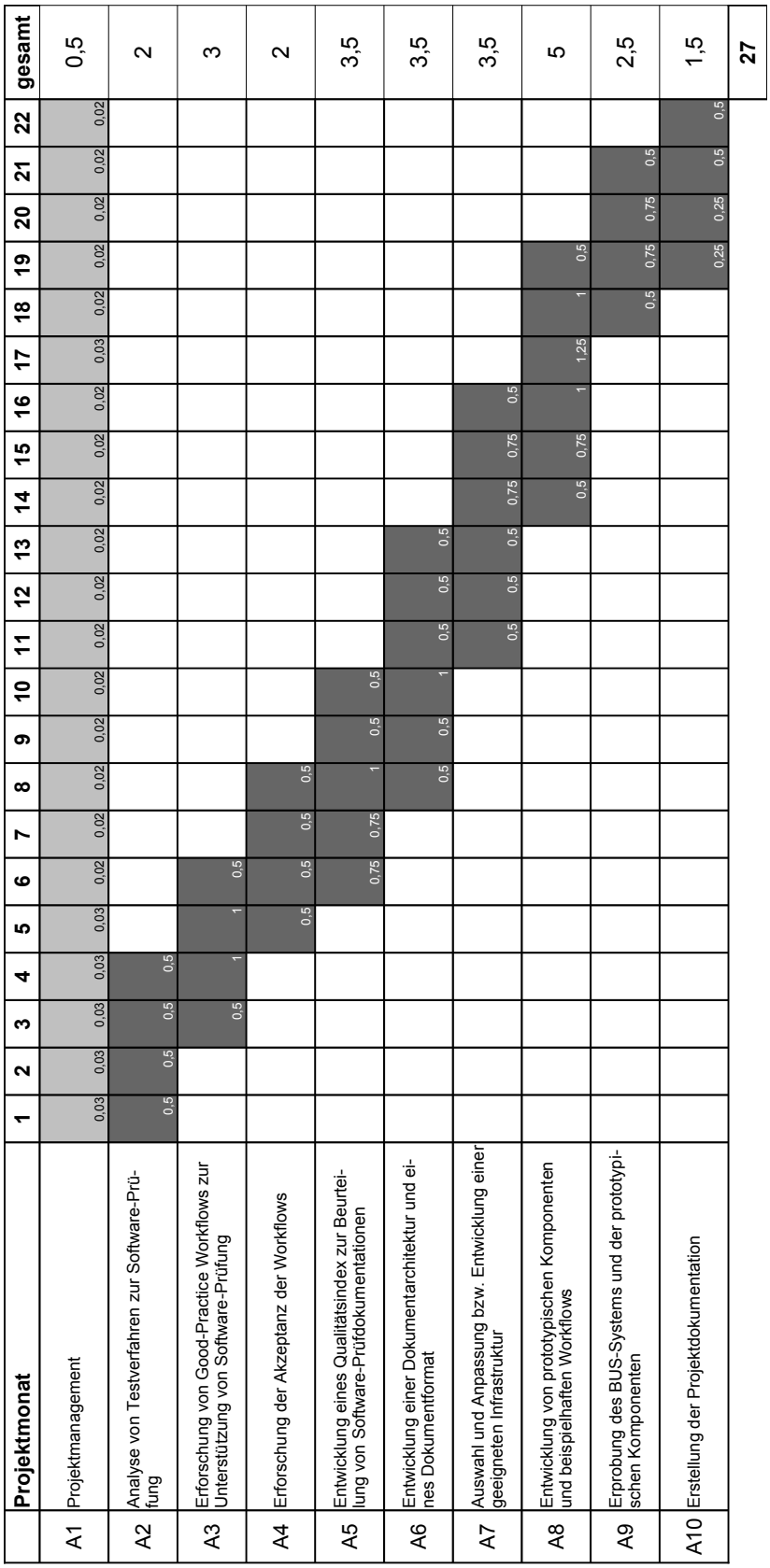



Abb. 3: Zeitplan über die Projektaktivitäten

## 4.4 Arbeitsplan

### A1: Projektmanagement

Kontinuierliche Koordination der Projektaktivitäten sowie Beobachtung aller Forschungsaktivitäten im Themenbereich. Bei jedem Meilenstein Überprüfung des Projektplans.

### A2: Analyse von Testverfahren zur Software-Prüfung

Intensive Recherchearbeiten zur Identifikation aktueller Entwicklungen bei Vorgehensweisen zur Prüfung von Software, zur Verwaltung und Versionierung von Informationen sowie zu verwendbaren (OpenSource-) Produkten und alternativen Ansätzen zur formalen Repräsentation von technischen Dokumentationen. Eingehende Analyse bekannter und standardisierter Testverfahren zur Prüfung von Software, zur Sicherstellung und Steigerung der Prüfungsqualität.

### A3: Erforschung von Good Practice Workflows zur Unterstützung von Software-Prüfung

Auf Basis der vorangegangenen Recherchen findet eine Erforschung und Entwicklung von erfolgversprechenden Workflows bei Prüfprozessen statt. Bei der Erstellung dieser Good Practice Workflows werden die Sichtweisen sämtlicher an einem Prüfprozess beteiligten Akteuere berücksichtigt. Dafür findet neben einer Primärerhebung bei den beteiligten Akteuren auch eine eingehende Untersuchung der zuvor identifizierten Vorgehensweisen der Software-Prüfung und des Produktmanagements statt.

Eine anschließende Analyse und Bewertung der Workflows, hinsichtlich der Realisierbarkeit durch ein IT-System, filtert unter den erstellten Workflows Geeignete heraus.

### A4: Erforschung der Akzeptanz der Workflows

Im Zuge der Erhebungen der vorangegangenen Aktivität findet ebenfalls eine Akzeptanzbefragung zu den identifizierten Good Practice Workflows statt. Im Rahmen dieser Aktivität werden in erster Linie Workflows mit hoher Akzeptanz für einen möglichen Einsatz in der Praxis identifiziert.

Zu diesem Schritt gehört insbesondere die Festlegung auf zunächst eine konkrete Art von Software, die besondere praktische Relevanz hat (bspw. Embedded-Software oder SPS-Software) und im Rahmen des Forschungsprojekts als Beispiel zur Demonstration der Machbarkeit des Ansatzes eingesetzt wird. Darauf aufbauende Aktivitäten (z.B. Implementierungen) berücksichtigen stets diese Beispielhaftigkeit und erhalten dabei die Erweiterbarkeit des Systems auf andere Software-Arten.

Die durchgeführten Arbeiten finden aufgrund von voraussichtlichen Wechselwirkungen größtenteils zeitgleich mit Aktivität A3 statt, resultieren allerdings in einem gesonderten Ergebnis.

### A5: Entwicklung eines Qualitätsindex zur Beurteilung von Software-Prüfdokumentationen

Diese Aktivität dient als methodische Vorbereitung der Entwicklungsarbeiten. Durch die Festlegung von Kriterien zur Bestimmung der Qualität von Dokumentformaten zur Abbildung von Software-Prüfvorgängen werden die darauf folgenden Forschungsaktivitäten messbar. Auf Basis der Ergebnisse dieser Aktivi-

tät wird im Verlauf des Projekts die Verifikation gewählter Methoden und Implementierungen ermöglicht.

### A6: Entwicklung einer Dokumentarchitektur und eines Dokumentformats

Auf Basis der in Aktivität A5 entwickelten Qualitätskriterien und in Aktivität A4 durchgeführten Akzeptanzbefragung, wird ein formales Dokumentmodell konzipiert, dass sämtliche Eigenschaften und Inhalte eines Prüfvorgangs von Software abbildet. Dabei werden alle notwendigen Dokumentarten, semantische Verknüpfungen sowie Meta-Informationen definiert und in Form einer Dokumentarchitektur beschrieben.

Darauf aufbauend wird für sämtliche Dokumenttypen, die innerhalb eines Prüfvorgangs Verwendung finden ein geeignetes Datenformat gewählt oder eigens konzipiert. Während graphische Dokumentteile beispielsweise durch geeignete Bildformate repräsentiert werden, wird für textbasierte Dokumente ein Schema in einer Auszeichnungssprache definiert. Insgesamt wird ein Container-Format entwickelt, das die verschiedenartigen Dokumenttypen innerhalb eines Prüfvorgangs in einer einheitlichen Struktur zusammenfasst.

### A7: Auswahl und Anpassung bzw. Entwicklung einer geeigneten Infrastruktur

Basierend auf den Ergebnissen aus Aktivität A2 und A3 findet eine Bewertung verschiedener BUS-Systeme (bspw. verschiedene Enterprise-Service-BUS-Systeme) statt. Das bestgeeignete System wird durch entsprechende Anpassungen hinsichtlich der Anforderungen, die sich aus der Dokumentarchitektur (A6) ergeben, zu einem Prototypen des zentralen BUS-Systems der Workbench (SIWOB) weiterentwickelt. Alternativ zur Nutzung einer bestehenden Software findet eine Neuentwicklung entsprechend der Anforderungen statt.

Das resultierende BUS-System wird prinzipiell in der Lage sein, jegliche Art von Prüfprozessen zu unterstützen. Durch die Konfiguration der Workflows in der nächsten Aktivität wird zunächst die Software-Prüfung als Teil einer exemplarischen Baumusterprüfung realisiert.

Als Teil dieser Aktivität wird außerdem ein individuelles Sicherheitskonzept entwickelt. Die Entwicklung erfolgt hierbei auf Basis der Empfehlungen des BSI und umfasst unter anderem Aspekte wie die Authentifizierung, Datensicherung, etc.. Ein BSI konformes IT-Grundschutz Konzept beinhaltet die Identifikation individueller Anforderungen und eine anwendungsfallbezogene Ergreifung zielgerichteter Maßnahmen.

### A8: Entwicklung von prototypischen Komponenten und beispielhaften Workflows

Im Rahmen dieser Aktivität werden notwendige Komponenten zur Erweiterung des BUS-Systems hin zur geplanten Workbench entwickelt und entsprechend angebunden. Die folgenden Komponenten sind für eine Unterstützung des Software-Prüfprozesses mindestens notwendig:

- Repository
  Zentrale Komponente für Verwaltungsaufgaben von Prüfvorgängen, wie Hinzufügen von Projekten, Statusänderungen, Hinzufügen/Ändern von Dokumenten oder Kommentierungen, Abruf von Meta-Informationen etc.. Sämtliche Operationen innerhalb des Repository werden dabei gegenüber möglichen

Dateninkonsistenzen und Fehlbedienungen abgesichert. Dazu gehört auch eine integrierte Rechteverwaltung, die Zugriffe auf Daten ausschließlich berechtigten Akteueren ermöglicht.

- Render-Einheit

  Zur Darstellung und Ausgabe von Teildokumenten eines Prüfvorgangs (beispielsweise Spezifikationen) werden die im Repository verwalteten Informationen zur Laufzeit in ein gängiges Format (bspw. PDF) überführt. Hierbei werden neben Textformatierungen auch Grafiken und Links generiert, die Zusammenhänge innerhalb von Prüfunterlagen verdeutlichen.

- User Interface für Prüfer

  Die Benutzungsschnittstelle für Prüfer stellt eine geeignete Bedienoberfläche zur direkten Interaktion mit SIWOB bereit. Prüfer haben hier insbesondere die Möglichkeit eingereichte Unterlagen verschiedener Dokumenttypen an einer zentralen Stelle einzusehen, zu kommentieren, Rückfragen zu formulieren sowie Arbeitspakete an den Hersteller per Email zu senden.

- Unabhängige Anwendung für Hersteller

  Die Benutzungsschnittstelle für Hersteller wird in Form einer gekapselten und nicht direkt mit SIWOB verbundenen Software realisiert. Sie ermöglicht Herstellern eine Unterstützung bei der Erzeugung von Software-Dokumentationen inkl. der Möglichkeit zur Eingabe von unterstützenden semantischen Informationen. Beispielsweise wird es möglich sein, einzelne Spezifikationselemente direkt mit Klassen oder Methoden des Quelltextes zu verknüpfen. Der Datenaustausch zu SIWOB findet durch geeignet verschlüsselte Austauschdateien z.B. via Email statt.

  Hersteller können so auch Anmerkungen und Kommentare von Prüfern entgegennehmen und ihrerseits mit Änderungen darauf reagieren.

Neben den beschriebenen Funktionen werden innerhalb dieser Aktivität auch die in Aktivität A3 entwickelten Workflows implementiert. Zu diesem Zweck werden die notwendigen Arbeitsschritte mittels einer geeigneten Beschreibungssprache für die Nutzung mit SIWOB realisiert. Die Implementierung der Workflows findet zeitgleich mit der Implementierung der oben aufgeführten Komponenten statt.

### A9: Erprobung des BUS-Systems und der prototypischen Komponenten

Zur Erprobung des Systems werden beispielhafte Software-Prüfprozesse verwendet, die im ersten Schritt in das prototypische Gesamtsystem eingepflegt werden. Anhand der in Aktivität A5 entwickelten Qualitätskriterien für technische Unterlagen wird im Anschluss die Darstellung der Informationen in den Benutzungsschnittstellen erprobt. In einem dritten Schritt werden zu den verschiedenen Beispielen Prüfprozesse aus Sicht eines Prüfers und aus Sicht des Maschinenherstellers unter Verwendung des Systems durchlaufen und die Ergebnisse dokumentiert. In dieser Projektphase soll der entwickelte Prototyp untersucht und die grundsätzliche Funktionsfähigkeit belegt werden.

Die zur Erprobung des Prototypen notwendigen, praxisnahen Fallbeispiele werden anhand exemplarischer Prüfunterlagen durchlaufen. Die verwendeten Unterlagen (Spezifikationen, Quelltexte etc.) werden dabei von anderen Forschungsprojekten der Hochschule Bonn-Rhein-Sieg zur Verfügung gestellt.

### A10: Erstellung der Projektdokumentation

Die im Rahmen dieses Projekts entwickelte universelle Datenplattform wird der in IEC 61508 beschriebenen Toolklasse T1 entsprechen. Die Safety Inspection Workbench dient zur Verwaltung von Dokumenten und Informationen. Bei diesen Vorgängen finden keine Änderungen an Daten (z.B. Quelltexten) statt, die zu fehlerhaften Interpretationen führen könnten. Für die Safety Inspection Workbench wird eine umfangreiche Dokumentation erstellt, um damit eine möglichst kurze Einarbeitungszeit in das System zu erreichen.

## 4.5 Meilensteine

Im folgenden Kapitel werden die im Rahmen des Projekts angestrebten Teilziele in Form von Meilensteinen vorgestellt. Während des gesamten Projekts wird durch das Projektmanagement sichergestellt, dass die geplanten Ziele erreicht werden. Die Planung des vorliegenden Forschungsvorhabens sieht eine Aufteilung des Gesamtziels in vier Meilensteine vor.

### M1: Theoretische Grundlagen

Dieser erste Meilenstein besteht aus einer Dokumentation von analysierten Testverfahren zur Prüfung von Software, die als Zusammenfassung sowohl über die einsetzbaren Methoden als auch über vorhandene Software-Lösungen dient.

### M2: Konzepte

Die Ergebnisse der Arbeiten zur Erstellung der zentralen Konzepte und Methoden des Projekts entsprechenden dem zweiten Meilenstein und sollen durch eine Akzeptanzbefragung belegt werden. Dabei wird in einem Dokument u.a. ein Index zur Ermittlung der Qualität von Formaten technischer Unterlagen erarbeitet. Mit der Entwicklung eines geeigneten Modells werden Dokumente erzeugt, die zentrale fachliche Fragestellungen aufgreifen und konkrete Lösungsansätze beschreiben.

### M3: Konkrete Software

Die Entwicklung einer prototypischen Implementierung und Erprobung des geplanten Systems besteht aus mehreren Teilaufgaben und stellt insgesamt den dritten Meilenstein dar. Die Vorgehensweise ist inkrementell und kann in die folgenden Komponenten aufgeteilt werden:

- BUS-System mit erforschter Dokumentarchitektur
- Repository zur Repräsentation und Speicherung sämtlicher Inhalte von Prüfprozessen
- Bedienoberfläche für Prüfer zur Dateneingabe und Verwaltung von Prüfvorgängen

### M4: Dokumentation

Der letzte Meilenstein des Projekts besteht aus zwei Teilergebnissen. Zum Einen kann mithilfe einer umfangreichen Dokumentation der Software die Bedienung erlernt werden, wodurch kurze Einarbeitungszeiten entstehen. Zum Anderen ermöglicht die Dokumentation der erstellten Software und verwendeten Methoden eine leichtere Weiterentwicklung des Systems bzw. Anschlussforschungen.

# 5 Ergebnisse

## 5.1 Stand der Wissenschaft

Der Stand der Wissenschaft[1] und Forschung umfasst aktuelle Entwicklungen, die für das Forschungsprojekts SIWOB relevant sind. Insbesondere sollen die theoretischen Grundlagen sowie die Techniken und vorhandene Best Practices für die Realisierung und Verwendung der prototypisch zu entwickelnden Werkzeuge erläutert werden. Zunächst werden hierzu die theoretischen Grundlagen beschrieben, sodass wesentliche Inhalte für das Forschungsprojekt deutlich werden. Weiterhin werden Techniken und vorhandene Best Practices für die prototypische Entwicklung von Werkzeugen zur Unterstützung einer Software-Prüfung vermittelt. Unter dieser Betrachtung werden insbesondere theoretische Grundlagen sowie Techniken und vorhandene Best Practice Ansätze näher erläutert, welche großes Potential für die Unterstützung der Software-Prüfung bieten.

### 5.1.1 Theoretische Grundlagen

Mit Hilfe der theoretische Grundlagen werden essentielle Inhalte für das SIWOB Forschungsprojekt dargestellt. Vor allem ist die Thematik der verwendeten Standards und Normen von großer Wichtigkeit für die Verwendung innerhalb des Projekts SIWOB, da grundlegende Richtlinien für die Prüfung von Software vorgegeben sind. Ebenso sind in diesem Kontext die grundsätzlichen Anforderungen an Software für eine gewissenhafte Konzeption und Entwicklung von hoher Bedeutung. Darüber hinaus soll ein grundlegendes Verständnis für den benutzerfreundlichen Umgang von Software aufgezeigt und elementare Prinzipien von Systemarchitekturen vorgestellt werden.

#### 5.1.1.1 Standards und Normen

Es existieren verschiedene Workflows und Ansätze zum Vorgehen zu testender Anwendungen. Weit verbreitet und bekannt ist der IEEE 829 "Standard for Software Test Documentation", der Institute of Electrical and Electronics Engineers (IEEE) der erstmals im Jahr 1983 etabliert wurde und bereits zahlreiche Aktualisierungen, die Letzte im Jahr 2008, erfahren hat. Die IEEE 829 ist ein Standard, der Vorlagen zur Dokumentation von Tests bereitstellt und als bewährter Standard häufig in industriellen Projekten eingesetzt wird. Daneben existieren der britische Standards (BS) BS 7925, von dem vor allem Teil 1 "Software Testing - Vocabulary" von großer Relevanz ist, da in diesem das Vokabular für einen Test standardisiert wird, wodurch eine einheitliche Verständigungsebene möglich wird. In Teil 2 "Software Testing - Software Component Testing" werden zusätzlich Methoden erläutert, die bei der Testfallgenerierung unterstützen sollen. Die IEEE 1008 "Standard for Software Unit Testing" aus dem Jahr 1987 unterstützt im Sinne des Testens durch eine Vorgehensbeschreibung beim Unit-Test. Neben diesen Standards und Normen existieren viele Weitere, zum Teil auch sehr spezialisierte, wie etwa die IEC 61508 "Functional Safety of Electrical/Electronic/Programmable Electronic Safety-related Systems" der International Electrotechnical Commission (IEC). In der IEC 61508 wird die Struktur, das Vorgehen und Methoden für den Test von

1. Innerhalb des Abschlussberichts wird die Formulierung "Stand der Wissenschaft" mit der Formulierung "Stand der Technik" sowie "Stand der Wissenschaft und Technik" gleichgesetzt.

Software im Falle sicherheitsrelevanter Systeme beschrieben, die für viele Hersteller eine bindende Norm darstellt und vollständig eingehalten werden muss, damit diese Produkte überhaupt erst in den Markt eingeführt werden können. Aus dem Grund der Vielzahl der Normen und Standards, die aus dem jeweiligen Bedarf zu verschiedenen Zeitpunkten entwickelt wurden und verschiedenste Aspekte im Software-Test abdecken, soll die ISO/IEC 29119, der Internationalen Organisation für Normung (ISO), diese Rolle als Standard übernehmen. Diese Norm soll bisheriges Wissen verschiedenster Normen und Standards inkludieren, sodass ein neuer einheitlicher Standard definiert werden kann, der Testbegriffe, Testprozesse, Testdokumentationen und Testtechniken vereinheitlicht. Dadurch soll es ermöglicht werden einen Standard bereitzustellen, der jegliches Software-Projekt unterstützen kann.[2]

## IEEE 829 - Dokumentation von Tests

Ein möglicher Workflow wird in der Abbildung 4 gezeigt, der sich am IEEE 829 Standard orientiert. Der Standard definiert eine Menge von strukturieren Dokumenten, wie etwa den Testplan, die für eine Testdürchführung relevant sind. Allerdings stellt der IEEE 829 Standard nur einen Vorschlag dar, der keine gesetzliche Bindung vorsieht und somit nicht vollständig eingehalten werden muss. Allerdings wird die Einhaltung des Standards empfohlen.[3]

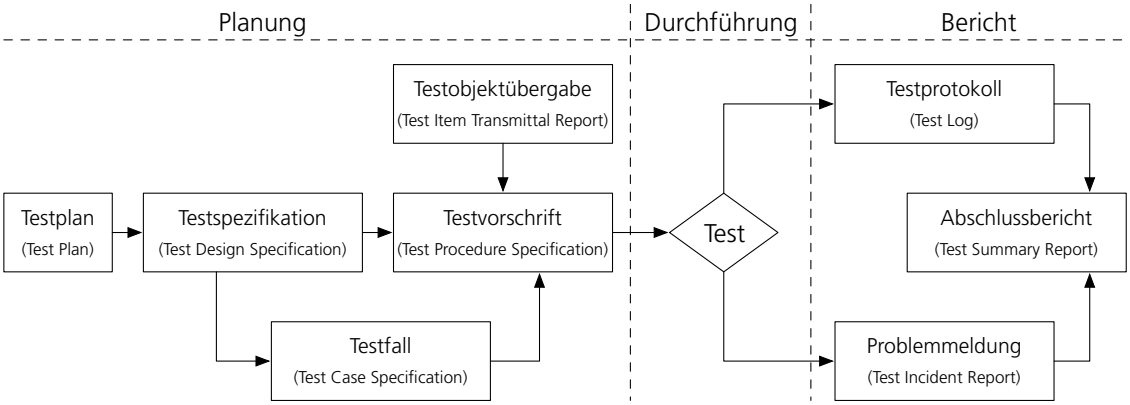



Abb. 4: IEEE 829 Zusammenhänge und Abhängigkeiten der Testdokumente[4]

Der Standard IEEE 829 unterscheidet in die Planungs- und die Berichtsphase, in denen gemäß des Standards strukturierte Dokumente im Verlauf des Projekts erstellt werden müssen. Hierbei werden in der Planungsphase der Testplan (Test Plan), die Testspezifikation (Test Design Specification), Testfälle (Test Case Specification), die Software-Übergabe (Test Item Transmittal Report) und Testvorschrift (Test Procedure Specification) notiert. In der Berichtsphase werden anschließend Testprotokolle (Test Log), Problemmeldungen (Test Incident Report) erfasst und der Abschlussbericht (Test Summary Report) formuliert.[5]

2. vgl. (Dussa-Zieger, 2011, S. 6)
3. vgl. (Sneed, Baumgartner, & Seidl, 2012, S. 85)
4. vgl. (Bath & McKay, 2011, S. 23)
5. vgl. (Frühauf, Ludewig, & Sandmayr, 2007, S. 72)

In der Phase der Testplanung wird ein Konzept erstellt, durch das die Vorgehensweise des Tests grob beschrieben wird und aus dem hervorgeht, welche Ressourcen für die Durchführung, wie etwa Ausstattung, Personal, etc., benötigt werden. Weiterhin werden die spezifischen Strategien für den Test gewählt, welche Merkmale welcher Software-Komponenten genau zu testen sind. Darüber werden Akzeptanzkriterien formuliert, die erfüllt sein müssen, damit die Software akzeptiert wird und vor allem was zu tun ist, wenn der Test unterbrochen wird und welche Maßnahmen ergriffen werden müssen, damit dieser wieder fortgesetzt wird. In der Testspezifikation werden gemäß der Software-Spezifikation die Bestandteile der Vorgehensweise weiter verfeinert und präzisiert. Hier werden die Funktionen identifiziert, die durch den späteren Test geprüft werden sollen und welche Testfälle bzw. Abläufe notwendig sind. Auch die Definition der Kriterien, unter denen die einzelnen zu testenden Funktionen der Software bestanden bzw. verfehlt wurden, muss hier formuliert werden. An die Testspezifikation knüpft sich zum einen die Formulierung der Testfälle, die anhand vorliegender Anforderungen sowie technischer und fachlicher Anforderungen formuliert werden. Zu jedem formulierten Testfall werden die Eingabedaten, die erwarteten Ausgaben und sofern dies möglich ist die Abhängigkeiten zu anderen Testfällen notiert. Zum anderen wird in der Testvorschrift die Durchführung und die Reihenfolge der Testfälle, welche Anforderungen an diese gestellt werden beschrieben und wie die Implementierung in den Test erfolgen soll. Eine besondere Rolle nimmt das Dokument der Testobjektübergabe ein, welches notwenig wird, sofern eine Entwicklung und ein Test durch mehrere Teams geplant wird. In diesem Dokument werden alle Informationen der Teilentwicklungen eines jeden Teams anhand verschiedener Informationen, wie etwa Versionsnummer, Speicherort, Status, usw. zusammengeführt, sodass ein Gesamtüberblick aller zu testenden Funktionen entsteht. Liegen alle Informationen der Planungsphase vor, so kann der eigentliche Test in der Phase der Durchführung gemäß der Planungen erfolgen aus denen detaillierte Berichte resultieren. Im Testprotokoll werden in diesem Fall alle Informationen gesammelt, die während der Durchführung eines Testfalls entstehen, wie beispielsweise die verwendeten Objekte und Umgebungen sowie Ergebnisse und vieles mehr. Besonders große Beachtung fällt hier auf außergewöhnliche Ereignisse und deren Auswirkung auf die Funktionalität, die mit dem Fehlerprotokoll verknüpft werden. Sollte eine Abweichung identifiziert werden, bedarf es einer detaillierten Protokollierung aller verfügbaren Informationen, die dazu beitragen können die entsprechende Anomalie in der betroffenen Funktion detektieren zu können. Anschließend müssen die Fehler behoben und die Tests in einer weiteren Durchführungsphase wiederholt werden. Diese Schritte können so lange iterativ durchgeführt werden, bis keinerlei Fehler mehr entdeckt werden oder die Prüfer mit dem Ergebnis des Tests einverstanden sind. Alle Testaktivitäten werden schlussendlich in einem zentralen Abschlussbericht gebündelt, durch den eine Zusammenfassung des Prüfumfangs, der Ergebnisse und der ausgeführten Aktivitäten möglich wird. Vor allem kann nachgeprüft werden, ob die Einschätzungen von Zeit, Kosten, Testmethoden, Personal, etc. aus der Planungsphase konform mit dem tatsächlichen Aufwand sind, wodurch zukünftige Testpläne realistischer erstellt werden können.[6]

6. vgl. (Bath·&·McKay,·2011,·S.·20ff.)

## IEC 61508-3 - Software-Anforderungen

Der Standard IEC 61508 liefert eine Vorlage bei der Entwicklung sicherheitskritischer Systeme und definiert darin Vorgaben für den Software- und Systemtest, an die sich jeder Hersteller halten muss, sofern das entwickelte Produkt eine potentielle Gefahr für den Benutzer darstellen könnte. Vor allem im dritten Teil dieser Norm wird Bezug auf die Entwicklung von Software genommen, in der ein standardisiertes Schema (siehe Abbildung 5) vorgegeben wird.[7]

Der Software-Sicherheitslebenszyklus aus dem dritten Teil der Norm gibt die Schritte vor, die eingehalten werden müssen, wenn sicherheitsrelevante Software im Zusammenhang mit Hardware erstellt wird. Die Entwicklung, sowie Dokumentation muss hierbei kongruent erfolgen.[8] Das V-Modell dient hierbei als Basis für die Software-Entwicklung und deckt sich mit dem Software-Sicherheitslebenszyklus.[9]

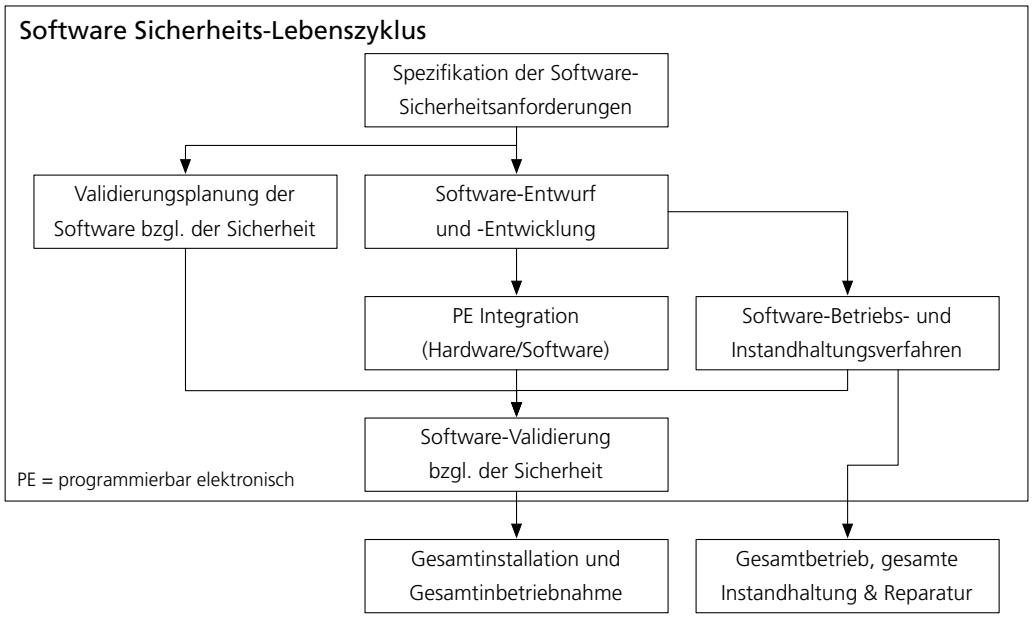



Abb. 5: Software-Sicherheitslebenszyklus als Teil des gesamten Sicherheitslebenszyklus[10]

Zuerst ist es notwendig, die Sicherheitsstufe des Produktes gemäß der Security Integrity Level (SIL) Definitionen einzuordnen, da sich anhand dieser Stufe der Aufwand für die Software-Prüfung und der damit verbundenen Dokumentation richtet. Hierbei kann festgehalten werden, dass mit steigendem SIL das Kritikalitätsniveau eines System steigt und somit auch der Aufwand. An der Sicherheitsstufe orientiert sich beispielsweise die Formulierung und Verfeinerung der Spezifikation der Software-Sicherheitsanforderungen, die neben Verständlichkeit, Präzision, Testbarkeit und weitere Kriterien erfüllen muss. In der Phase des Software-Entwurfs und der Software-Entwicklung werden alle Schritte gemäß der formulierten Anforderungen ausgerichtet. Es wird eine Architektur entwickelt, mit allen enthaltenen System- und Modul-

7. vgl. (Liggesmeyer, 2009, S. 386)
8. vgl. (Börcsök, 2011, S. 414)
9. vgl. (BSI, 2010, S. 17)
10. in Anlehnung an (Börcsök, 2011, S. 414)

komponenten, die den Sicherheitsansprüchen genügt und somit einen definierten Rahmen der Fehlertoleranz einhalten muss. Basierend auf den Entwürfen findet die Entwicklung der Software statt, wobei die Sicherheitsstufe Maßnahmen für das Vorgehen definiert, wie etwa eine stark strukturierte, wodurch Software-Module losgelöst getestet werden können, oder die defensive Programmierung, die Maßnahmen vorsieht, die das System in einen definierten Zustand überführen. Weiterhin finden angemessene Modul- und Integrationstests statt, in der Fehler identifiziert werden sollen. Sofern Fehler protokolliert werden, müssen diese ausnahmslos beseitigt werden, gegebenenfalls in mehreren Iterationen, bevor eine Integration von Software und Hardware erfolgt. An dieser Stelle wird durch weitere spezielle Integrationstest, wie etwa Funktions-, Leistungstests oder Black-Box-Tests, überprüft, ob die Software und Hardware zusammen funktionieren, da eine Simulation von komplexer Hardware (bspw. CPU) nicht möglich ist. Im letzten Schritt findet eine Validierung gegenüber den Anforderungen aus der Entwurfsphase statt durch die schlussendlich belegt werden soll, sofern alle vorherigen Schritte ordnungsgemäß erfüllt wurden, dass die entwickelte Software auch die Arbeiten verrichtet, die zuvor formuliert wurden.[11]

Durch die IEC 61508 wird eine umfangreiche Dokumentation in jedem einzelnen Abschnitt und jeder Iteration verlangt. Die Vorgaben dieser Norm müssen strickt eingehalten werden. Welche Informationen allgemein und vor allem im Zusammenhang mit Software-Prüfung dokumentiert und vorgelegt werden müssen, wird ausführlich in dieser Norm formuliert.[12]

Für jede Phase ist detailliert beschrieben, welche Abhängigkeiten und Anforderungen an die Dokumentation bestehen. Vor allem wird definiert, welche Informationen als Grundlage notwendig sind, damit ein Schritt im Software-Sicherheitslebenszyklus durchgeführt werden kann, und welche Dokumente als Output der betroffenen Phase vorliegen müssen.[13]

Eine weitere Standardisierung findet durch eine ausführliche Anleitung zur Auswahl von Techniken und Maßnahmen im Zusammenhang mit dem Software-Sicherheitslebenszyklus statt. Diese Anleitung gibt Techniken und Maßnahmen vor, wie etwa Spezifikationsanforderungen oder Software- und Designmethoden, die im Verlauf des Lebenszyklus zu berücksichtigen sind. In diesem Zusammenhang definiert das SIL welche Methoden zwingend berücksichtigt werden müssen und somit auch letztendlich die Komplexität, bzw. die Dauer den der Lebenszyklus in Anspruch nimmt.[14]

### ISO/IEC 29119 - Software Testing

Die ISO/IEC 29119 ist eine internationale Norm, die von der Arbeitsgruppe 26 des Joint Technical Committee seit dem Jahr 2008 entwickelt wird. Nachdem alle Prozessstufen der Entwicklung für eine neue Norm durchlaufen sind, soll gemäß der definierten Projekt Timeline eine Veröffentlichung der vollständigen ISO/IEC 29119 im Verlauf des Jahres 2013 erfolgen. Durch die Veröffentlich ist jedoch noch keine

11. vgl. (Löw, Pabst, & Petry, 2010, S. 99ff.)
12. vgl. (Liggesmeyer, 2009, S. 386)
13. vgl. (BSI, 2010, S. 18ff.)
14. vgl. (BSI, 2010, S. 47)

Anwendung durch Unternehmen oder Organisationen gewährleistet. Bis diese die Norm auch tatsächlich in Gebrauch nehmen wird noch weitere Zeit vergehen. Weiterhin ist auch noch unklar, ob die neue Norm allen Anforderungen gerecht werden kann, da der Zeitraum der Entwicklung als erheblich betrachtet werden kann, in der sich auch andere Normen weiterentwickelt haben.[15]

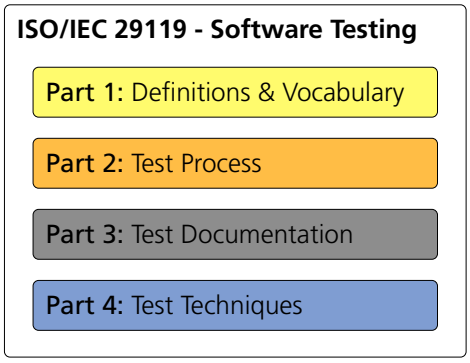



Abb. 6: Die vier Teile der ISO/IEC 29119

Die ISO/IEC 29119 "Software Testing" gliedert sich in vier große Bereiche (siehe Abbildung 6) auf, den Definitionen und Vokabular (Part 1), den Testprozessen (Part 2), der Testdokumentation (Part 3) und den Testtechniken (Part 4). An dieser Stelle wird die Absicht dieser umfangreichen Norm, dem inkludieren bisheriger Standards, sehr deutlich. Als Beispiel kann hier die IEEE 829 genannt werden, die vollständig im dritten Teil dieser Norm, der Testdokumentation, aufgehen soll. Mit Hilfe der neuen ISO 29119 soll der Testablauf besser strukturiert werden. Beispielsweise sieht der Testprozess eine Aufteilung in drei große Bereiche vor, der Organisation, dem Management von Testprozessen und den Testprozessen selbst. Teilweise findet eine weitere Aufteilung dieser Bereiche und Untergruppen statt, in denen dann detailliert das Vorgehen und die Reihenfolge durchzuführender Aktivitäten beschrieben werden. Im Falle der Testprozesse wird hier für die notwendigen Testmethoden das Design, bzw. die Implementierung definiert, in welcher Testumgebung diese ausgeführt werden sollen und vor allem welche Informationen dokumentiert werden müssen. Eine Beschreibung dieser Norm kann an dieser Stelle nur grob erfolgen, da der Status der Veröffentlichung einen Zugriff von nicht Gremiumsangehörigen im jetzigen Status einschränkt.[16]

### 5.1.1.2 Spezifikationsgrundlagen

Die Erstellung einer Spezifikation wird durch unterschiedliche Normen standardisiert, die Vorgaben über die Struktur und deren Inhalte definiert. Im Zusammenhang mit der Erstellung einer Spezifikation für sicherheitsrelevante Bauteile ist die sicherheitsspezifische Norm Europäische Norm (EN) ISO 13849 zu nennen, insbesondere Teil 1, die allgemeine Gestaltungsvorgaben definiert, die unter anderem die Software mit einschließt. Daneben existiert die IEEE 830-1998, die einen Standard zur Spezifikation von Software liefert. Aufgrund des Fokus des Projekts SIWOB wird sich nur auf die Bereiche der Normen bezogen, die für die Entwicklung von Software eine Rolle spielen.

15. vgl. (Dussa-Zieger, 2011, S. 6)
16. vgl. (Dussa-Zieger, 2011, S. 7)

### Norm 1: EN ISO 13849 Teil 1

Die EN ISO 13849 stellt Gestaltungsleitsätze für sicherheitsbezogene Bauteile von Steuerungen bereit. Teil 1 dieser sicherheitsspezifischen Norm stellt unter anderem Leitsätze für die sicherheitsbezogene Steuerung von Maschinen und der sicherheitsbezogenen Entwicklung von Software auf. Diese Norm bietet sich als Vorlage für eine Auswertung der Normstruktur an, die in sechs Kapitel aufgeteilt ist und im Folgenden grob beschrieben wird.

#### 1. Allgemeine Angaben

Das erste Kapitel wird als "Allgemeine Angaben" betitelt und soll Informationen enthalten, wie beispielweise verwendete Abkürzungen, Begriffe und Definition oder anzuwendende Normen und Vorschriften.

#### 2. Projektmanagement

Das darauf folgende zweite Kapitel enthält Strukturen für das "Projektmanagement", wie beispielweise Ablaufplan, Projektorganisation und Zuständigkeiten. Ebenso sind Modifikationsverfahren sowie das Dokumentenmanagement Kern dieses Dokumentkapitels.

#### 3. Funktionale Angaben

Kapitel drei wird als "Funktionale Angaben" bezeichnet und stellt eines der Kernkapitel der EN ISO 13849 dar. Dieser Bereich enthält exakte Beschreibungen aller Sicherheitsfunktionen, die in einzelnen Teilfunktionen beschrieben werden müssen. Weiterhin findet in diesem Kapitel die Priorisierung von Sicherheitsfunktionen statt. Ebenso müssen die Nicht-Sicherheitsfunktionen erläutert werden und dabei strikt von Sicherheitsfunktionen differenziert werden.

#### 4. Systembeschreibung und Maßnahmen

Innerhalb dieses Kapitels sind sowohl notwendige Hardware-Beschreibungen, als auch eine vorgesehene Realisierung des Performance Level enthalten. In Beziehung zur Software-Prüfung stellt das Unterkapitel Software, den Kernpunkt für die Software-Prüfung dar. In diesem Teilkapitel erfolgt eine Beschreibung der Funktionen, ausgehend von den Beschreibungen der Sicherheitsfunktionen, die durch Software realisiert werden. In diesem Zusammenhang soll die Software-Struktur und die Funktionsabläufe verständlich erläutert werden.

#### 5. Technische Daten

Das vorletzte Kapitel besitzt hohe Anteile technischer Aspekte. Zum einen soll ein Abschnitt über elektrische Daten verfasst werden, zum anderen müssen auch mechanische und sonstige Daten in diesem Kapitel aufgenommen werden. Des Weiteren enthält dieser Teil des Dokuments das Produktdesign und dessen besondere Vorgaben.

### 6. Sonstiges

Die EN ISO 13849 schließt das Dokument mit dem Kapitel "Sonstiges" ab. Dieser letzte Abschnitt enthält beispielweise Anhänge in Form von Bildern, Diagrammen oder Tabellen.[17]

## Norm 2: IEEE/ANSI 830-1998

Die IEEE/ANSI 830-1998, des American National Standards Institute (ANSI), beschreibt Vorgaben und Empfehlungen, um Anforderungen in einer Spezifikation zu strukturieren und abzubilden. Diese Vorgaben beinhalten eine Kapitelstruktur sowie detaillierte Vorschläge für die Strukturierung der einzelnen Unterkapitel und deren Inhalte. Für die spätere Erstellung einer prototypischen Vorlage werden diese Vorgaben analysiert und festgehalten.

Nach IEEE/ANSI 830-1998 besteht die Kapitelstruktur für eine Spezifikation aus mindestens vier übergreifenden Kapiteln, der "Einleitung", "Allgemeine Beschreibung der Anforderungen", "Detaillierte Beschreibung der Anforderungen" und "Appendix". Nachfolgend werden diese Kapitel detailliert analysiert. Zu diesem Zweck erfolg eine Abbildung der Kapitelstruktur, in Form einer abgestuften Nummerierung, damit später eine leichtere Zuordnung zur prototypischen Dokumentvorlage möglich wird.

### 1. Einleitung

Die Einleitung gliedert Inhalte, die einen Gesamtüberblick über die Software-Spezifikation und das Produkt geben. Innerhalb der Einleitung sind weitere Unterkapitel enthalten, die diese Inhalte innerhalb dieses Kapitels weiter aufgliedern und strukturieren.[18]

#### 1.1 Zweck des Dokuments

Innerhalb dieses Kapitels wird beschrieben, welche Zielsetzung und welchen Zweck das Dokument hat. Im Weiterem ist zu erläutern, an welche Adressaten das Dokument sich richtet.

#### 1.2 Abgrenzung der Software

Neben dem Zweck des Dokuments muss ein Überblick über das Produkt gegeben werden. Dazu wird der Name des Produkts beschrieben. Um das Produkt weiter abzugrenzen sind Kernfunktionalitäten zu schildern. Für ein Grundverständnis darüber, welchen Mehrwert das Produkt erzeugt, werden die Ziele und der Nutzen in dem Einsatzgebiet der Software erläutert.[19]

#### 1.3 Begriffsdefinitionen und 1.4 Abkürzungen

Teil der Einleitung sind Begriffsdefinitionen und Abkürzungen und die dazu gehörenden Definitionen, die in der Regel in einer tabellarischen Auflistung zusammengefasst werden.

---

17. vgl. (Deutsches Institut für Normung, 2008b)
18. vgl. (IEEE Computer Society, 1998, S. 11)
19. vgl. (IEEE Computer Society, 1998, S. 11)

### 1.5 Referenzen

Wenn Referenzen zu anderen Dokumenten vorhanden sind, erfolgt eine Auflistung der Referenzen in der Regel in einer tabellarischen Darstellung. Diese Tabelle enthält die Elemente Titel, (wenn zutreffend) Report Nummer, Datum, die veröffentlichende Organisation und die Quelle.

### 1.6 Überblick und Struktur des Dokumentes

Innerhalb dieses Kapitels wird eine Zusammenfassung der Inhalte, des Aufbaus und der behandelten Themen gegeben.[20]

## 2. Allgemeine Beschreibungen

Die allgemeinen Beschreibungen setzen das zu entwickelnde Produkt in Zusammenhang zu seiner Umwelt und geben einen Überblick der Interaktionen mit der Software und der Operationen der Software. Diese Beschreibungen enthalten Erläuterungen über die Interaktionen und die dadurch entstehenden Einschränkungen bzw. Abhängigkeiten bezüglich des Produkts. Es werden keine ausführlichen Definitionen formuliert, sondern nur eine kurze Einführung in das Produkt gegeben.[21]

### 2.1 Produktperspektive

In der Produktperspektive wird die Software in Bezug auf die Interaktionen mit der näheren Umwelt beschrieben. Dazu sind verschiedene Schnittstellen des Produkts mit der Umwelt sowie aus der Umwelt bedingte Abhängigkeiten zu beschreiben.

#### 2.1.1 Systemschnittstellen

In den Systemschnittstellen werden die vom Produkt verwendeten Verbindungen aufgelistet und beschrieben, in welcher Art die zu erstellende Software diese Schnittstelle verwendet. Ein einzelner Eintrag beinhaltet den Namen und die Beschreibungen der jeweiligen Schnittstelle.[22]

#### 2.1.2 Nutzerschnittstellen

In den Nutzerschnittstellen werden notwendige Konfigurationskriterien und Optimierungsaspekte jeder Schnittstelle erläutert. Solche Konfigurationskriterien sind beispielsweise Bildschirm-, Fenstergrößen, Menüpunkte oder Report-Inhalte. Ein Eintrag einer Nutzerschnittstelle enthält die Beschreibungen Konfigurationskriterien und Optimierungsaspekte.

#### 2.1.3 Hardwareschnittstellen

In den Hardware-Schnittstellen werden die Verbindungen von der Software zu den Hardware-Komponenten beschrieben, die unter anderem die Beschaffenheit dieser enthalten. Hierfür sind die Geräte aufzuführen die von der Schnittstelle unterstützt werden und die Funktionsweise ist

20. vgl. (IEEE·Computer·Society,·1998,·S.·12)
21. vgl. (IEEE·Computer·Society,·1998,·S.·12)
22. vgl. (IEEE·Computer·Society,·1998,·S.·12f.)

kurz zu erläutern. Ein Eintrag der Hardware-Schnittstellen enthält einen Namen und eine Beschreibung.[23]

### 2.1.4 Softwareschnittstellen

In den Software-Schnittstellen wird benötigte Software und die Interaktion mit dieser definiert. Eine solche Software könnte beispielsweise ein Betriebssystem oder ein Datenbanksystem sein und wird mit grundlegenden Daten beschrieben. Diese Daten beinhalten den Namen, eine Abkürzung, die Versionsnummer, sowie die Quelle der Software. Im Weiteren erfolgt eine Beschreibung der genutzten Schnittstellen und deren Einsatzzweck, die unter anderem die Nachrichtenart und das Nachrichtenformat einschließen. Wenn bereits eine Dokumentation der Schnittstelle besteht ist auf dieser zu referenzieren.

### 2.1.5 Kommunikationsschnittstellen

In den Kommunikationsschnittstellen werden Protokolle für den Datenaustausch und daraus resultierende Anforderungen erläutert, wie beispielsweise Netzwerkprotokolle. Ein Eintrag in den Kommunikationsschnittstellen enthält einen Namen und eine Beschreibung.[24]

### 2.1.6 Speichernutzung

In der Speichernutzung werden mögliche Einschränkungen und Limitierung durch den Speicher in einzelnen Einträgen durch Inhaltselemente beschrieben.

### 2.1.7 Softwareeinsatz

Im Softwareeinsatz werden die Einsatzmöglichkeiten für die Software beschrieben, beispielsweise welche Vorgänge die Bedienung des Nutzers erfordern sowie Beschreibungen von Funktionen zum Verarbeiten, Sichern und Wiederherstellen von Daten.

### 2.1.8 Installation und Initialisierung

In der Installation und Initialisierung werden die Anforderungen an die Software am gewünschten Einsatzort beschrieben. Diese geben Maßnahmen vor, die eingehalten werden müssen oder eine Voraussetzung darstellen, um die Software in Betrieb nehmen zu können. Ein Eintrag stellt die Beschreibung einer Anforderung in einfacher inhaltlicher Form dar.[25]

## 2.2 Produktfunktionen

Die Beschreibungen der Produktfunktionen geben eine Zusammenfassung der wichtigsten Funktionen und Merkmale des Produkts. Es soll keine Detailbeschreibung der Funktionen erfolgen, es können aber, einfache Diagramme benutzt werden, wenn es der Erklärung dient, um logische Zusammenhänge einzelner Funktionen zu veranschaulichen.

---

23. vgl. (IEEE·Computer·Society,·1998,·S.·13)
24. vgl. (IEEE·Computer·Society,·1998,·S.·13f.)
25. vgl. (IEEE·Computer·Society,·1998,·S.·14)

### 2.3 Nutzereigenschaften

Die Formulierungen in einer Spezifikation hängen zumeist vom Wissensstand des jeweiligen Autors ab und stützen sich auf die technischen Kenntnisse oder Erfahrungen dieser Person. Damit die Gründe für spezielle Formulierungen besser nachvollzogen werden können, sind die besonderen Nutzereigenschaften zu erläutern. Hierbei handelt es sich um eine textuelle Beschreibung der Nutzertypen.

### 2.4 Weitere Beschränkungen

Grundsätzlich sind weitere Beschränkungen für die Entwicklermöglichkeiten mit einer kurzen Erläuterung aufzuführen, wie beispielsweise die Limitierung durch Hardware, Schnittstellen zu anderen Applikationen oder festgelegte Richtlinien. Weitere Aspekte können Sicherheitsbedenken, parallele Arbeitsprozesse, Prüfungs- und Kontrollfunktionen und Anforderungen an die Zuverlässigkeit der Software sein.

### 2.5 Annahmen und Abhängigkeiten

Annahmen und Abhängigkeiten, die Einfluss auf die Funktion der Software haben, sind in einem weiteren Abschnitt aufzulisten. Eine dieser Annahmen könnte sein, dass ein spezifisches Betriebssystem vorhanden ist. Sollte sich diese Annahme ändern, muss die Spezifikation der Software angepasst werden, um den neuen Abhängigkeiten gerecht zu werden..

### 2.6 Zurückgestellte Anforderungen

In einer Entwicklung können Anforderungen bestehen, die erst in einer späteren Produktversion technischen realisiert werden sollen. Damit die Vollständigkeit dieser Anforderungen jedoch erhalten bleibt, werden diese zurückgestellten Anforderungen passend zusammengefasst.[26]

## 3. Detaillierte Beschreibungen der Anforderungen

Dieses Kapitel wird für eine Ausführung aller zuvor oberflächlich spezifizierten Anforderungen verwendet, die in einer Detailtiefe beschrieben werden, anhand derer ein Entwickler mit den spezifizierten Informationen die Software umsetzten kann. Weiterhin müssen die detaillierten Anforderungen testbar sein, sodass mindestens die Eingangsinformationen und Antworten der Software und daran beteiligte Funktionen definiert werden. Zudem sollte die Lesbarkeit, durch die Struktur und Anordnung der Anforderungen, bestmöglich unterstützt werden.[27]

### 3.1 Externe Schnittstellen

Durch die externen Schnittstellen erfolgt eine detaillierte Beschreibung über die verschiedenen Eingangs- und Ausgangsinformationen.

---

26. vgl. (IEEE Computer Society, 1998, S. 14f.)
27. vgl. (IEEE Computer Society, 1998, S. 16)

### 3.1.1 - 3.1.4 Nutzer-, Hardware-, Software-, Kommunikationsschnittstellen

Alle in den externen Schnittstellen enthalten Unterkapitel (Nutzer-, Hardware-, Software- und Kommunikationsschnittstellen) weisen jeweils die gleiche interne Struktur auf und werden deshalb zusammengefasst betrachtet. Die Definition einer einzelnen Schnittstelle enthält zahlreiche Elemente, wie beispielsweise Name der Schnittstelle, Beschreibung des Zwecks, Beendigungsnachrichten und viele weitere, die eine detaillierte Beschreibung in inhaltlicher Form darstellen.[28]

## 3.2 Hauptmerkmale der Software

Funktionale Anforderungen beschreiben die genauen Vorgänge innerhalb der Software. Hierfür werden Voraussetzungen für die Annahme und Verarbeitung von Eingangsinformation erläutert und welche Ausgabeinformation zu erwarten sind. Mit Hilfe einer schrittweisen Beschreibung der Transformation dieser Informationen, erfolgt eine Erläuterung über den Zusammenhang zwischen Eingabe- und Ausgabeinformationen. Weiterhin wird der sequentielle Ablauf der einzelnen Funktionen, eine Fehlerbehandlung und die Behandlung sonstiger abnormale Situationen beschrieben.[29]

## 3.3 Performanzanforderungen

Mit den Anforderungen an die Performanz wird die Leistungsfähigkeit der Software beschrieben, die im Bezug auf die Interaktionen mit dieser existieren. Beispielsweise wird beschrieben, wie viele Nutzer gleichzeitig auf die Software zugreifen können oder die Menge an parallel zu verarbeitenden Informationen definiert.

## 3.4 Logische Datenbankanforderungen

Mit Hilfe der Anforderungen an die logische Datenbank erfolgt eine Beschreibung über die Struktur und die Daten, die innerhalb eines Datenbanksystems gespeichert werden. Diese Anforderungen enthalten verschiedene Informationen, unter anderem über Datentypen, Entitäten, Beziehungen und weitere, die von gesetzlichen oder betrieblichen Normen definiert werden.[30]

## 3.5 Einschränkungen des Entwurfs

In diesem Kapitel werden die Limitierungen, die beispielsweise aus Normen oder Hardware resultieren, beschrieben. Hierzu werden Normen und Standards berücksichtigt, welche für die Entwicklung einer Software wichtige Rahmenbedingung vorgeben. Diese werden in passenden Anforderungen erläutert, wie beispielsweise Bedingungen an das Report-Format oder die Datenbezeichnung. Zusätzlich können weitere Einschränkungen spezifiziert werden.[31]

---

28. vgl. (IEEE·Computer·Society,·1998,·S.·16)
29. vgl. (IEEE·Computer·Society,·1998,·S.·16)
30. vgl. (IEEE·Computer·Society,·1998,·S.·16f.)
31. vgl. (IEEE·Computer·Society,·1998,·S.·17)

### 3.6 Softwareattribute

Die Eigenschaften einer Software werden innerhalb der Software-Eigenschaften detailliert formuliert. Zusätzlich muss berücksichtigt werden, dass die einzelnen Anforderungen verifizierbar beschrieben werden.

#### 3.6.1 - 3.6.5 Zuverlässigkeit, Verfügbarkeit, Sicherheit, Wartbarkeit, Portabilität

Alle Eigenschaften einer Software, die in den passenden Unterkapitel formuliert werden, weisen jeweils die gleiche interne Struktur auf und werden deshalb zusammengefasst betrachtet. In diesem Kapitel werden die Faktoren beschrieben, mit der die Zuverlässigkeit, Verfügbarkeit, Sicherheit, Wartbarkeit und Portabilität der Software sichergestellt werden können. Hierbei handelt es sich um eine natürlich sprachliche Beschreibung der Eigenschaften.[32]

### 4. Appendix

Das abschließende Kapitel enthält zusätzliche Informationen, die den Inhalt der Dokumentation vervollständigen oder erweitern und aufgrund ihres Umfanges nicht in die zuvor beschriebene Dokumentstruktur eingegliedert werden können. Dies können beispielsweise Referenzen auf Abbildungen oder externe Quellen sein. Alle Eigenschaften der Anhänge werden in natürlicher Sprache beschrieben.

In der IEEE/ANSI 830-1998 sind verschiedene Empfehlungen zur Strukturierung der detaillierten Beschreibungen der Anforderungen gegeben. Diese Empfehlungen leiten sich aus projektspezifischen Darstellungsbedürfnissen ab. Demnach ist für die Erstellung einer Software empfehlenswert, im Vorfeld zu definieren, welche Strukturierung für die Spezifikation am besten geeignet ist.[33] Im Folgenden werden die verschiedenen projektspezifischen Darstellungsoptionen zusammengefasst.

#### Variante A oder B

Eine Software kann auf Basis verschiedener Zustände arbeiten, die eine erweiterte oder beschränkende Funktion erlauben, wobei sich der Zugriff auf die Software ändern kann. Wenn eine Software auf Basis von verschiedenen Modi arbeitet (Variante A und B), ist eine Gliederung nach System- oder Software-Modus vorzunehmen, sodass die funktionalen Anforderungen nach Modus geteilt und weiter spezifiziert werden. Sollten zusätzlich die Schnittstellen betroffen sein, sind diese ebenfalls unter den jeweiligen Modus zu gliedern (nur Variante B).[34]

#### Variante C

Wird die Funktionalität der Software in Abhängigkeit ausführender Nutzertypen beschrieben, ist als Strukturierung die Darstellung nach Nutzerklassen zu wählen.[35]

---

32. vgl. (IEEE·Computer·Society,·1998,·S.·17f.)
33. vgl. (IEEE·Computer·Society,·1998,·S.·18)
34. vgl. (IEEE·Computer·Society,·1998,·S.·19-21)
35. vgl. (IEEE·Computer·Society,·1998,·S.·19-22)

#### Variante D

Ist der Ablauf der Software-Funktionen nach realen Objekten ausgelegt, ist eine Gliederung nach Objekten oder Klassen vorzunehmen. Sollten Objekte gleiche Eigenschaften oder Funktionen nutzen, sind diese Eigenschaften und Funktionen als Klasse zu bündeln. Innerhalb der Struktur werden die Attribute eines Objekts oder einer Klasse beschrieben und jede genutzte Anforderung detailliert spezifiziert.[36]

#### Variante E

Findet eine Gliederung der Software nach den Hauptmerkmalen statt, wird zu jedem Merkmal eine einleitende Beschreibung, eine Stimulus-Antwort-Sequenz und die mit diesem Merkmal assoziierten funktionalen Anforderungen formuliert.[37]

#### Variante F

Anforderungen können nach Stimulus oder Antwort strukturiert werden. Hierzu werden den Stimuli (bzw. Antworten) die verwendeten funktionalen Anforderungen untergeordnet und anhand der Stimuli gruppiert.[38]

#### Variante G

Wenn keine der oben beschriebenen Strukturen auf die Software zutreffen, kann eine Aufteilung nach funktionaler Hierarchie vorgenommen werden. Hierbei wird nach Eingangsinformationen, Ausgangsinformationen oder internen Datenzugriffen gegliedert. In diesen Strukturen erfolgt eine inhaltliche Beschreibung weiterer Elemente.[39]

### 5.1.1.3 Usability in der Software-Entwicklung

Grafische Dialogsysteme (User Interface) stellen den verbreitetsten Standard dar, durch die ein Benutzer mit einer Software interagiert. Dieses Kapitel beschreibt die Regeln und Normen, die bei der Entwicklung für ein User Interface beachtet werden müssen. In diesem Zusammenhang wird Usability nach ISO 9241-11 auch als Gebrauchstauglichkeit bezeichnet und beschreibt die Benutzerfreundlichkeit von entwickelten Anwendungen unter der die Begriffe Effektivität, Effizienz und Zufriedenheit vereint werden, anhand derer gemessen werden kann, wie gut ein Benutzer ein bestimmtes Ziel erreicht.[40]

Die Anzahl der Usability-Modelle ist unübersichtlich und häufig wird der Begriff Usability von verschiedensten Autoren unterschiedlich interpretiert. Hervorzuheben sind in diesem Kontext die ISO-Norm 9241-110[41], sowie die Modelle von Shneiderman[42] und Nielsen[43]. Im Grundprinzip ähneln sich diese Mo-

36. vgl. (IEEE·Computer·Society,·1998,·S.·19-25)
37. vgl. (IEEE·Computer·Society,·1998,·S.·23)
38. vgl. (IEEE·Computer·Society,·1998,·S.·24)
39. vgl. (IEEE·Computer·Society,·1998,·S.·24ff.)
40. vgl. (Woywode,·Mädche,·Wallach,·&·Plach,·2012,·S.·23)
41. vgl. (Deutsches·Institut·für·Normung,·2008a)
42. vgl. (Shneiderman·&·Plaisant,·2004,·S.·31ff.)

delle, wurden jedoch von jedem Autor in einigen Aspekten verfeinert. So definiert Shneiderman Effektivität, Erlernbarkeit, Flexibilität und das Verhalten einer Software als Ziele seines Usability-Modells, wohingegen nach ISO-Norm 9241-110 lediglich Effizienz und Zufriedenheit genügen. Die genannten Modelle werden für eine bessere Übersicht in den nachfolgenden Kapiteln kurz zusammengefasst.

Auf Basis dieser Modelle entwickelte Seffa ein konsolidiertes Modell QUIM (Quality in Use Integrated Measurement), um die Defizite der etablierten Ansätze zu beheben, da seiner Meinung nach keine dieser Lösungen ausreiche.[44] Eine Begründung lieferte Seffa mit einigen einschlägigen Argumenten. Zum Einen berücksichtigen alle etablierten Modelle nicht den Individualitätsfaktor, der beispielsweise bei der Erlernbarkeit eine wichtige Rolle einnimmt und somit zu nicht vergleichbaren Ergebnissen führt. Zum Anderen sind diese Modelle statisch und erlauben keine zeitliche Betrachtung, weshalb unter anderem eine Klassifikation von möglichen Gefahren ausgeschlossen ist. Weiterhin ist es schwierig, Usability-Standards in die Praxis zu überführen, um Usability zu messen, da nicht immer deutlich ist, welche Faktoren oder Kriterien in Verbindung stehen oder in welcher Kombination diese betrachtet werden müssten. Als letztes Argument für einen konsolidierten Ansatz nennt Seffa die Notwendigkeit, einen Leitfaden zur Interpretation aller gesammelten Informationen zu liefern, wodurch auch Nicht-Experten in der Lage wären diese adäquat interpretieren zu können. [45]

#### 5.1.1.3.1 Grundsätze der Dialoggestaltung

In der heutigen Zeit existieren viele unterschiedliche User Interfaces, die mehr oder weniger gut von einem Benutzer zu bedienen sind, da sich diese im Aufbau und Design erheblich voneinander unterscheiden können. Zu diesem Zweck wurden Grundsätze der Dialoggestaltung in der ISO-Norm 9141-10 aufgestellt, die einen Ansatz für ein gutes Interface Design liefern. Diese Grundsätze beschreiben die Anforderungen für ein angemessenes Schnittstellen-Design die erfüllt sein müssen, um den gestellten Ansprüchen durch den Benutzer gerecht zu werden. Anhand dieser Normgrundsätze kann ein User Interface gestaltet werden, das eine gute Bedienung des Systems gewährleistet. In diesem Zusammenhang werden im Folgenden die sieben Grundsätze der ISO-Norm beschrieben.[46]

1. **Aufgabenangemessenheit**

   Ein Dialog muss für eine Aufgabe angemessen entwickelt werden. Es dürfen keine unnötigen Funktionen vorhanden sein, die eine Bedienung erschweren und keine wichtigen Funktionen fehlen, um die gestellten Aufgaben erfüllen zu können.

43. vgl. (Nielsen, 1993, S. 26ff.)
44. vgl. (Seffah, Donyaee, Kline, & Padda, 2006, S. 161)
45. vgl. (Seffah et al., 2006, S. 165ff.)
46. vgl. (Hering, 2005, S. 283ff.) & vgl. (Heinecke, 2004, S. 168ff.)

#### 2. Selbstbeschreibungsfähigkeit

Ein User Interface ist intuitiv zu bedienen, sofern Rückmeldungen von ausreichend vielen und angemessenen Informationen an den Benutzer erfolgt.

#### 3. Steuerbarkeit

Die Geschwindigkeit und die Richtung des Dialoges darf nur vom Benutzer festgelegt werden und nicht vom System. Weiterhin muss es möglich sein, den Dialog zu unterbrechen und anschließend an einer geeigneten Stelle wiederaufzunehmen.

#### 4. Erwartungskonformität

Der Benutzer muss sein implizites Wissen auf das System adaptieren können, wodurch dieses einfacher bedient werden kann. Die Übertragung von ähnlichen Designelementen mit gleichen Funktionen erfüllt dieses Ziel.

#### 5. Fehlertoleranz

Ein User Interface erfüllt das Kriterium der Fehlertoleranz, wenn eine Aufgabe erfüllt werden kann, ungeachtet fehlerhafter Bedienung. Entweder werden Fehler automatisch korrigiert oder das System weist den Benutzer auf die möglichen Fehler hin.

#### 6. Individualisierbarkeit

Ein Benutzer muss ein System an seine Fähigkeiten und Präferenzen anpassen können, um eine Aufgabe effizient bearbeiten zu können.

#### 7. Lernförderlichkeit

Ein Dialog muss den Fähigkeiten des Benutzers angepasst werden können. Anfänger benötigen eine andere Lernunterstützung als Experten, damit der Umgang mit dem User Interface erlernt werden kann. Zu Beginn wird ein Anfänger stärker in seinem Lernprozess gefördert und kann diese Unterstützung nach Bedarf deaktivieren, wodurch die Effizienz mit steigenden Wissensstand verbessert wird.

### 5.1.1.3.2 Zehn Heuristiken von Jakob Nielsen

Bei der Usability geht es vor allem um eine gute Gestaltung von User Interfaces, wodurch die Effektivität, Effizienz und Zufriedenheit bei der Interaktion verbessert werden soll. Jakob Nielsen formulierte in dem Zusammenhang mit der Benutzerführung zehn Grundregeln, auf Basis seines Erfahrungswissens, durch die eine möglichst effiziente und effektive Interaktion mit einem User Interface ermöglicht werden soll.[47]

#### 1. Einfache und natürliche Dialoge

Dem Benutzer sollten nur die notwendigsten Informationen angezeigt werden, damit er die Benutzung so schnell wie möglich erlernen kann. Außerdem sollten die Dialoge gemäß den Regeln zur menschli-

47. vgl. (Nielsen, 1993, S. 115ff.)

chen Wahrnehmung ausgelegt sein. Wichtige Informationen sollten visuell hervorgehoben werden, um den Benutzer in seiner Tätigkeit zu unterstützen.

### 2. Sprich die Sprache des Benutzers

Informationen sollten immer in einer natürlichen Sprache vermittelt werden. Die verwendete Fachsprache muss sich hierbei immer am Leistungsstand des Benutzers orientieren und sollte wahlweise umgestellt werden können.

### 3. Minimiere die Belastung für den Benutzer

Ein Benutzer darf nicht mit zu vielen Informationen überfordert werden. Der Umfang von Informationen sollte immer dem Leistungsvermögen des Kurzzeitgedächtnisses entsprechen.

### 4. Konsistenz

Der selbe ausgeführte Befehl muss immer das gleiche Ergebnis erzielen. Des Weiteren muss sich hinter gleichen Objekten die gleiche Funktion verbergen.

### 5. Rückmeldungen

Rückmeldungen sollten einen Benutzer über den Zustand einer Operation informieren, sodass dieser immer weiss, was gerade geschieht. Dies ist vor allem wichtig, wenn die Ausführung einer Aufgabe Zeit benötigt. Ein Benutzer verliert die Aufmerksamkeit und wird von der eigentlichen Tätigkeit abgelenkt, sofern eine ausgeführte Aufgabe mehr als 10 Sekunden Zeit in Anspruch nimmt, bei der keine Rückmeldung erfolgt.

### 6. Klar gekennzeichnete Auswege

Über gekennzeichnete Dialoge muss es dem Benutzer ermöglicht werden, zu einem vorherigen Zustand zurückzukehren, sofern er sich in einem ungewollten Zustand befindet. Fehler sollten rückgängig gemacht oder es sollte zu einem festgelegten Startpunkt zurückgekehrt werden können.

### 7. Abkürzungen

Eine Unterstützung von bekannten Abkürzungen, den Softkeys auf der Tastatur, sollte angeboten werden.

### 8. Gute Fehlermeldungen

Sofern ein Benutzer einen Fehler hervorruft, muss der Dialog in der Fehlermeldung eine gute Beschreibung zur Lösung des Problems liefern. Eine schlichte Meldung des Fehlers löst meistens das Problem nicht und der Benutzer bricht die Kommunikation ganz ab.

### 9. Fehlervermeidung

Wenn Fehler vorausgesehen werden können, ist es sinnvoll, diese Fehler durch das System automatisch korrigieren zu lassen, wenn sich diese Möglichkeit ergibt.

#### 10. Hilfe und Dokumentation

Eine Hilfe sollte den Benutzer in der aktuellen Ausführung einer Aufgabe unterstützen, sofern er nicht mehr weiter weiß. Deshalb muss eine Hilfe die aktuelle Situation gut und verständlich für den Benutzer dokumentieren.

### 5.1.1.3.3 Acht goldene Regeln von Ben Shneiderman

Die acht goldenen Regeln für eine gute Schnittstelle zum Benutzer erstellte Shneiderman auf Basis seiner Erfahrungen in der Software-Entwicklung. Die Regeln können der jeweiligen Situation entsprechend angepasst werden. Sie gleichen in einigen Punkten den Heuristiken von Jakob Nielsen, wodurch erkennbar wird, dass ein grundlegendes Maß an Usability existieren muss, um eine Interaktion mit einer Software durch ein User Interface zu gewährleisten.[48]

#### 1. Konsistenz

Es müssen einheitliche Konventionen zu Design, Stil und Text für die Entwickler festgelegt werden. Dadurch wird ein einheitliches Ergebnis erzielt, mit dem ein Benutzer arbeitet.

#### 2. Berücksichtige unterschiedliche Erfahrungen

Benutzer haben unterschiedliche Fähigkeiten und einen anderen Wissensstand, der berücksichtigt werden muss. Ein Anfänger benötigt mehr Unterstützung bei der Benutzung als ein Experte. Ein Experte hingegen kann in seiner Effizienz durch Softkeys unterstützt werden. Des Weiteren muss ein Anfänger in seiner Entwicklung zu einem Experten unterstützt werden.

#### 3. Rückmeldungen auf Aktionen des Benutzers

Die Rückmeldungen auf Aktionen des Benutzers entsprechen den gleichen Anforderungen aus dem Punkt 5 der zehn Heuristiken von Jakob Nielsen.

#### 4. Abgeschlossene Operationen

Operationen, die aus mehreren Schritten bestehen, sollten auch als solche dargestellt werden. Ein Ablauf muss für einen Benutzer visuell dargestellt werden, sodass er über die aktuelle Position im Ablauf Bescheid weiß.

#### 5. Fehler verhindern

Fehler sollten nicht nur visuell und konstruktiv für den Benutzer dargestellt werden, sondern durch geeignete Routinen direkt vermieden werden, soweit es möglich ist. Um Fehler zu vermeiden, können dem Benutzer Alternativen zur Auswahl angeboten werden, aus denen er die gewünschte Option auswählen kann.

---

48. vgl. (Shneiderman & Plaisant, 2004, S. 74ff.) & vgl. (Dahm, 2006, S. 151ff.)

**6. Einfache Rücksetzungsmöglichkeiten**

Damit ein Benutzer in seiner Arbeit und in seinem Vertrauen im Umgang mit der technischen Schnittstelle bestärkt wird, muss ihm eine Möglichkeit angeboten werden, um beispielsweise Fehler rückgängig zu machen. Ein Benutzer wird dadurch in seiner Lerntätigkeit gefördert, da er Funktionen einfach ausprobieren kann, ohne Konsequenzen befürchten zu müssen.

**7. Benutzerbestimmte Eingaben**

Dem Benutzer muss immer vermittelt werden, dass er die Kontrolle über eine Aktion hat. Er muss stets die Möglichkeit haben, eine Aktion anzuhalten oder abzubrechen.

**8. Geringe Belastung des Kurzzeitgedächtnisses**

Das Kurzzeitgedächtnis des Benutzers sollte nicht durch zu viele Aktionen und Operationen überfordert werden. Der Umfang von Informationen sollte immer dem Leistungsvermögen des Kurzzeitgedächtnisses entsprechen.

#### 5.1.1.4 Das Prinzip einer serviceorientierten Architektur

Der Aufbau dieses Kapitels beinhaltet eine kurze Zusammenfassung der Grundlagen einer serviceorientierter Architektur. Dabei soll das Prinzip und die Funktionsweise dieser Struktur aufgezeigt werden.

Die serviceorientierte Architektur (SOA) wird auch als Lehrvorgang in der Software-Entwicklung der letzten Jahrzehnte gesehen, denn Eigenschaften verteilter Systeme, sowie aktuelle Entwurfprinzipien, wie beispielweise Kapselung, Abstraktion und Schnittstellenimplementierung schreiben SOA eine hohe Bedeutung zu.[49] Bis zum heutigen Tage gibt es keine allgemeine Definition einer serviceorientierten Architektur und dennoch ähneln sich Aufbau sowie die Notwendigkeit der von Analysten oder großen Unternehmen definierten SOA.[50]

Für beide Parteien ist der Service als standardisierte Darstellung von Funktionalität das wichtigste Element von SOA. Die Services oder auch Dienste dienen als selbstbeschreibende, offene Komponenten, welche eine zügige und kostensparende Zusammenstellung von verteilten Anwendung ermöglichen.[51]

Die Sichtweise der Hersteller und Analysten beschreibt mit SOA ein Architekturmodell, welches lose gekoppelte Services als Kernelemente enthält. Zudem besitzt diese Architektur eine oder mehrere standardisierte Schnittstellen sowie die Service Implementierung als applikatorische Funktion.[52]

Eine SOA soll die Unabhängigkeit einzelner Services auszeichnen, indem jeder Service als Abstraktion eine vordefinierte Schnittstelle für die Kommunikation nach Außen anbietet. Das Prinzip der Kapselung wird durch die Abgrenzung der Funktionalität nach innen erreicht. Weiterhin wird das Prinzip einer SOA

49. vgl. (Burbiel, 2007, S. 481)
50. vgl. (Liebhart, 2007, S. 22)
51. vgl. (Liebhart, 2007, S. 22)
52. vgl. (Liebhart, 2007, S. 22)

oft mit dem Prinzip der Software-Komponente verglichen. Dies wird damit begründet, dass eine serviceorientierte Architektur durch die Verbindung der Dienste untereinander eine Geschäftslogik realisiert.[53]

Obwohl in modernen SOA-Systemen die Verbindung von Client und Server nicht typisch als Basis einer servcieorientierten Architektur angesehen wird, kommt das Client/Server Modell oftmals zum Einsatz. Ebenso können sich auch Server-seitige Dienste in einer SOA untereinander nutzen.[54] Als Service Provider wird der Anbieter eines solchen Dienstes bezeichnet und als Service Consumer der Konsument eines Dienstes. Weiterhin ist keine zentrale Anbindung der Dienste notwendig. Dadurch besteht die Möglichkeit, Dienste in verteilten Systemen zu realisieren. Aufgrund Dienst-unabhängiger Host-Plattformen, können somit heterogene Systeme vollständig unterstützt werden.[55]

Weitere wichtige Aspekte, die SOA von Standardarchitekturen abheben, sind Web Services und die Eigenschaft der Geschäftsprozessmodellierung. Web Services fügen einem funktionalen System die Grundmechanismen des Webs zu, sodass Funktionalität mit der universellen Verfügbarkeit von heutigen textuellen Informationen vergleichbar wird. Die Funktionsweise eines Web Services ähnelt stark einer Web-Seite, denn der Aufruf eines Web Services erfüllt das Prinzip der Client/Server-Aufrufsequenz zwischen Webserver und dem Browser eines Clienten. Dabei sendet der Client eine Anfrage in Form eines sogenannten HTTP-POST Request an die Adresse des Web Services. Der Inhalt dieses Requests enthält eine Simple Object Access Protocol (SOAP) Nachricht, Uniform Resource Locator (URL) Adresse, Aufruf des Services und Aufruf-Parameter. Die vorhandene Nachricht wird von dem Web Service, in ein mit der Server-Anwendung kompatibles Format demaskiert und vom eigentlichen Dienst verarbeitet. Dieser Dienst führt die Verarbeitung der Anfrage durch und schickt diese an die Service-Schnittstelle weiter. Im anschließenden Schritt maskiert der Web Service die Daten in eine SOAP Nachricht und schickt diese per HTTP-Response an den Clienten zurück, welcher die Nachricht öffnet und weiterverarbeitet.[56]

SOAP sowie die Web Service Description Language (WSDL) sind dringend erforderliche Standards für eine funktionierende SOA. WSDL dient der Schnittstellenbeschreibung eines Web Services. SOAP ist in einer SOA die Sprache der Consumer und Provider.[57]

Die letzte wichtige Grundlage für eine funktionierende SOA wird als Workflow Management-System oder ebenfalls als Business Process Management (BPM) System bezeichnet. Dieses System orchestriert die Sammlung von Diensten, durch einen Workflow, als betriebliches Informationssystem. Ein Workflow wird auch als betrieblicher Ablauf deklariert und ist seit Jahren als wichtiges Werkzeug in die Unternehmensorganisation integriert. Zudem besteht die Möglichkeit mit der Unterstützung einer Business Process Exe-

53. vgl. (Burbiel, 2007, S. 481)
54. vgl. (Burbiel, 2007, S. 481)
55. vgl. (Burbiel, 2007, S. 482)
56. vgl. (Liebhart, 2007, S. 22f.)
57. vgl. (Liebhart, 2007, S. 23)

cution Language (BPEL), die modellierten Abläufe eines betrieblichen Informationssystem als grafische Darstellung zu erstellen und parallel aus der Abbildung Code zu generieren.[58]

#### 5.1.1.5 Zentrale Standards innerhalb einer SOA

Im Rahmen des Forschungsprojektes werden eine Reihe von Standards zur Unterstützung der SOA verwendet. Im Folgenden sollen die wesentlichen Standards für den Umgang mit serviceorientierten Architekturen unter Verwendung eines prototypischen BUS-Systems aufgeführt werden.

##### XML

Die Extensible Markup Language (XML) ist ein textbasiertes Format für die Darstellung von strukturierten Informationen. Als Beispiel können Dokumente, Daten, Konfigurationen, Bücher, Transaktionen, Rechnungen und vieles mehr in XML dargestellt werden. Aktuell ist XML eines der meist verwendeten Formate für den lokalen oder netzwerkbasierten Austausch von strukturierten Informationen zwischen verschiedenen Anwendungen, Menschen oder Computern. Die vorgegebenen Syntaxregeln von XML sind strikt einzuhalten. XML-Werkzeuge verarbeiten keine fehlerhaften Dateien, sondern weisen darauf hin, dass der Fehler zu beheben ist, sodass XML-Dokumente zuverlässig durch Software verarbeitet werden können. XML besitzt eine Reihe von Vorteilen gegenüber anderen Formaten. Zwar besteht die Möglichkeit ein anderes Dateiformat als XML zu verwenden, doch führt dies zu unnötigen Kosten und Anpassungen des Formats. Außerdem gibt es keinerlei Bereitstellung von Editor- oder Suchwerkzeugen, wie XML dies aktuell ermöglicht. Zudem bietet XML den Vorteil der redundanten Syntax, wodurch Fehler innerhalb des XML-Dokuments vermieden werden können. Die Lesbarkeit von XML und die Verwendung von Element- und Attributnamen in XML bedeutet, dass Formate schnell erkannt und verstanden werden können. Weiterhin trägt die Vernetzung von XML zu den wesentlichen Vorteilen gegenüber anderen Formaten bei. Dies bedeutet, dass jedes XML-Dokument von einem beliebigen XML-Werkzeug ausgeführt und modifiziert werden kann. Dadurch wird der Wert der einzelnen XML-Werkzeuge und Dokumente gegenseitig erhöht, wodurch dieser Datentyp aktuell einer der weit verbreitetsten Datentypen ist, vor allem in Verbindung mit Web Services.[59]

##### SOAP

SOAP bietet ein einfaches Protokoll für den webbasierten Austausch von XML-Daten. Das Protokoll verpackt die XML-Daten als Nachricht in einen Umschlag (Envelope) bestehend aus zwei Teilen. Der SOAP-Header beinhaltet Informationen bezüglich der Sicherheit, der Adressierung sowie Zusatzinformationen für die Verarbeitung des sogenannten SOAP-Body. Einzelne Elementblöcke strukturieren die Informationen. Weiterhin werden die Informationen über eine eindeutige Uniform Resource Identification (URI) ermittelt und können somit dem jeweiligen Knoten zugeordnet werden. Generell verfügt dieser Mechanis-

58. vgl. (Liebhart, 2007, S. 23)
59. vgl. (Liam R. E. Quin, 2014)

mus über hohe Flexibilität, sodass zusätzliche Spezifikationen von weiteren Blöcken definiert werden können.[60]

Der SOAP-Body beinhaltet die eigentlich zu kommunizierenden XML-Daten. Grundsätzlich basiert SOAP auf der Gegebenheit, dass bei Bedarf auch eine Kommunikation über Wide Area Networks (WAN) realisiert werden kann. Unter Einsatz von XML, für diese Art der Kommunikation, kann eine grundlegende Interoperabilität zwischen den verschiedenen Plattformen und Technologien bereitgestellt werden.[61]

Der typische Nachrichtenaustausch über SOAP beinhaltet mehrfachen Nachrichtenaustausch zwischen zwei Knoten, der über ein klassisches Modell erfolgt, das als Anfrage- und Antwortmuster bezeichnet wird. Vorgänger der aktuellen Version verweisen explizit auf die Verwendung von Remote Procedure Calls (RPC) in Zusammenhang mit dem Anfrage-/Antwortmuster. In der aktuellen Version des SOAP Standards unterstützten nicht alle Anfrage-/Antwortmuster die Verwendung von RPC. Vor allem für die Bereitstellung eines Modells, welches ein bestimmtes Programmverhalten benötigt, wird RPC genutzt. Somit können auszutauschende Nachrichten auf eine vordefinierte Beschreibung des Remote-Aufrufs und dessen Antwort reagieren.[62]

### WSDL

Das World Wide Web Consortium (W3C) dokumentiert unter anderem den Standard zur Beschreibung von Web Services. Die Web Services Description Language 2.0 spezifiziert ein Modell und ein XML-Format zur Beschreibung von Web Services. WSDL Version 2.0 wird innerhalb der Abschlussarbeit durch die Schreibweise ohne Versionsnummer ersetzt. WSDL stellt die Beschreibung einer abstrakten Funktionalität, getrennt von den tatsächlichen Details einer Service Beschreibung dar, respektive wo und wie die Funktionen angeboten werden. In beiden Bereichen der Beschreibungen bestehen eine Reihe an Konstrukten, welche die Wiederverwendbarkeit einer Beschreibung verbessern soll. Auf abstrakter Ebene beschreibt WSDL das Empfangen und Senden von Web Service Nachrichten. Die verwendeten Nachrichten basieren auf dem XML-Schema und sind unabhängig von einem bestimmten Format. Eine Operation wird in WSDL als Austauschmuster von Nachrichten bezeichnet. Das Nachrichtenaustauschmuster identifiziert die Reihenfolge und die Mächtigkeit von Nachrichten. Weiterhin erkennt es wer Nachrichten logisch sendet oder empfängt. Das sogenannte Binding legt einen Teil der konkreten Beschreibung eines Web Services dar. Transport- oder Anbindungsprotokolle werden in diesem Abschnitt eines WSDL-Dokumentes beschrieben. Zusätzlich bietet WSDL einen Endpunkt, welcher die Netzwerk-Adresse eines Services und Bindings verbindet. Letztlich beinhaltet ein WSDL-Dokument eine gemeinsame Schnittstelle in Form einer Gruppierung der Endpunkte.[63]

60. vgl. (Liebhart, 2007, S. 15)
61. vgl. (Liebhart, 2007, S. 16)
62. vgl. (Nilo Mitra, 2007)
63. vgl. (Roberto Chinnic, Jean-Jacques Moreau, 2007)

**BPEL**

BPEL dient zur Beschreibung und Abbildung von möglichen Geschäftsprozessen. Im Vergleich zu anderen Modellierungstechniken für Geschäftsprozesse kann aus den BPEL modellierten Prozessen, die Steuerung des sogenannten Workflows erstellt werden. Mit BPEL können also mehrere Dienste zu einer Gesamtanwendung orchestriert werden. Des Weiteren unterscheidet BPEL zwischen zwei Arten der Geschäftsprozesse. Zum einen werden Geschäftsprotokolle genannt, welche als abstrakte Beschreibung für den ausführbaren Geschäftsprozess bereitstehen und zum anderen die eigentlichen ausführbaren Geschäftsprozesse. Ein BPEL-Prozess enthält ein Prozess-Interface und ein Prozess-Schema. Das Interface ist in WSDL beschrieben, da ein BPEL-Prozess selbst einen Service darstellt. In einem Prozess-Schema wird der genaue Prozessablauf unter Verwendung von sogenannten Actions bereitgestellt. Auf welche Art und Weise ein BPEL-Prozess instanziiert wird, bestimmen die Correlations Sets. Falls ein Prozess einen Partner beispielweise in Form eines Services aufrufen möchte, wird dies über den Partner Link definiert. Die Beschreibung eines BPEL-Prozesses kann grafisch unter der Verwendung eines BPEL-Editors erfolgen. Die eigentliche Strukturierung eines Prozesses erfolgt aus einer Kombination von hierarchischen Graphen und Blöcken, welche untereinander verschachtelt werden können. Ein Beispiel für ein typische strukturierte Aktivität ist der Einsatz eines "switch" Konstruktes mit dessen Verwendung eine bedingte Ausführung definiert werden kann. Eine strukturierte Aktivität wird für die Steuerung eines Flusses atomarer Aktivitäten genutzt und bildet einen Knoten innerhalb eines Ausführungsbaums. Weitere strukturierte Aktivitäten sind beispielweise für das Durchlaufen eines Graphen oder der parallelen Verarbeitung bestimmt.[64]

### 5.1.1.6 Enterprise Service Bus

In diesem Kapitel werden die Grundlagen, sowohl für die Charakteristiken eines Enterprise Service Bus (ESB), als auch das Kernkonzept eines ESBs verdeutlicht und dabei die Funktionsweise der wichtigsten Komponenten erläutert. Im weiteren Verlauf des Abschlussberichts wird für den Begriff des Busses sowie BUS-Systems, ebenfalls die Bezeichnung Enterprise Service Bus gewählt.

Gerade für IT-Unternehmen besteht immer eine hohe Dringlichkeit, messbare Verbesserung ihrer Datenflüsse zu erzielen. Dabei ist nicht von Belang, ob Finanzdienstleistungsunternehmen ihren Kunden einen größeren Ertrag in Devisengeschäften garantieren möchten, ein Baustofflieferant versucht den Auftragsfluss, der eine komplexe Vertriebskette durchläuft, zu beschleunigen, oder eine Einzelhandelskette den Datenfluss zurück zu ihrem Marken-Manager innerhalb der Hauptunternehmenszentrale verbessern möchte. Denn alle verbindet die Gemeinsamkeit einer erheblichen technischen Herausforderung. Die losen Daten und Informationen verteilen sich über Anwendungen innerhalb unterschiedlicher Unternehmensabteilungen und können nur durch sehr kosten- und zeitintensive Verarbeitung aufgerufen werden. Diese Aspekte zeigen auf, dass Unternehmen weitestgehend Probleme mit der Integration ihrer Dienste bewältigen müssen. Dennoch haben die letzten Jahre mit neuen Technologien, wie beispielsweise SOA, Business to Business (B2B) und Web Services dazu beigetragen, dass die Anzahl integrierter Geschäfts-

64. vgl. (Liebhart, 2007, S. 92f.)

prozesse zunimmt und IT-Unternehmen dem Thema Integration große Aufmerksamkeit schenken. Diese und weitere Technologien soll der ESB, als BUS-System in der Software-Technik, mit seinen Eigenschaften vereinen.[65]

#### 5.1.1.6.1 Charakteristiken eines Enterprise Service Busses

Aufgrund der zuvor dargestellten Problematik der IT-Unternehmen und der immer dringender notwendigen Integration ihrer Dienste, haben sich unterschiedliche Meinungen und Definitionen gebildet, welche Charakteristiken einen ESB auszeichnen. David A. Chapell, der erfahrene technische Chefdirektor des Pionier Enterprise Service Bus namens Sonic ESB, welcher zum Thema Enterprise Service Bus im Jahr 2004 das gleichnamige Buch veröffentlichte, hat die folgenden die Kerncharakteristiken eines ESBs beschrieben.[66]

##### Individuelle Integrationsumgebung für den Einsatz eines Enterprise Service Busses

Das Anwendungsszenario eines Enterprise Service Busses kann sehr flexibel ausfallen. Zum einen besteht die Möglichkeit, den ESB zur vernetzten Unterstützung eines großen Unternehmens zu verwenden, welches im Hintergrund hohe Reichweiten zwischen verschiedenen Abteilungen, Geschäftsbereichen oder Geschäftspartnern adressieren muss. Eine weitere Anwendungsmöglichkeit bezieht sich auf den Einsatz eines ESBs in standortbasierten Projekten. Die Bereitstellung eines flexiblen Grundgerüsts bietet Einsatzmöglichkeiten für jede Art von Integrationsumgebung. Somit können Anwendungen nach Bedarf an den Bus angeschlossen werden und dadurch Sichtbarkeit und Teilbarkeit gemeinsam genutzter Daten anderer Anwendungen oder Dienste in Anspruch nehmen. Weiterhin muss eine Anwendung in Form von vorhandenen Diensten oder Services für die Verwendung eines ESBs nicht weiter als Web Service angepasst werden, da Web Service Schnittstellen ein wichtiger Bestandteil in der Architektur eines ESBs sind, wodurch vorhandene Dienste oder Services mit gegebener Protokollunterstützung, Application Programming Interface (API) Technologien oder Dritthersteller-Adapter angepasst werden.[67]

##### Standard-basierte Integration

Eine weitere fundamentale Charakteristik des ESBs ist die standardbasierte Integration. So stehen einem ESB für die Unterstützung der Konnektivität Java Enterprise Edition (Java EE) Komponenten zur Verfügung, wie beispielsweise Java Messaging Service (JMS) für die Kommunikation der Message-oriented-Middleware (MOM) oder die Java EE Connector Architecture (JCA) für die Konnektivität von Anwendungsadaptern. Des Weiteren verfügt ein ESB über die Fähigkeit Anwendungen zu adressieren, welche mit den Programmiersprachen C, C# oder C++ oder unter .NET entwickelt werden. Weiterführend können alle Anwendungen mit SOAP oder einem Web Service API auf einfache Weise in den ESB integriert werden. Auch für die Bereitstellung von Datentransformationen, intelligentem Routing oder Datenabfrage bezieht sich der ESB auf die etablierten Standards wie XML. Für die Anforderungen einer serviceorien-

65. vgl. (Chappell, 2004, S. 1)
66. vgl. (Chappell, 2004)
67. vgl. (Chappell, 2004, S. 7f.)

tierten Architektur kann ein Enterprise Service Bus auf den Standard WSDL für die Beschreibung abstrakter Service Schnittstellen zurückgreifen. Für das Routing von Geschäftsprozessen wird, wie in einer SOA oft verwendet, der Einsatz von BPEL hinzugezogen. In einer sinnvollen Anordnung vereint der Enterprise Service Bus die standardbasierten Komponenten und Schnittstellen, so dass offene Architekturen realisiert werden können. Die Infrastruktur eines ESBs unterstützt sowohl in industriestandardisierte Komponenten, als auch standardisierte Schnittstellen für den Gebrauch eigener Elemente. [68]

Ein einfacher beispielhafter Überblick einer ESB Integration wird in Abbildung 7 dargestellt. Der ESB integriert eine Java EE Anwendung durch die Verwendung von JCA und JMS. Weiterhin wird eine .Net Anwendung über einen C# Clienten integriert sowie ein Anwendungspaket eines Drittherstellers über einen JCA kompatiblen Adapter. Der untere Bereich der Abbildung zeigt die Anbindung externer Anwendungen über Web Services die den Standard SOAP verwenden.[69]

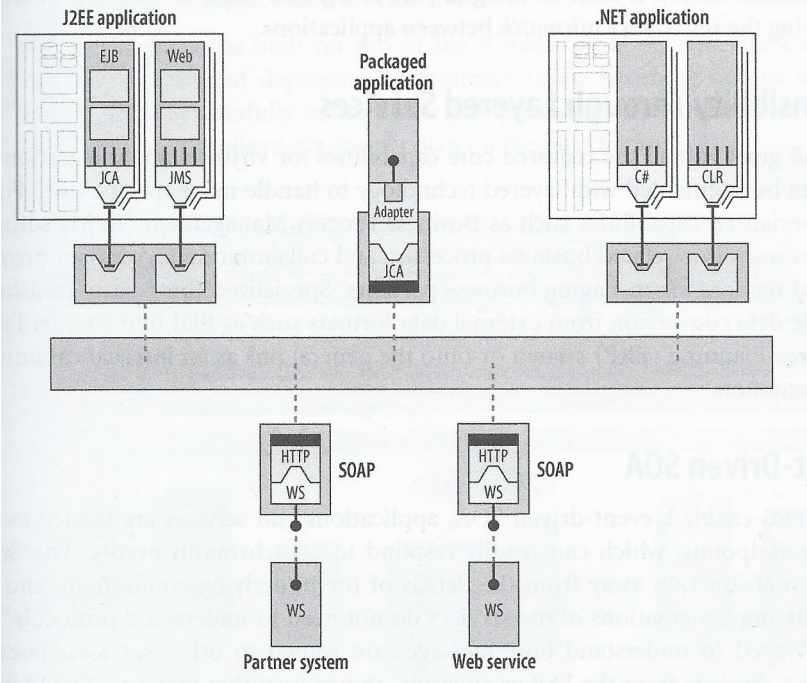



Abb. 7: ESB Integration unterschiedlicher Technologien[70]

### Verteilte Integration

Datentransformation, Adapter für Anwendungen, Datenrouting, sowie Geschäftsprozess-Orchestrierung gehören zu den traditionellen Funktionen, welche ein ESB bereitstellt. Diese Funktionen entsprechen denen eines klassischen Enterprise Architecture Integration Broker (EAI Broker), welche gewöhnlich stark zentralisiert und monolithisch aufgebaut werden. Der ESB soll die typischen Eigenschaften eines EAI Bro-

68. vgl. (Chappell, 2004, S. 8f.)
69. vgl. (Chappell, 2004, S. 9)
70. vgl. (Chappell, 2004, S. 9)

kers als individuelle Services in einer stark verteilten Umgebung anbieten sowie eine unabhängige Erweiterung neuer Services gestatten.[71]

### Verteilte Datentransformation

Zahlreiche Anwendungen nutzen die gleichen Daten, jedoch in einem unterschiedlichen Format. Der ESB kann speziell für die Anwendung angepasste Transformationsservices einbinden und diese überall innerhalb einer verteilten ESB Umgebung aufsetzen, sodass diese Services von jeder anderen Anwendung innerhalb des Busses aufgerufen werden können. Aufgrund des hohen internen Stellenwerts der Datentransformation innerhalb eines ESB, wird ein ESB auch oft als Lösung für die Impedanzfehlanpassung zwischen Anwendungen betrachtet.[72]

### Ereignisgesteuerte SOA

Eine ESB-fähige ereignisgesteuerte SOA stellt Services und Anwendungen als abstrakte Endpunkte dar, welche auf asynchrone Ereignisse reagieren können. Aufgrund der Tatsache, dass SOA Services stark abstrahieren und das weder Informationen zu vorhanden Verbindungen noch deren Installationen zugrunde liegen, benötigen die Services Implementierungen keine Unterstützung von Protokollen. Weiterhin bedarf ein Service kein Verständnis für das Nachrichten Routing mit anderen Diensten, sodass ein Service in einer ESB-fähigen Umgebung lediglich die ereignisgetriebene Nachricht des ESB empfangen und verarbeiten muss. Angepasste Integrationsdienste können als ESB Funktionalitäten entwickelt, erweitert und wiederverwendet werden. Des Weiteren können Services durch spezielle Integrationsdienste als zusammengefügte Anwendungsendpunkte realisiert werden oder Prozessabläufe können kombiniert und wiederverwendet werden, mit dem Zweck der Automatisierung von Geschäftsfunktionen, sodass Echtzeit-Unternehmen profitieren.[73]

### Prozessablauf

Die Leistungsfähigkeit der Prozessablauf Funktionalität in einem ESB erstreckt sich über einfach strukturierte Sequenzen für begrenzte Aufgabenbereiche, bis hin zu hochentwickelter Orchestrierung von Geschäftsprozessabläufen, welche auf parallele Prozessausführungen aufbaut. Ein Prozessablauf kann innerhalb eines ESB durch einfache Meta-Nachrichten oder einer Orchestrierungssprache, wie beispielsweise BPEL realisiert werden. Die Prozessablauf Funktionalität eines ESB kann einen lokalen Geschäftsprozess für spezielle Abteilungen oder Geschäftseinheiten definieren und diesen in komplexeren Integrationsnetzwerken ausbauen und wiederverwenden. Diese Möglichkeit kann alleine mit typischen BPM Werkzeugen nur schwer umgesetzt werden. Weiterhin kann der Prozessablauf in einem ESB individuelle Integration-Services umfassen, welche Nachrichten dem Inhalt entsprechend routen können. Aufgrund der Tatsache, dass sich die Prozessablauf Funktionalität an der Spitze einer verteilten Service orientierten Architektur befindet, stellt sie ebenfalls die Möglichkeit bereit, aufgespannte hochverteilte

71. vgl. (Chappell, 2004, S. 9f.)
72. vgl. (Chappell, 2004, S. 10)
73. vgl. (Chappell, 2004, S. 10)

Topologien zu realisieren, ohne auf physikalischen Netzwerkgrenzen oder vielfache Protokollsprünge zwischen Services und Anwendung zu achten.[74]

### Sicherheit und Zuverlässigkeit

Die Kommunikation zwischen zwei ESB Knoten kann über Firewalls realisiert werden. Des Weiteren verfolgt der ESB strikte Authentifikation, Anmeldeverfahren sowie Zugriffskontrollen, dabei spielt es keine Rolle, ob Verbindungen zwischen Anwendung und ESB oder unter den ESB Endpunkten gemeint sind.[75]

Die Zuverlässigkeit wird durch eine weitere Kernfähigkeit des ESB gestellt. Dabei handelt es sich um eine unternehmensfähige MOM, welche beispielsweise asynchrone Kommunikation oder Geschäftsdaten Übermittlung unterstützt. MOM ist ein Konzept, das die Weitergabe von Daten zwischen Anwendungen innerhalb einem Kommunikationskanal bereitstellt, welcher in sich geschlossene Informationseinheiten in Form von Nachrichten trägt. Im Rahmen einer MOM basierten Kommunikation werden Nachrichten asynchron versendet und empfangen. Weiterhin werden Anwendungen durch den Einsatz einer MOM abstrakt entkoppelt, so dass Empfänger und Sender keinerlei Informationen zueinander besitzen. Stattdessen werden lediglich Nachrichten zu einem Nachrichtensystem geschickt oder von diesem empfangen. Die MOM hat darauf als Aufgabe, die Nachrichten an den richtigen Bestimmungsort weiterzuleiten. Zudem ist das Nachrichten System für die Verwaltung der Verbindungspunkte zwischen mehreren Nachrichten Clienten zuständig sowie für die Bereitstellung mehrerer Kanäle für die Kommunikation zwischen den unterschiedlichen Verbindungspunkten. In der Regel werden Nachrichten Systeme als Software Prozess implementiert, sodass es einen sogenannten Nachrichten Server sowie einen Nachrichten Vermittler gibt.[76]

### Entfernte Konfiguration und Management

Einige Geschäftsmodelle benötigen nicht zwingend eigenes IT-Personal an jedem entfernten Standort. Als einfaches industrielles Beispiel kann eine Video Verleihkette zugezogen werden. Diese Unternehmensstruktur verfügt über hunderte oder auch tausende entfernte Standorte, welche alle die gleiche Reihe von Anwendungen nutzten sowie die gleiche Art von Operationen in Hinblick auf die Bestands-, Berichts- oder Kontoverwaltung. Mit der Unterstützung eines ESBs wird ein Integrationsentwurf etabliert, welcher die lokale Kommunikation zwischen Anwendungen am sogenannten Remote Store steuern kann. Unter anderem beinhaltet dieser die Schnittstellendefinitionen zu den Anwendungen, das Routing von Nachrichtenverkehr, Nachrichtenkanalverwaltung, sowie Sicherheitsgenehmigungen. Weiterhin können Integrationskomponenten enthalten sein, wie beispielsweise Anwendungsadapter, Protokolladapter oder Datentransformationskomponenten. Der Entwurf für entferntes Deployment ist nicht nur für Anwendungen im Einzelhandel nützlich, sondern ebenfalls für viele andere Anwendungen in der Industrie.

74. vgl. (Chappell, 2004, S. 11)
75. vgl. (Chappell, 2004, S. 12)
76. vgl. (Chappell, 2004, S. 77)

Folglich ist die einheitliche entfernte Management Funktion ein Kernfaktor für einen ESB in einer hochverteilten Umgebung.[77]

### XML als nativer Datentyp eines ESBs

Mit dem XML Standard ergibt sich ein idealer Datentyp für die Repräsentation eines Datenflusses zwischen Anwendungen innerhalb eines ESBs. Die konsumierten und produzierten Daten existieren in einer Reihe von unterschiedlichen Formaten und Verpackungsschemas, welche alle von einer Vielzahl von Anwendungen genutzt werden. Mit einem ESB besteht die Möglichkeit Daten in verpackter oder umhüllter Form zu verarbeiten, dennoch ist es ein immenser Vorteil, dass in-flight Daten, also Daten die sich gerade in der Übermittlung befinden, als XML repräsentiert werden. Dies führt weiterhin zu einer Unterstützung durch XML, als spezialisierte Dienste, welche ESB Daten aus verschiedenen Quellen kombinieren, um eine neue Ansicht der Daten zu modellieren sowie Nachrichten für erweiterten Datentausch zwischen Anwendungen zu bereichern und neu auszurichten. Das Synchronisieren von Upgrades zwischen Anwendungen entfällt durch den Einsatz von XML.[78]

### 5.1.1.6.2 Java Business Integration als ESB Container

Die Java Business Integration (JBI) dient der Beschreibung auf welche Art und Weise Komponenten integriert werden können, sodass beispielsweise ein ESB Service in ein herstellerneutrales und transportables System integriert werden kann (siehe Abbildung 8).[79]

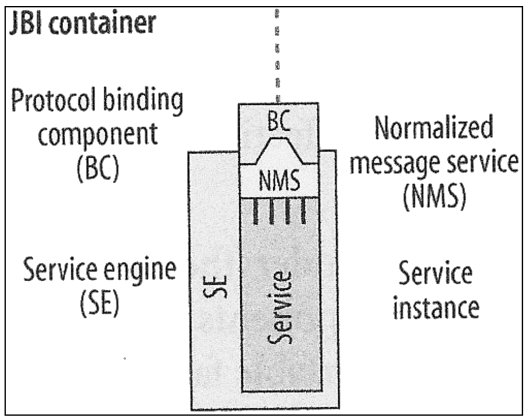



Abb. 8: JBI Modell[80]

Das JBI Modell in Abbildung 8 umfasst einen JBI Container, welcher über sogenannte Service Engines verfügt, welche wiederum den konkreten Service beinhalten. Der JBI Container verfügt über Protokollunabhängigkeit zwischen den einzelnen Binding Components (BC) und dem Normalized Message Service (NMS). Eine Binding Component wird als Schnittstelle an externe Protokolle angeboten und ist für jedes Protokoll speziell angepasst und in jeden JBI Container einsetzbar. Dies ermöglicht jeder JBI Komponente

77. vgl. (Chappell, 2004, S. 14f.)
78. vgl. (Chappell, 2004, S. 16)
79. vgl. (Ron Ten-Hove, Peter Walker, 2005, S. 7)
80. in Anlehnung an (Chappell, 2004, S. 184)

die Kommunikation, durch Einbinden in die Binding Component über etablierte Standards, wie beispielsweise JMS oder SOAP. Die Verantwortlichkeit der Binding Component besteht darin, die protokollspezifischen Nachrichten zu produzieren oder konsumieren und anschließend an den Normalized Message Service weiterzuleiten, welcher die Nachricht normalisiert und für die konsumierende Service Engine vorbereitet. Der NMS bietet ein gemeinsames Service Provider Interface (SPI) für alle Services und Service Engines (SE), sodass diese auf dem Interface schreiben können. Weiterhin soll mit Hilfe des NMS nicht der Nachrichten Payload definiert werden, sondern Versand und Erhalt von Nachrichten, sowie die Bereitstellung von Mitteln für Transport, Transaktionsinhalt oder sicherheitsrelevanter Inhalt zwischen Protokoll und Service Engine. Des Weiteren besteht die Möglichkeit den JBI Container über eine WSDL-Schnittstelle anzusprechen.[81]

JBI bietet Mittel für die Integration von Services, welche in einer verwalteten Umgebung gehostet werden. In einem JBI Container können steckbare Service Engines von Drittanbietern Interoperabilität untereinander in einem portablen System adressieren. Service Engines und ihre jeweiligen Services dienen der Integration und dem Prozess Management. Dies bedeutetet, dass eine SE als Orchestrierungs-Engine mit BPEL genutzt werden kann oder als Extensible Stylesheet Language Transformations (XSLT)-basierte Komponente, welche XML-Dokumente transformieren kann. Die Services können von unterschiedlichen Anbietern bereitgestellt werden und dennoch in einer verwalteten Umgebung zusammenarbeiten. Weiterhin können Service Engines sowie ESB Container über eine WSDL-Schnittstelle angesprochen werden. Die Binding Components können selber ebenfalls von unterschiedlichen Anbietern zur Verfügung gestellt werden.

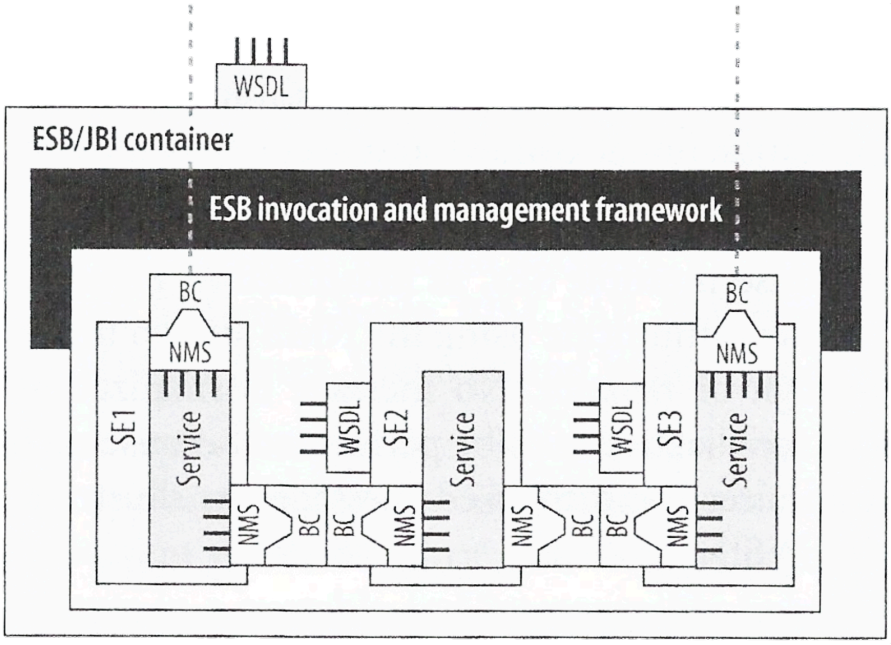



Abb. 9: ESB Container als JBI Container[82]

81. vgl. (Chappell, 2004, S. 184f.)
82. vgl. (Chappell, 2004, S. 186)

Als Beispiel kann Abbildung 9 hinzugezogen werden. In dieser Abbildung kann ein JMS Anbieter seine eigene JMS Binding Component für das Einbinden in die Nachrichtenebene anbieten. Der ESB Anbieter könnte hingegen den JBI Container bereitstellen und diesem ein Framework für Verwaltungs- und Aufrufaufgaben überordnen. Die übrigen Inhalte dieser Abbildung sind bereits in diesem Kapitel erläutert worden.[83]

## 5.1.2 Techniken und vorhandene Best Practices

Nachfolgend werden Techniken sowie Best Practice Ansätze für die prototypische Konzeption und Entwicklung anwendbarer Software vorgestellt. In diesem Zusammenhang werden beispielsweise grundlegende Techniken für die Verwendung einer Dokumentverwaltung in Form eines Repositories präsentiert, welches die zentrale Komponente für Verwaltungsaufgaben von Prüfvorgängen darstellt. Anschließend folgen Techniken zur Erstellung unterschiedlicher Versionierungen der Dokumente innerhalb des Repositories, durch den Einsatz von Hash-Algorithmen. Daraufhin sollen Best Practices zur Umsetzung einer Differenzvisualisierung folgen, welche zwischen verschiedenen Versionen einer Datei, Änderungen visualisieren soll. Nach den Best Practices für das Werkzeug der Differenzvisualisierung werden die aktuellen Techniken zum Werkzeug Traceability formuliert, welches Spezifikation und Quellcode in Dokumenten finden soll und eine Verknüpfung von Anforderungen zu korrespondierenden Implementierungen durchführt. Diesbezüglich wird ebenfalls eine Erläuterung für die Spezifikationsgenerierung gegeben, mit dessen Unterstützung Spezifikationen in ein hybrides Format gespeichert werden können, sodass diese von Menschen sowie Maschinen lesbar sind. Die weiteren Inhalte beziehen sich zunächst auf den Stand der Technik des prototypisch entwickelten Word Spezifikationsvorlage, welches eine Methodik zur Verfügung stellen soll, womit Strukturen von Software-Spezifikationen erfasst und auf Basis dieser eine Formularvorlage erstellt werden kann. Abschließend sollen alle benötigten Ansätze für die Erstellung Browser basierter Applikationen, wie beispielsweise das User Interface, dem Comment-System sowie der Traceability vermittelt werden.

### 5.1.2.1 Speicherarchitektur zur Speicherung der Daten

Ein Repository erfüllt im Bereich der Dokumentverwaltung die Aufgabe sämtliche anfallenden Daten einen Systems zu speichern und liegt zumeist in Form eines Verzeichnisbaumes auf Dateiebene, als Datenbanklösung oder einer Kombination aus beiden Varianten vor. Benutzer können die im Repository enthaltenen Daten aufrufen, ändern oder neue hinzufügen. Dabei arbeitet ein Benutzer immer auf Grundlage der aktuellsten Version einer Datei. Das Repository übernimmt in diesem Zusammenhang die Aufgabe alle Änderungen an den gespeicherten Daten zu protokollieren und diese zu speichern. Weiterhin ist es dafür zuständig, die gespeicherten Änderungen in Form von Versionen vorzuhalten, sodass auch ältere Versionen problemlos eingesehen oder revidiert werden können.[84]

---

83. vgl. (Chappell, 2004, S. 185)

84. vgl. (Collins-Sussman, Fitzpatrick, Lichtenberg, & Pilato, 2007, S. 10) & vgl. (Haenel & Plenz, 2011, S. 19f.)

Bei der Speicherverwaltung wird in zwei mögliche Architekturen unterschieden, der zentralen oder dezentralen Speicherung von Daten, in denen alle anfallenden Daten eines Benutzers oder mehrerer Benutzer verwaltet werden.

### Zentrale Speicherung

Bei einer zentralen Speicherung bildet ein einzelnes Repository den Kern des Systems, in dem alle Daten vorgehalten werden. Eine beliebige Anzahl von Benutzern kann auf die gespeicherten Daten zugreifen und mit diesen arbeiten. Beispiele für diese Form der Speicherungen sind Datenbanken oder gemeinsam genutzte Speicherpunkte auf Dateisystemebene.[85]

### Dezentrale Speicherung

Bei der dezentralen Speicherung hingegen verfügt jeder Benutzer über ein persönliches Repository mit oben genannter Funktionalität, wobei ein Benutzer entscheidet, welche Projektdaten aufgenommen werden, sodass dieser unabhängig vom Netzwerk arbeiten kann. Die Lokalität verwalteter Daten kann mit Hilfe geeigneter Methoden aufgelöst werden, sogenannten Push-und-Pull Methoden, indem Änderungen von Daten in andere Repositories übermittelt oder von anderen Repositories herangezogen werden. Dadurch wird es ermöglicht abweichende Datenbestände aus verschiedenen Repositories untereinander synchron zu halten, indem diese miteinander verglichen und Abweichungen zwischen den Repositories ausgetauscht werden. Weiterhin kann eine Isolation von Repositories realisiert werden, indem auf die genannten Austauschmethoden verzichtet wird, sodass eine physische Trennung der Repositories zwischen Benutzern möglich wird. Wie auch in der zentralen Speicherung können die Daten in Datenbanken oder auf lokaler Dateisystemebene vorgehalten werden, mit dem Unterschied, dass diese voneinander isoliert sind, sodass kein gemeinsamer Zugriff möglich ist.[86]

Die Versionierung von Dateien stellt im Rahmen der Dokumentverwaltung eine wesentliche Rolle dar, insbesondere weil von Benutzern, im Verlauf der Arbeitstätigkeiten, neue Dokumente angelegt, kommentiert oder verändert werden. Folglich ist für einzelne Dokumente von einer Vielzahl vorliegender Versionen auszugehen. Veränderungen nachzuhalten sollte dabei nicht Aufgabe des Lesers oder Autors der Dokumente sein, sondern möglichst technisch gelöst werden. Bei der Versionierung besteht gegebenenfalls das Problem, verschiedenen Benutzern die Arbeit an den selben Dokumenten zu gewähren, ohne dabei Änderungen der anderen Benutzer zu überschreiben. Das Problem der wechselseitigen Änderungen an Dokumenten durch verschiedene Benutzer kann mit passenden Modellen aufgelöst werden. Zwei wichtige Modelle werden hierzu nachfolgend erläutert.[87]

### 1. Lock-Modify-Unlock

Eine einfache Lösung stellt das Lock-Modify-Unlock Modell dar, bei dem jeweils nur ein einzelner Benutzer gleichzeitig an einem Dokument arbeiten kann. Wenn ein Benutzer eine nicht gesperrte Datei

85. vgl. (Collins-Sussman et al., 2007, S. 10)
86. vgl. (Gürsoy, 2011, S. 3)
87. vgl. (Collins-Sussman et al., 2007, S. 10)

öffnet, wird diese in einen blockierenden Status ("Lock") versetzt. Die Datei gilt in diesem Zustand für andere Benutzer als gesperrt, sodass eine Bearbeitung nicht möglich ist. Der bearbeitende Benutzer hat in diesem Zustand die alleinigen Rechte die Dateiinhalte zu verändern ("Modify"). Erst wenn dieser Benutzer die bearbeitete Datei wieder freigibt ("Unlock"), indem die Datei beispielsweise gespeichert oder konform geschlossen wird, können andere Benutzer damit weiterarbeiten. In diesem Zeitraum ist für andere Benutzer nur ein lesender Zugriff auf die blockierte Datei möglich. Sperrungen einer Datei führen deshalb zu einer Serialisierung, da immer nur eine einzelne Person am entsprechenden Dokument arbeiten kann. Dabei spielt es keine Rolle, ob sich die Änderungen inhaltlich überschneiden oder nicht. Die Sperrung einer Datei kann jedoch zu administrativen Problemen führen, sofern ein Benutzer vergisst eine bearbeitete Datei wieder freizugeben. Wird eine Datei in einem blockierten Zustand belassen, kann diese von keinem anderen Benutzer bearbeitet werden. In diesem Fall kann nur ein Benutzer mit administrativen Rechten eine Sperre manuell aufheben. Weiterhin vermitteln Sperrungen eine falsche Sicherheit, da Benutzer davon ausgehen, gekapselte Dokumente zu bearbeiten. Durch eine Sperrung wird bei abhängigen Dateien nicht gewährleistet, dass die erarbeiteten Ergebnisse auch kompatibel sind.[88]

### 2. Copy-Modify-Merge

Bei der Copy-Modify-Merge Methode arbeitet jeder Benutzer mit einer eigenen Arbeitskopie, weshalb mehrere Benutzer an einem Dokument parallel arbeiten können. Zu diesem Zweck wird parallel zum Aufruf einer Datei eine Kopie ("Copy") erstellt und für den Benutzer zur Bearbeitung ("Modify") bereitgestellt. Abschließend wird die Arbeitskopie mit dem Original zusammengeführt ("Merge"), wobei die Kontrolle der Zusammenführung dem Benutzer obliegt. Sofern sich keine Änderungen zwischen der Kopie und dem Original überschneiden, können diese einfach in das Original integriert werden, ansonsten muss der Benutzer entscheiden, welche der vorliegenden Versionen übernommen wird. Somit kann das Zusammenführen verschiedener Versionen weitestgehend nur manuell erfolgen, wobei die Komplexität steigt, je mehr Versionen beteiligt sind. Bei schlechter Absprache zwischen Benutzern entstehen mehr logische Konflikte, die anschließend manuell gelöst werden müssen. Weiterhin können Änderungen in Grafikdateien nicht nachvollzogen werden, wodurch beim Zusammenführen Informationen einer Version verloren gehen können.[89]

Für eine gute Versionierung existieren eine Reihe von Kriterien, die bei der Realisierung eines derartigen Systems berücksichtigt werden müssen. Ein Beispiel hierfür ist die Erstellung eindeutiger Bezeichner für unterschiedliche Versionen. Diese können etwa nach der Major-/Minor-/Micro-Methode, dem Date-Release oder durch Hash-Algorithmen gebildet werden. Weitere Kriterien entstehen aus Anforderungen an Funktionen, Arbeitsprozesse und Bedürfnisse an die Usability durch beteiligte Benutzer, wie auch Daneben existieren technische Aspekte, welche den Funktionsumfang einer Versionierung ebenso beein-

88. vgl. (Collins-Sussman·et·al.,·2007,·S.·10ff.)
89. vgl. (Collins-Sussman·et·al.,·2007,·S.·12ff.)

flussen. Aber auch Kostenargumente müssen berücksichtigt werden. Detaillierte Informationen hierzu können in der ausgewiesenen Literatur vertieft werden.[90]

Im Rahmen einer Prüfung entstehen rechtlich bindende Dokumente, die damit den gesetzlichen Vorgaben zur Aufbewahrung und Archivierung von Dokumenten entsprechen müssen. Ein Problem bei der Langzeitarchivierung derartiger Daten resultiert aus der Eigenschaft digitaler Dokumente ohne passenden Interpreter, wie etwa Word, nicht lesbar zu sein. Darüber hinaus müssen Informationen über die Integrität und Authentizität verfügbar sein. Diese Anforderungen müssen bei der Archivierung mindestens für die gesetzlich vorgeschriebene Aufbewahrungsdauer berücksichtigt und eingehalten werden. Mit Hilfe der technischen Richtlinie zur vertrauenswürdigen Langzeitspeicherung vom Bundesamt für Sicherheit in der Informationstechnik (BSI) liegt eine Empfehlung vor, auf deren Grundlage eine langfristige Archivierung von digitalen Daten realisiert werden kann. Dabei wird in der Richtlinie beschrieben, mit welchen Techniken, Komponenten, Schnittstellen, etc., eine einheitliche langfristige Archivierung aufgebaut und betrieben werden kann. Das BSI definiert weiterhin geeignete Datenformate auf Basis langfristig relevanter Kriterien, wie etwa XML, PDF, JPEG und weitere, die für eine Archivierung vorgezogen werden sollten. Zusätzlich empfiehlt das BSI, Dokumente gleichzeitig in mehreren Formaten zu archivieren, damit eine spätere Reproduktion mit höherer Wahrscheinlichkeit ermöglicht wird.[91]

#### 5.1.2.2 Hash-Algorithmen zur Bildung von Prüfsummen

Prüfsummen werden bei der Verarbeitung von digitalen Daten zum Einsatz gebracht, um für diese eindeutige Fingerabdrücke zu erzeugen. Mit entsprechenden Hash-Algorithmen, wie beispielsweise Message-Digest Algorithm 5 (MD5), oder Secure Hash Algorithm (SHA) wird dazu über den Inhalt eines digitalen Dokumentes eine Prüfsumme berechnet. Ein Hash-Algorithmus bildet die einzelnen Bytes eines Dokumentes, nach festen Berechnungsvorschriften, auf eine Zeichenkette gleichbleibender Länge ab, die sich anschließend effizient vergleichen lässt.[92]

Durch den konsequenten Einsatz der Prüfsummenbildung können digitale Dokumente, wie etwa Textdokumente oder auch Grafiken, effizient auf Änderungen geprüft werden, ohne diese manuell auf Modifikationen untersuchen zu müssen. Bereits minimale Änderungen des Inhaltes eines digitalen Dokumentes, wie das Entfernen oder Hinzufügen eines Zeichens führt zu einem meist stark differierenden Ergebnis bei der Bildung von Prüfsummen. Im Folgenden einfachen Beispiel (Abbildung 10) wird das Ergebnis einer Prüfsummenberechnung mit dem MD5-Berechnungsverfahrens deutlich, die bei einer Veränderung eines einzelnen Buchstaben hervorgerufen wird. So führt der Austausch des Buchstaben "D" durch den Buchstaben "B" zu einer vollkommen anderen Prüfsumme. Zu beachten ist hierbei, das die Länge der Prüfsumme immer gleich bleibt, unabhängig davon wie umfangreich oder komplex das Dokument ausfällt, über welches die Prüfsumme gebildet wird.

90. vgl. (Haenel & Plenz, 2011, S. 70) & vgl. (Gürsoy, 2011, S. 3f.)
91. vgl. (Helmbrecht, 2009, S. 3f.)
92. vgl. (Rivest, 1992) & vgl. (Eastlake 3rd & Jones, 2001)

```
MD5("Der Bauer sitzt im Dach") = cebd4f10259c4b398d8b796d321f4f56
MD5("Der Bauer sitzt im Bach") = 1bdc8a1a2e18e872b5bb4bf455949b9c
```

Abb. 10: Berechnung von zwei Prüfsummen mit minimaler Textveränderung

Eine andere Möglichkeit stellt die Überprüfung von Datenpaketen in der Datenübertragung dar, die immer weiter in den Fokus rückt, da die papierbasierte Arbeit in Zukunft weiter abnehmen wird. Hierbei unterstützen Prüfsummen bei der Feststellung auf korrekte Datenübertragung. Zu diesem Zweck wird über die einzelnen Datenpakete oder Dateien, die vom Sender übertragen werden sollen, ein Hash-Wert gebildet und mit den Datenpakten oder Dateien an den Empfänger übertragen. Auf der Empfängerseite wird wiederum ein Hash-Wert über die eingegangenen Informationen gebildet und die generierten Hash-Werte mit denen des Senders verglichen. Sofern die Prüfsummen von Sender und Empfänger identisch ausfallen, liegen dem Empfänger mit sehr hoher Wahrscheinlichkeit die selben Informationen vor, wie dem Sender.[93]

Ein wichtiges Kriterium bei der Bildung von Prüfsummen mittels Hash-Algorithmus stellt die Eindeutigkeit dar, mit der eine Prüfsumme aus einem Quelldokument generiert wird. Hierbei stehen vor allem gezielte Manipulationen an einem Dokument im Vordergrund, die den Inhalt verändern, aber dennoch zu einer gleichen Prüfsumme führen könnten. Einfache Hash-Wert Verfahren, wie etwa Quersummen fallen aus diesem Grund für eine Verwendung weg, da die Berechnung zu wenig Spielraum für eindeutige Prüfsummen bietet und von Manipulatoren leicht ausgenutzt werden könnten. Algorithmen, wie MD5 oder SHA stellen für die Prüfsummenberechnung eine bessere Wahl dar, da diese nicht auf den Inhalt eines Dokumentes schließen lassen und aus diesem Grund sicher gegen einfache Manipulationen sind. Weiterhin zeichnen sich derartige Algorithmen durch eine größere Abbildungslänge aus, die im obigen Beispiel der Abbildung 10, genau 128 Bit in Form einer 32-stelligen Hexadezimalzahl entspricht. Durch längere Hash-Werte sinkt die Wahrscheinlichkeit einen gleichen Wert über zwei Dokumente mit verschiedenem Inhalt zu generieren, wodurch die Sicherheit weiter steigt. Aus diesem Grund sind SHA Algorithmen zu bevorzugen, die üblicherweise in 256 Bit oder 512 Bit Länge generiert werden und dem aktuellen Standard entsprechen. Diese Hash-Werte sind doppelt, bzw. viermal so umfangreich in ihrer Länge, im Vergleich zu MD5, wodurch die Wahrscheinlichkeit gleiche Hash-Werte zu generieren für SHA (256 Bit) weiter sinkt. Mathematisch ausgedrückt liegt die Wahrscheinlichkeit zwei gleiche 256 Bit SHA Hash-Werte aus 64 Zeichen zu bilden bei null (siehe Abbildung 11), wobei das Hexadezimalsystem den Zeichenraum auf 16 Zeichen beschränkt.[94]

$$P = \frac{n}{16^{64}}, bei\ n = 2 \approx -1,73 \times 10^{-77}$$

Abb. 11: Wahrscheinlichkeit von SHA-2 (256 Bit) zwei gleiche Hash-Werte zu bilden

93. vgl. (Eckert, 2013, S. 381ff.)
94. vgl. (Eckert, 2013, S. 389ff.)

Durch die beispielhafte Berechnung von Abbildung 11 wird diese These rechnerisch belegt. Selbst bei vielen Milliarden Hash-Werten würde sich der Exponent im Ergebnis nur um wenige Stellen verschieben und kann folglich gleich null gesetzt werden. Grundsätzlich erfolgt die Berechnung von MD5-Prüfsummen gegenüber SHA-Prüfsummen schneller, wobei die Unterschiede marginal ausfallen. Weiterhin ist es aber nicht weiter notwendig noch längere Hash-Werte zu bilden, wie etwa SHA (512 Bit), da durch die steigende Komplexität der Algorithmen die Berechnungsdauer weiter verlangsamt wird. Folgerichtig fallen längere Hash-Werte dadurch aus dem Raster der Verwendung. SHA-Algorithmen stellen deshalb den besten Kompromiss zwischen Berechnungsgeschwindigkeit und Sicherheit dar, sodass die Wahl des zu verwendenden Algorithmus auf SHA in einer 256 Bit Version fallen sollte.[95]

#### 5.1.2.3 Werkzeuge zur Differenzbildung

Mit Hilfe eines Werkzeuges zur Berechnung von Differenzen (Differenz-Tool), das für SIWOB entwickelt und an den BUS gekoppelt wird, soll es möglich werden, zwischen verschiedenen Versionen eines Dokumentes, Änderungen auf dem Graphical User Interface (GUI) für einen Prüfer zu visualisieren. Grundsätzlich sollen alle anfallenden Dokumente, sowohl strukturierte Code-Formate, wie beispielsweise C- oder Java-Programme, aber auch komplexe Formate, wie beispielsweise Konzeptbeschreibungen, in denen auch Spezifikationen enthalten sind, auf Änderungen zwischen Versionen analysiert werden können, damit diese nicht manuell gesucht werden müssen.

Zur Vereinfachung werden vorerst ausschließlich idealtypische Formate berücksichtigt, die eine erste prototypische Implementierung erlauben. Ein Beispiel hierfür stellen C-Programme dar, die eine Vielzahl des anfallendes Codes in einer Software-Prüfung abdecken. Derartiger Programm-Code liegt grundsätzlich immer im Plain-Text Format mit festen Strukturen vor, wie etwa Klassen, Methoden, Kommentare und weiteren logischen Konstrukten, sodass die jeweilige Programmiersprache die Grenzen der lesbaren Vielfalt definiert. Formatierungen in Schriftart und Schriftgröße werden bei Programmdateien nicht eingebettet, sondern durch die jeweilige Entwicklungsumgebung gesichert, sodass reine Textdateien zum Vergleich vorliegen. Eine Vergleichsanalyse verschiedener Versionen ist aus diesem Grund mit geringem Aufwand realisierbar.

Bei Spezifikationen hingegen erschwert sich eine Differenzbildung aufgrund der komplexen Dokumentstrukturen, da diese zumeist in Formaten wie Word oder PDF eingereicht werden. Zu diesem Zweck muss zuerst ein passendes Format generiert werden, im Idealfall gemäß einer ISO-Norm, auf dessen Basis eine Vergleichsanalyse zwischen den Dokumentversionen unter diesen Idealbedingungen erfolgen kann. Gleichwohl sind auch Analysen natürlichsprachlicher Prosatexte möglich, sofern diese neben den Textinformationen keine zusätzlichen Meta-Informationen enthalten und somit weniger Struktur aufweisen. Grenzen werden im Sinne der Konzeptbeschreibungen und Spezifikationen durch die verwendeten Dokumentformate gestellt, da diese nicht in einfachen Formaten, wie etwa *.txt Dateien anfallen, sondern in "Container"-Formaten vorliegen, die zusätzliche Informationen neben dem eigentlichen Textinhalt beinhalten, wie etwa Formatierungsanweisungen. Diese Container-Formate in die notwendigen Bestandtei-

95. vgl. (Passing·&·Dressler,·2006,·S.·885f.)

le für die Differenzbildung zu zerlegen, damit ausschließlich der Textinhalt verfügbar wird, stellt eine große Herausforderung dar.

Zu Beginn stellt sich die Frage, nach welchem Ansatz eine Entwicklung zur Differenzbildung zwischen zwei Dokumentversionen erfolgen soll. Hierbei ist es im Wesentlichen möglich eine vollständig neue Entwicklung zu realisieren, um die gesetzten Ziele der Differenzbildung zu erreichen. Durch die eigene maßgeschneiderte Entwicklung wird es möglich auf die Individuellen Anforderungen in der eingesetzten Umgebung eingehen zu können. Dies betrifft insbesondere das Einsatzfeld in Form eines Web Service mit Hilfe von Java EE, sodass eine Anbindung an den SIWOB überhaupt möglich wird, womit im gleichen Zug die Programmiersprache für die Entwicklung des Werkzeuges zur Differenzbildung, in Form eines in Java geschriebenen Web Service, aufoktroyiert ist.

Allerdings sollten bewährte Algorithmen und Konzepte zur Bildung von Differenzen nicht außer acht gelassen werden und vor Beginn der prototypischen Entwicklung, in einer getrennten Analyse, eingehend untersucht werden. Die Analyse soll bestehende Konzepte dahingehend untersuchen, inwiefern diese als Ausgangspunkt für eine Neuentwicklung verwendet werden können oder diejenigen Algorithmen identifizieren, die bei der prototypischen Entwicklung eingesetzt werden können. In diesem Zusammenhang werden ausschließlich Konzepte betrachtet, die auf Open-Source Ansätzen basieren, damit eine Verwendung von Algorithmen, Programm-Code oder Strukturen im SIWOB-Projekt durch die Open-Source Lizenz gewährleistet werden kann. Zu diesem Zweck erfolgt in den folgenden Kapiteln eine Zusammenfassung zu den betrachteten Algorithmen und Konzepten, Google Diff-Patch-Match, Java Diff-Utils, Daisy-Diff und GNU Diffutils zur Bildung von Differenzen zwischen Dokumentversionen.

### Google Diff-Patch-Match

Google Diff-Patch-Match stellt eine Library dar, mit der Plain-Text Dokumente auf Unterschiede verglei-chen werden können. Eine nähere Aufschlüsselung der grundlegenden Software-Details erfolgt in Tabelle 1. Alle in diesem Kapitel zusammengefassten Informationen stammen aus der gleichen Quelle, soweit ein Nachweis nicht durch einen anderen Literaturbeleg erfolgt.[96]

| Merkmal | Beschreibung |
|---|---|
| Name | Google Diff-Patch-Match |
| Software | Open Source |
| Lizenz | Apache License 2.0 |
| Code | Java, C, C#, Objective-C, Python und weitere |
| Download | https://code.google.com/p/google-diff-match-patch/downloads/list |
| Dokumentation | https://code.google.com/p/google-diff-match-patch/wiki/API |
| Version | Revision 103 (19. November 2012) |

96. vgl. (Fraser, 2012)

| | |
|---|---|
| Diff-Algorithmus | Myer's Diff Algorithmus[97] |
| Einbindung | Über ein externes Paket in die Java Library<br>Über pom.xml mittels Repository Adresse und Dependency Download |

| | |
|---|---|
| Eingabe | Die Funktion erwartet als Eingabe zwei Strings in denen die Informationen für den Vergleich gespeichert sind.<br>Hierzu müssen zuvor die zu vergleichenden Dokumente in diese zwei Strings eingelesen und anschließend zur Auswertung an die Schnittstelle zur Differenzberechnung übergeben werden. |
| Verarbeitung | Die Schnittstelle berechnet gemäß Myer's Diff Algorithmus die Differenzen und gibt als Rückgabewert das Ergebnis zurück.<br>Der Algorithmus ermöglich die Lokalisierung von Delete, Insert und Equal.<br>Die Rückgabe erfolgt grundsätzlich in Form einer verknüpften Liste mit den Differenzeinträgen (LinkedList<Diff>) die mit passenden Methoden für eine Ausgabe formatiert werden kann. |
| Ausgabe | In verschiedenen Ansichten als Plain oder HTML Ausgabe möglich, in Abhängigkeit der gewählten Methode.<br>• Plain (@return: LinkedList<Diff>): Zeilenweise Ausgabe der Ergebnisse.<br>• HTML (@return: String): Ausgabe als fertig formatierter HTML Block, bei dem Modifikationen ausschließlich in der Kernfunktion erfolgen können. |

Tabelle 1: Allgemeine Informationen zu Google Diff-Patch-Match

### Java Diff Utils

Java Diff Utils stellt eine Library dar, mit der Plain-Text Dokumente auf Unterschiede vergleichen werden können. Eine nähere Aufschlüsselung der grundlegenden Software-Details erfolgt in Tabelle 2. Alle in diesem Kapitel zusammengefassten Informationen stammen aus der gleichen Quelle, soweit ein Nachweis nicht durch einen anderen Literaturbeleg erfolgt.[98]

| **Merkmal** | **Beschreibung** |
|---|---|
| Name | Java Diff Utils |
| Software | Open Source |
| Lizenz | Apache License 2.0 |
| Code | Java |
| Download | https://code.google.com/p/java-diff-utils/downloads/list |
| Dokumentation | https://code.google.com/p/java-diff-utils/downloads/list |
| Version | 1.3.0 (04. März 2013) |
| Diff-Algorithmus | Myer's Diff Algorithmus[99] |

97. vgl. (Myers, 1986)
98. vgl. (Naumenko, 2013)
99. vgl. (Myers, 1986)

| | |
|---|---|
| Einbindung | Über ein externes Paket in die Java Library<br>Über pom.xml Dependency Download |

| | |
|---|---|
| Eingabe | Die Funktion erwartet als Eingabe zwei Array-Listen (List<String>) in denen die Informationen für den Vergleich gespeichert sind.<br>Deshalb müssen die zu vergleichenden Dokumente zuvor in diese zwei Array-Listen eingelesen und anschließend zur Auswertung an die Schnittstelle zur Differenzberechnung übergeben werden. |
| Verarbeitung | Die Schnittstelle berechnet gemäß Myer's Diff Algorithmus die Differenzen und gibt als Rückgabewert das Ergebnis zurück.<br>Der Algorithmus ermöglich die Lokalisierung von Delete, Insert und Equal.<br>Die Rückgabe erfolgt grundsätzlich in Form einer Array-Liste mit den Differenzeinträgen (List<String>). |
| Ausgabe | Die Ausgabe erfolgt als Plain-Text.<br>• Plain (@return: List<String>): Zeilenweise Ausgabe der Ergebnisse, wobei die Ausgabe für das Frontend durch passende Erweiterungen formatiert werden kann. |

Tabelle 2: Allgemeine Informationen zu Google Java Diff Utils

### Daisy Diff

Daisy Diff stellt eine Library dar, mit der Plain-Text Dokumente auf Unterschiede vergleichen werden können. Eine nähere Aufschlüsselung der grundlegenden Software-Details erfolgt in Tabelle 3. Alle in diesem Kapitel zusammengefassten Informationen stammen aus der gleichen Quelle, soweit ein Nachweis nicht durch einen anderen Literaturbeleg erfolgt.[100]

| Merkmal | Beschreibung |
|---|---|
| Name | Daisy Diff |
| Software | Open Source |
| Lizenz | Apache License 2.0 |
| Code | Java |
| Download | https://code.google.com/p/daisydiff/downloads/list |
| Dokumentation | - |
| Version | 1.2.0 (09. Oktober 2011) |
| Diff-Algorithmus | - |
| Einbindung | Über ein externes Paket in die Java Library |

| | |
|---|---|
| Eingabe | Die Funktion erwartet als Eingabe zwei Web-Adressen oder lokale Adressen zu HTML Dokumenten, die verglichen werden sollen. |

100. vgl. (Van·den·Broeck,·2011)

| | |
|---|---|
| Verarbeitung | Die Schnittstelle berechnet die Differenzen und gibt als Rückgabewert das Ergebnis zurück.<br>Der Algorithmus ermöglich die Lokalisierung von Delete, Insert und Equal.<br>Die Rückgabe erfolgt grundsätzlich in Form einer HTML oder Extensible Stylesheet Language (XSL) Struktur. |
| Ausgabe | Die HTML oder XSL Struktur kann anschließend in ein Dokument überführt und/oder angezeigt werden. |

Tabelle 3: Allgemeine Informationen zu Daisy Diff

Da mit Daisy Diff ausschließlich ein Vergleich von HTML Quellen möglich ist und zusätzlich der Algorithmus zur Berechnung von Differenzen nicht ersichtlich ist, erscheint diese Library als gänzlich ungeeignet und scheidet für eine Verwendung vollständig aus.

### GNU Diffutils

GNU Diffutils stellt eine systemnahe UNIX-Library dar, mit der Plain-Text Dokumente auf Unterschiede vergleichen werden können. Eine nähere Aufschlüsselung der grundlegenden Software-Details erfolgt in Tabelle 4. Alle in diesem Kapitel zusammengefassten Informationen stammen aus der gleichen Quelle, soweit ein Nachweis nicht durch einen anderen Literaturbeleg erfolgt.[101]

| Merkmal | Beschreibung |
|---|---|
| Name | Diffutils |
| Software | Open Source |
| Lizenz | GNU General Public License |
| Code | C |
| Download | http://ftp.gnu.org/gnu/diffutils/ |
| Dokumentation | http://www.gnu.org/software/diffutils/manual/ |
| Version | 3.3 (24. März 2013) |
| Diff-Algorithmus | Myer's Diff Algorithmus[102] |
| Einbindung | nicht möglich, da UNIX-Implementierung |

| | |
|---|---|
| Eingabe | Die Funktion erwartet als Eingabe zwei Adressen zu lokalen Dokumenten, die verglichen werden sollen. |
| Verarbeitung | Die Schnittstelle berechnet gemäß Myer's Diff Algorithmus die Differenzen und gibt als Rückgabewert das Ergebnis zurück.<br>Der Algorithmus ermöglich die Lokalisierung von (D)elete, (A)ppend und (C)opy. |

101. vgl. (Free Software Foundation, 2013)

102. vgl. (Myers, 1986)

| | |
|---|---|
| Ausgabe | Die berechneten Differenzen können auf verschiedene Wege ausgegeben werden.<br>• Auf dem Terminal<br>• In einer Datei<br>Es besteht die Möglichkeit die Ausgabe rudimentär zu formatieren, beispielsweise in einer zweispaltigen Ansicht oder mit einer Anreicherung von Zeileninformationen. |

Tabelle 4: Allgemeine Informationen zu GNU Diffutils

Mit der GNU Diffutils Library ist ausschließlich auf UNIX-Systemen, über Terminal oder Software, ein Vergleich von Dokumenten möglich. Aufgrund dieser Einschränkung, der Implementierung in C und einer fehlenden Portierung in Java, wird diese Library als ungeeignet eingeordnet.

#### 5.1.2.4 Methoden zur Differenzvisualisierung

Die wesentliche Entscheidung bei der Berechnung von Differenzen zwischen Dokumentversionen betrifft die Granularität des Differenzvergleiches in Bezug auf die Länge der eingelesenen Zeichenketten, die maßgeblich die visuelle Ausgabe für den Anwender beeinflusst.

Die Differenzbildung zwischen zwei Dokumenten kann drei verschiedene Möglichkeiten identifizieren, die unterschiedlich visualisiert werden müssen, die jeweils Vorteile und Nachteile aufweisen und in den nachfolgenden Kapiteln genauer beschrieben werden. Hierbei handelt es sich um die Identifizierung von entfernten, hinzugefügten oder unveränderten Textinformationen. Entfernte Informationen werden im Ergebnis durch rote Markierungen hervorgehoben, wohingegen hinzugefügte Informationen durch grüne Markierungen herausgestellt werden. Sollte keine Änderung ermittelt werden, so wird die entsprechende Information durch keinerlei Markierung hervorgehoben. Hierbei wäre es sogar denkbar die unveränderten Informationen herauszufiltern, damit lediglich eine kompakte Übersicht der Änderungen visualisiert werden kann.

##### Zeichenbasierter Vergleich

Diese Methodik ließt einzelne Zeichen (Charakter) aus Dokumenten ein, sodass ein Vergleich auf Zeichenebene realisiert werden soll. In Abbildung 12 werden zwei Textbeispiele aus verschiedenen Dokumentversionen gezeigt, die einen kleinen Auszug darstellen und Veränderungen beinhalten, die identifiziert werden sollen. Hierbei beinhaltet der erste Absatz der Text Version 1 eine starke Abweichung zum ersten Absatz der Text Version 2 und die zweiten Absätze eine geringe Abweichung. Hierzu wird das erste eingelesene Zeichen des ersten Dokumentes mit dem ersten Zeichen des zweiten Dokumentes verglichen und auf eine Änderung überprüft. Der Vergleich wird solange durchgeführt, bis alle Inhalte der Dokumente vollständig verarbeitet wurden.

**Text Version 1**

Wer würde ihm schon folgen, spät in der Nacht und dazu noch in dieser engen Gasse mitten im übel beleumundeten Hafenviertel?

Oder geörten die Schritte hinter ihm zu einem der unzäligen Gesezeshüter der Stadt, und die stählerne Acht um seine Handgelenke würde gleich zuschnapen?

**Text Version 2**

Hatte einer seiner zahllosen Kollegen dieselbe Idee gehabt, ihn beobachtet und abgewartet, um ihn nun um die Früchte seiner Arbeit zu erleichtern?

Oder gehörten diese Schritte hinter ihm zu den unzähligen Gesetzeshütern dieser Stadt, und die stählerne Acht um seine Handgelenke würde gleich zuschnappen?

Abb. 12: Text-Beispiele aus einem versionierten Dokument

Durch den Vergleich einzelner Zeichen ist es möglich, kleinste Änderungen in Dokumenten zu identifizieren, beispielsweise wenn kleine Rechtschreibfehler im Dokument durch den Verfasser korrigiert wurden. Sind Zeichen identisch, werden die nächsten Zeichen analysiert. Sollte eine Änderung lokalisiert werden, wird diese durch die entsprechende Markierungen, für entfernte, hinzugefügte oder unveränderte Informationen, sichtbar dargestellt (siehe Abbildung 13). Hierbei besteht die Schwierigkeit darin zu ermitteln, über welchen Bereich der Dokumente sich Änderungen erstrecken, sodass ein lesbares Differenzergebnis visualisiert werden kann.

**Ergebnis der Differenz (zeichenbasiert)**

~~W~~Hatte einer ~~würd~~se ~~ihm s~~ener zahllosen ~~f~~Kollegen~~,~~
diespäelbe Idee gehabt, ihn ~~der N~~beobachtet und
~~da~~zbgewartet, um ihn~~och~~ ~~i~~nun um die~~ser eng~~Früchte~~n Ga~~sse
mitte~~n im übe~~lr Arbe~~leumundeten~~it zu ~~Hafenvi~~erleichtelrn?¶
¶
Oder gehörten diese Schritte hinter ihm zu~~ einem~~ de~~r~~n
unzähligen Gesetzeshütern dieser Stadt, und die stählerne
Acht um seine Handgelenke würde gleich zuschnappen?

Abb. 13: Visualisierung der berechneten Differenz (zeichenbasiert) der Text Versionen

Diese Vergleichsform ist sehr rechenaufwändig, da jedes einzelne Zeichen der Dokumente eingelesen und auf eine Veränderung überprüft werden muss, was bei umfangreichen Dokumenten zu zeitaufwendigen Operationen führen kann. Ein weiteres Problem besteht in der Darstellung der Ergebnisse, die sehr kryptisch ausfallen kann, sofern viele Änderungen zwischen den Vergleichsdokumenten vorliegen, wie im ersten Absatz der Abbildung 13 zu erkennen ist. Durch die Vielzahl der kleinen Änderungen im Dokument wird dem Betrachter eine Differenzausgabe generiert, die schwer zu lesen ist und die Überprüfung der Änderungen erschwert. Sind nur wenige Änderungen vorgenommen worden, wie im zweiten Absatz der Abbildung 13 dargestellt, sind diese vergleichsweise leicht identifizierbar.

### Wortbasierter Vergleich

Diese Methodik liest einzelne Zeichen aus den beiden Dokumenten (siehe Abbildung 12) ein und und versucht diese zu Worten zusammenzusetzen, wodurch ein Vergleich auf Wortebene realisiert werden soll. Als Merkmal zur Identifikation eines Wortendes könnte das gebrauchsübliche Leerzeichen verwendet werden. Eine Wortbildung ist jedoch nicht in jedem Fall robust, da Dokumente beispielsweise mit Silbentrennung erstellt wurden. Zum Vergleich wird das erste zusammengesetzte Wort des ersten Dokumentes mit dem ersten zusammengesetzten Wort des zweiten Dokumentes verwendet und auf eine Änderung überprüft wird. Der Vergleich wird solange durchgeführt, bis alle Inhalte der Dokumente vollständig verarbeitet wurden.

Durch den Vergleich einzelner Worte ist es möglich, alle Änderungen in Dokumenten (siehe Abbildung 12) zu identifizieren, die auch im zeichenbasierten Vergleich erfasst werden. Sind Worte identisch, werden die nächsten zusammengesetzten Worte analysiert. Sollte eine Änderung lokalisiert werden, wird dies durch die entsprechenden Markierungen, für entfernte, hinzugefügte oder unveränderte Informationen, visualisiert. Diese Methodik wird solange durchgeführt, bis alle Inhalte der Dokumente auf die beschriebene Weise vollständig verarbeitet wurden.

Auch diese Vergleichsform ist sehr rechenaufwändig, insbesondere da vor einem Vergleich die Zusammensetzung von Zeichen zu Worten erfolgen muss. Allerdings reduziert sich die absolute Anzahl von relativ teuren Vergleichsoperationen durch die vorherige Komposition, da nicht jedes einzelne Zeichen verglichen werden muss, weshalb diese Berechnung der Differenzen effizienter ausfällt, als die zeichenbasierte Vergleichsform.

Durch die vorherige Zusammensetzung zu Worten, die verglichen werden, kann die Darstellung der Ergebnisse stärker strukturiert werden, sodass Änderungen in der Ergebnisansicht leichter identifiziert werden können (siehe Abbildung 14). Es müssen nicht mehr einzelne Zeichen in der Differenzausgabe lokalisiert werden, sondern die betroffenen Worte, in denen die Änderung vorliegt. Hierbei ist es unerheblich, wie viele Fehler in einem Wort durch den Verfasser korrigiert oder hinzugefügt wurden, da ein direkter Vergleich von vorheriger Version (rot) und nachfolgender Version (grün) eines Wortes visualisiert wird und vom Leser entschieden werden kann, ob diese Änderung akzeptabel erscheint.

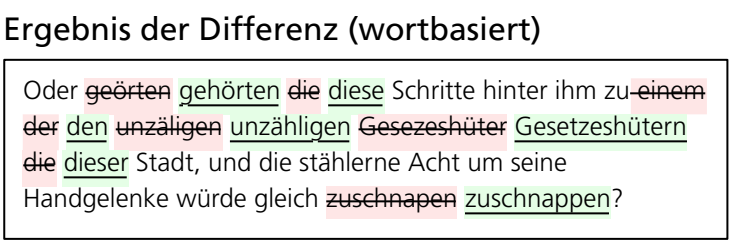



Abb. 14: Visualisierung der berechneten Differenz (wortbasiert) des zweiten Textabsatzes

### Zeilenbasierter Vergleich

Diese Methodik ließt einzelne Zeilen aus Dokumenten ein, sodass ein Vergleich auf Zeilenebene realisiert werden kann. In Abbildung 15 werden zwei Quellcode-Beispiele aus verschiedenen Programm-Versionen

gezeigt, die einen kleinen Auszug darstellen und Veränderungen beinhalten, die identifiziert werden sollen. Hierzu wird die erste Zeile des ersten Dokumentes mit der der ersten Zeile des zweiten Dokumentes verglichen und auf eine Änderung überprüft. Der Vergleich wird solange durchgeführt, bis alle Inhalte der beiden Dokumente vollständig verarbeitet wurden.

Programm Version 1

```
1  #include <stdio.h>
2  int main()
3  {
4  enum Days{Sun,Mon,Tue,Wed,Thu,Fri};
5
6  Days TheDay;
7  int j = 0;
8  printf("Please enter the day of the week (0 to 6)\n");
9  scanf("%d",&j);
10
11 if(Days(j) == Sun || Days(j) == Sat)
12 printf("Hurray it is the weekend\n");
13
14 return 0;
15 }
```

Programm Version 2

```
1  #include <stdio.h>
2  int main()
3  {
4  enum Days{Sun,Mon,Tue,Wed,Thu,Fri,Sat};
5
6  Days TheDay;
7  int j = 0;
8  printf("Please enter the day of the week (0 to 6)\n");
9  scanf("%d",&j);
10
11 if(Days(j) == Sun || Days(j) == Sat)
12 printf("Hurray it is the weekend\n");
13 else
14 printf("Curses still at work\n");
15
16 return 0;
17 }
```

Abb. 15: Quellcode-Beispiele aus einem versionierten C-Programm

Durch den Vergleich von einzelnen Zeilen ist es möglich, jede Änderung zu identifizieren, beispielsweise wenn Korrekturen im Dokument eingefügt oder einfach nur Inhalte entfernt wurden. Sofern Zeilen identisch sind, werden die nächsten Zeilen analysiert. Sollte eine Änderung lokalisiert werden, wird diese durch die entsprechende Markierungen, für entfernte, hinzugefügte oder unveränderte Informationen, sichtbar dargestellt. Hierbei besteht die Schwierigkeit darin zu ermitteln, über welchen Bereich der Dokumente sich Änderungen erstrecken, sodass ein lesbares Differenzergebnis visualisiert werden kann.

Diese Vergleichsform stellt eine effiziente Variante dar, da immer eine vollständige Zeile für den Vergleich eingelesen und auf eine Veränderung überprüft werden muss, sodass auch umfangreiche Dokumente in geringer Zeit überprüft werden können. Vorzugsweise werden bei dieser Methode Dokumente eingelesen, die eine stark zeilenbasierte Struktur aufweisen, wie etwa Code-Dokumente (siehe Abbildung 16). Diese werden auf Grund von Programmiermethodik und besserer Lesbarkeit vom Verfasser in zeilenbasierter Schreibweise erstellt, sodass diese Dokumente viele zumeist kurze Zeilen beinhalten.

Aufgrund der zeilenbasierten Auswertung kann das Ergebnis in der Ausgabe stark strukturiert visualisiert werden, sodass Änderungen in der Ergebnisansicht leichter lokalisiert werden können (siehe Abbildung 16). Durch die starke Strukturierung wird ein direkter Vergleich von vorheriger Version (rot) und nachfolgender Version (grün) einer Zeile visualisiert, sodass auch mehrere Korrekturen in einer Zeile leicht visuell erfassbar wären.

Ergebnis der Differenz (zeilenbasiert)

```
1   #include <stdio.h>
2   int main()
3   {
4   enum Days{Sun,Mon,Tue,Wed,Thu,Fri};
4   enum Days{Sun,Mon,Tue,Wed,Thu,Fri,Sat};
5
6   Days TheDay;
7   int j = 0;
8   printf("Please enter the day of the week (0 to 6)\n");
9   scanf("%d",&j);
10
11  if(Days(j) == Sun || Days(j) == Sat)
12  printf("Hurray it is the weekend\n");
13  else
14  printf("Curses still at work\n");
15
16  return 0;
17  }
```

Abb. 16: Visualisierung der berechneten Differenz (zeilenbasiert) der Quellcode-Versionen

Gerade bei Code-Dokumenten, die zeilenbasiert erstellt werden, bietet diese Art des Vergleichs eine effektive Möglichkeit, Änderungen von verschiedenen Versionen zu differenzieren. Sowohl kleine als auch große Inhaltliche Änderungen sind leicht zu identifizieren, da zumeist komplette Methoden-Bausteine in derartigen Dokumenten verändert werden, die im Normalfall zusammengefasst dargestellt werden, sodass die direkte visuelle Zuordnung möglich ist.

Bei Textdokumenten hingegen ist ein Vergleich auf Zeilenebene jedoch weniger effektiv, da das Ende einer Zeile nur durch entsprechende Systemzeichen, wie beispielsweise einem "Tastatur-Return" identifiziert werden kann. Dies hat zur Folge, dass eine Zeile sich bis zum Ende eines Absatzes im Dokument erstreckt.

Ergebnis der Differenz (zeilenbasiert)

~~Oder gehörten die Schritte hinter ihm zu einem der unzäligen Gesezeshüter der Stadt, und die stählerne Acht um seine Handgelenke würde gleich zuschnapen?~~
Oder gehörten diese Schritte hinter ihm zu den unzähligen Gesetzeshütern dieser Stadt, und die stählerne Acht um seine Handgelenke würde gleich zuschnappen?

Abb. 17: Visualisierung der berechneten zeilenbasierten Differenzausgabe

Die dazu generierte Differenzausgabe in Abbildung 17 zeigt zwar dass eine Änderung vorliegt, jedoch nicht an welcher Stelle in einem Absatz und wie viele Änderungen in einem Absatz enthalten sind, sodass diese aufwändig manuell gesucht werden müssten. Für Textdokumente wäre folglich der zeichenbasierte oder wortbasierte Vergleich vorzuziehen.

#### 5.1.2.5 Parser Framework für die Quellcode-Analyse

Damit Quellcode in seine Module zerlegt werden kann, wird ein Parser benötigt, der die Struktur fertiger Programme erkennt und diese einzeln ansprechen kann. Eine Möglichkeit besteht hier in der Implementierung einer Software auf Basis von regulären Ausdrücken (Regex), die beispielsweise eine Funktion auf Basis einer Folge von sprachspezifischen Wörtern erkennt und identifiziert, wie in Abbildung 18 zu sehen ist.

```
1 public void printName(String name) {
2       ...
3 }

Regex: ([\w]*) (static*) ([\w]+) ([\w]*)
```

Abb. 18: Funktion und Regex zur Identifikation

Dabei ist noch keine Logikleistung enthalten, welche die extrahierten Strings reservierten Wörtern zuordnet. Wenn jetzt Rückgabe- oder Eingabewerte wie Listen, generische Datentypen und andere Typen identifiziert werden sollen, wird es sehr schnell kompliziert und unübersichtlich, da hierfür jeweils ein regulärer Ausdruck formuliert werden muss.

Daher wird nach einer Möglichkeit gesucht, bereits bestehende Programme einzubinden, die eine Programmiersprache parsen können. Ein Parser stellt an dieser Stelle eine praktikable Lösungsmöglichkeit dar. Die Aufgabe eines Parsers besteht darin eine bestimmte Eingabe, wie z.B. XML-Dateien einzulesen und auf die Inhalte angemessen zu reagieren. Je besser entwickelt ein Parser ist, desto genauer erkennt er diese Inhalte.

Another Tool for Language Recognition (ANTLR) stellt in diesem Zusammenhang ein Framework für verschiedenste Aufgaben dar. ANTLR in der Version 4 ist ein frei verfügbares Java Programm und lässt sich einfach in die Gesamtstruktur einer Anwendungslandschaft integrieren. Es ermöglicht unter anderem anhand einer einfachen Grammatik eine Eingabe in eine Baumstruktur zu überführen und diese dann abzuarbeiten.

```
1 grammar Simple;
2 ID : WORD NUMBER;
3 WORD : [A-Z]+ ;
4 NUMBER : [0 -9]+ ;
```

Abb. 19: Einfache ANTLR Grammatik

ANTLR bietet für viele Programmiersprachen fertige Grammatiken an, wie beispielsweise C-Code, Java-Code und viele weitere, mit Hilfe derer die Eingabe der jeweiligen Sprache analysiert werden kann. Um

eine Vorstellung von solch einer Grammatik zu erhalten, wird in Abbildung 19 eine simple Grammatik gezeigt. Diese sucht in der Eingabe nach einer Identifikationsnummer (ID), die aus einem Wort in Großbuchstaben belieber Länge, gefolgt von einer Zahl belieber Länge, zusammengesetzt ist. Sofern eine derartige Struktur in der Eingabe enthalten ist, wird diese erkannt und in die Ergebnisliste aufgenommen.

### 5.1.2.6 Technologien für browserbasierte Applikationen

Folgendes Kapitel beschreibt die Techniken und Best Practices für die Realisierung von Applikationen für Internet-Browser. Hierbei handelt es sich um grundlegende Techniken zur prototypischen Entwicklung für das Comment System, die Traceability-Visualisierung sowie das User Interface als Benutzerschnittstellte.

#### Web-Entwicklung mit Bereitstellungstechnologien für HTML

Die sogenannte Hypertext Markup Language (HTML) wurde in Verbindung mit dem Hypertext Transfer Procotol (HTTP) im Jahr 1991 eingeführt und erfreut sich seitdem an einer fortwährenden Entwicklung. Zu Beginn bestand HTML aus einer wenig komplexen Struktur, sodass ein einfacher Umgang mit dieser statischen Programmiersprache problemlos erlernt werden konnte. HTML diente in diesen Anfängen lediglich dazu Text- oder Bildinhalte mit Hilfe von Auszeichnungselementen für die Ausgabe auf einer Web-Oberfläche strukturieren zu können. Aufgrund der vielfachen Anwendungsszenarien dieser Sprache war eine Weiterentwicklung zu einer Layout-Sprache unabdingbar. Diese Erweiterungen dienten dazu, eingebundene HTML-Elemente mit zusätzlichen Design- und Layout-Informationen zu erweitern, sodass eine flexiblere Visualisierung auf der Web-Oberfläche möglich wurde. In Folge dessen entstand aus der überschaubaren Anzahl an HTML-Elementen eine unübersichtliche Verkettung kombinierter Auszeichnungslemente. Damit dieser Zustand verbessert werden konnte, wurde die ergänzende Layout-Sprache Cascading Style Sheets (CSS) entwickelt. CSS wurde entwickelt, um diese zusätzlichen Informationen von den grundlegenden Strukturinformationen zu trennen. Durch diese konsequente Trennung wurde es möglich, dass HTML wieder für die Bedeutungsauszeichnung verwendet werden konnte, während CSS die Design-Informationen kapselt.[103]

Neben der reinen statischen Entwicklung einer Weboberfläche mit HTML und CSS, erhielt ebenfalls zu dieser Zeit die dynamische Programmiersprache JavaScript eine hohe Bedeutung für die Bereitstellung von Inhalten in einem Web Browser. Mittels JavaScript konnte die statische Struktur von HTML aufgebrochen werden und erlaubte es dem Entwickler, Inhalte für den Benutzer in der Web-Oberfläche dynamische, je nach eintretender Situation, zu verändern. Die Vereinigung dieser beiden Programmierparadigmen fand in den sogenannten Java Servlets statt, welche im Jahr 1997 bereitgestellt wurden. Der Vorteil dieser Java-Servlet-Technologie ist die dynamische Generierung von HTML-Seiten direkt auf der Server Seite. Der Kern dieser Technologie besteht aus der Möglichkeit, HTML-Elemente in Java-Code einzubinden, sodass damit HTML-Code erzeugt werden kann, wie im folgenden Beispiel dargestellt wird.

```
println("<html><body> Text </body> </html>");
```

103.vgl. (Marinschek·&·Kurz,·2010,·S.·1)

Dieses Beispiel stellt den Ausgabetext in HTML Struktur dar, obwohl eine Java-Methode aufgerufen wird. Für den Aufbau von komplexen HTML-Seiten ist diese Handlungsweise dennoch ineffizient und unübersichtlich, da die einzelnen HTML-Strukturen zur Ausgabe auf einer Web-Oberfläche in einer Programmierung von Java aufwendig integriert werden müssten.[104]

Parallel zu dem Ansatz der Java-Servlets wurde die Technologie der JavaServer Pages (JSP) entwickelt. Hierbei bestand der Fokus nicht auf den entwickelten Java-Methoden, sondern auf den HTML-Strukturelementen. Die Idee von JSP bestand darin, die HTML-Strukturelemente in sogenannten Scriptlets zu kapseln und einzubinden, die Aufrufe von Java-Methoden zur Bereitstellung von dynamischen HTML-Elementen darstellen. Somit konnte die Entwicklung von komplexen HTML-Seiten mit dynamischem Java Script-Code und CSS-Elementen erheblich vereinfacht werden.[105]

Darüber hinaus wurde auch die Technologie JavaServer Faces (JSF) entwickelt, welche die vielfältigen Ansätze zur Entwicklung in Web-Anwendungen mit Java bündeln sollte. JSF definiert hierzu ein komplexes Framework für die Schnittstellenentwicklung mit dem Benutzer in Java-Webanwendungen. Um eine konsistente Bündelung zu gewährleisten, war eine zentrale Standardisierung eines Web-Frameworks zwingend erforderlich.[106]

Seit 2004 ist JSF in der Version 1.0 von dem Java Community Process im Java Specification Request (JSR)-127 spezifiziert. Im Jahr 2006 folgte bereits Version 1.2, welche als Teil von Java EE 5 bereitgestellt wurde. Bis 2009 hat die Version 1.2 von Java EE maßgeblich zum Erfolg von JSF beigetragen und wurde daraufhin im Jahr 2009 mit dem Erscheinen von Java EE 6 in Version 2.0 abgelöst.[107]

Die unterschiedlichen Funktionalitäten von JSF tragen ebenfalls zu dem Erfolg dieses Web-Frameworks bei. Zum einen besteht die Möglichkeit, verschiedenste Bibliotheken mit zahlreichen Komponenten zu verwenden, zum anderen besteht sogar die Möglichkeit, dass die Basisbestandteile des Frameworks erweitert oder ausgetauscht werden.[108] Die wesentlichen Aufgaben der JSF-Technologie können hierbei in folgende Teilbereiche zusammengefasst werden und beziehen sich, sofern nicht anders erwähnt, auf die gleiche Literatur.[109]

- Komponenten

  Es wird ermöglicht, vollständige Web-Anwendungen aus einzelnen Komponenten zu entwickeln. Zudem können benutzerdefinierte Komponenten erstellt und wiederverwendet werden.

- Datentransfer

  Die Verwendung von JSF bietet die Realisierung eines einfachen Datenaustausches zwischen Benut-

104. vgl. (Marinschek & Kurz, 2010, S. 2)
105. vgl. (Marinschek & Kurz, 2010, S. 2)
106. vgl. (Marinschek & Kurz, 2010, S. 6)
107. vgl. (Marinschek & Kurz, 2010, S. 6)
108. vgl. (Marinschek & Kurz, 2010, S. 7)
109. vgl. (Marinschek & Kurz, 2010, S. 23)

zerschnittstelle und Anwendung. Weiterhin ist eine automatisierte Konvertierung und Validierung der Daten möglich.

- **Zustandsspeicherung**
  Der aktuelle Zustand einer Anwendung kann jederzeit automatisiert gespeichert werden, sodass sowohl Anwendungsdaten als auch einzelne Komponenten zusammengetragen werden können. Die Speicherung erfolgt hierbei auf der Server- oder Client-Seite.

- **Ereignisbehandlung**
  Mit der Bereitstellung von Methoden zur Ereignisbehandlung können generierte Ereignisse des Benutzers auf der Server-Seite behandelt werden. Dies wird durch eine Verknüpfung von Ereignisbehandlungsmethoden und Komponenten realisiert.

Des Weiteren implementiert JSF das sogenannte Model2 - MVC (Model View Controller), welches sich im Bereich der Webentwicklung implementiert hat. Der Unterschied zum allgemein bekannten Model-View-Controller-Muster besteht darin, dass bei Model2 die einzelnen Komponenten genau definiert werden (siehe Abbildung 20).[110]

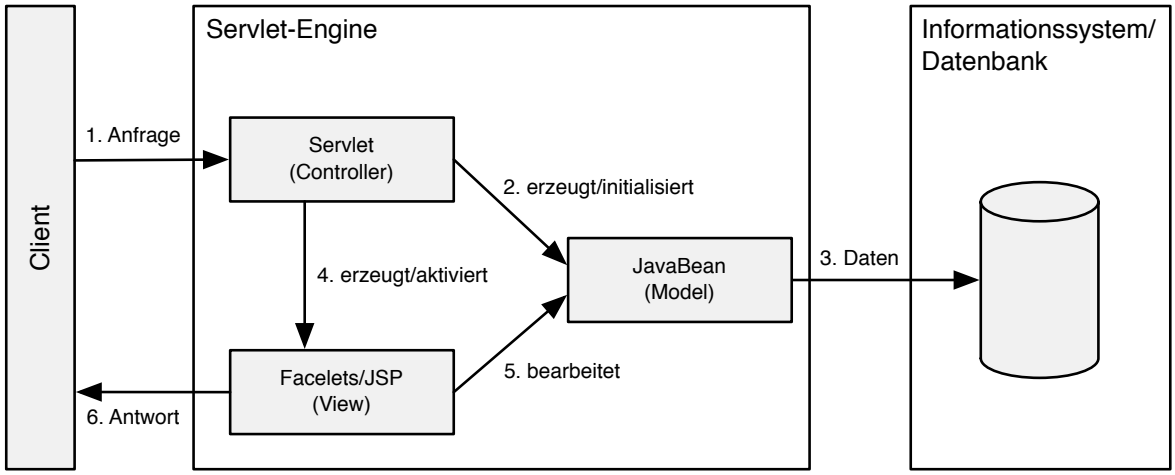



Abb. 20: JSF Model 2[111]

Das Model2-Muster besteht dabei aus drei zentralen Bausteinen:

- Servlet (Controller)
- JavaBean (Model)
- Facelets/JSP (View)

Ein Servlet ist eine Java-Klasse, die bei einem Webserver Anfragen entgegennimmt und diese beantwortet. Dabei kann der Inhalt dynamisch sein und muss nicht wie bei HTML statisch vorliegen. Das JavaBean stellt hierbei ein zur Laufzeit instanziiertes Java-Objekt dar. Dieses wird mit den benötigten Informationen

110. vgl. (Marinschek·&·Kurz,·2010,·S.·5)
111. vgl. (Marinschek·&·Kurz,·2010,·S.·5)

befüllt, wie zum Beispiel Benutzerinformationen nach einem Login. Der nächste Schritt erzeugt bzw. aktiviert definierte Facelets. Diese sind in JSF XHTML-Dateien mit speziellen Notationen abgelegt, die eine Verbindung auf die JavaBeans erzeugt. Durch diese Verbindung werden Inhalte dynamisch erzeugt und in die Facelets an definierten Stellen eingebunden. Bei Model2-Webanwendungen wird jede Anfrage über ein zentrales Servlet geleitet, welches diese bearbeitet und dem Nutzer die passenden Facelets zuordnet. Dieses zentrale Servlet wird deswegen auch Front-Controller-Servlet genannt und ist für den Benutzer nicht sichtbar.[112]

Facelets bestehen in JSF immer aus XHTML und sogenannten JSF-Tags. Diese Tags verweisen dabei auf POJO's (Plain old Java Objects) die durch zugefügte Annotationen in den Systemklassen automatisch als Beans vorgehalten werden (z.B. @ManagedBean vor Klassennamen). Zum Beispiel könnte es eine Klasse "benutzer.java" geben mit einer Variable "name". Soll dieser in einem Label ausgegeben werden, würde der XHTML Quellcode mit JSF-Tag wie folgt aussehen

```
<h:label value="#{benutzer.name}"></h:label>
```

Das JSF-Tag hat dabei den Aufbau #{Klassenname.Variable/Methode}. Dieses Tag kann an beliebiger Stelle im Facelet verwendet werden und führt entweder eine Methode in einer spezifischen Klasse aus oder gibt den Inhalt einer Variable auf der XHTML-Seite wieder.[113]

### Dynamische Webentwicklung mit Javascript

Javascript wurde für HTML-Entwickler bereitgestellt, welche mit Hilfe dieser Programmiersprache ihre Web-Seiten erheblich optimieren können. Der Javascript Quellcode kann sowohl in einer separaten Datei abgelegt werden, als auch direkt in der HTML-Datei. Moderne Web-Browser enthalten Interpreter-Software, mit deren Unterstützung der vorhandene Javascript Quellcode zur Laufzeit im Browser interpretiert wird. Mit Hinblick auf den Sicherheitsaspekt wird Javascript in einer abgeschlossenen Sandbox im Web-Browser ausgeführt. Eine Sandbox agiert als Sicherheitskäfig über die bereitgestellte Javascript Funktionalität, sodass der Javascript Quellcode um sicherheitskritische Funktionen beschnitten wird. Damit wird gewährleistet, dass keine beliebigen Daten / Funktionen ausgeführt, gelesen oder geschrieben werden können. Auf diese Weise wird unterbunden, dass Javascript-Entwickler auf Rechnern von Anwendern ungewollte Manipulationen durchführen können.[114]

Derzeit wird Javascript als Ergänzung zu HTML angesehen, aber nicht als alleinstehender Ersatz. Allerdings besteht die Möglichkeit eine Web-Seite ausschließlich in Javascript zu programmieren, jedoch funktionieren derartige Seiten nur, wenn der Anwender Javascript vollständig in seinem Browser aktiviert hat und dieser alle Javascript-Konstrukte interpretieren kann. Aus diesem Grund wird empfohlen Javascript

112. vgl. (Urbanek, 2010, S. 28ff.)
113. vgl. (Marinschek & Kurz, 2010, S. 5) & vgl. (Marinschek & Kurz, 2010, S. 21)
114. vgl. (SELFHTML E.V. (Hrsg.), 2014)

nur in solchem Umfang einzusetzen, dass eine Web-Seite auch mit deaktiviertem Javascript-Quellcode dem Benutzer korrekt angezeigt wird.[115]

### JavaEE

Java EE steht für "Java Platform, Enterprise Edition" und beschäftigt sich mit dem Bereich der Entwicklung von verteilter Software. Verteilte Software beschreibt Programme, die auf mehreren Servern laufen, von verschiedensten Rechnern (Clients) ausgeführt werden und mit anderen Programmen aus dem Unternehmensumfeld verknüpft sind. Insgesamt ist Java EE eine sogenannte Dachspezifikation mit vielen dazugehörigen einzelnen Spezifikationen, die die Standard Edition von Java in verschiedenen Bereichen der Anwendungsentwicklung erweitern bzw. ergänzen. Die durch diese Spezifikationen festgelegten Schnittstellen und Umgebungseigenschaften geben den Anwendungsentwicklern einen Rahmen für ihre Entwicklung. In diesem Rahmen werden zudem auch viele standardisierte Infrastrukturdienste definiert. Durch diesen Rahmen müssen Anwendungsentwickler nicht die ganze Laufzeitumgebung im Blick haben, sondern lediglich die eigene Software.[116]

Eine Java EE Anwendung kann grob in drei Teile aufgeteilt werden. Zum einen gibt es die Client-Schicht. Diese ist für die Präsentation von Daten zuständig. Die zweite Schicht beinhaltet die serverseitige Implementierung (Server-Schicht). Diese kann zum Beispiel Anfragen von einem Client entgegennehmen und diesem antworten. Die dazugehörigen Informationen entnimmt die Server-Schicht dann der Datenspeicherung, oder auch Datenschicht.[117]

Auf der Client-Schicht existieren für Clients zwei verschiedene Möglichkeiten zur Nutzung von Java EE. Die häufigere ist die Nutzung vorhandener Internet Browser wie beispielsweise Mozilla Firefox, Google Chrome oder andere. Durch diese Methode wird eine neue Entwicklung einer eigenen Implementierung vermieden. Bei der Nutzung von Browsern als Client wird die gesamte Präsentationslogik auf dem Server bereitgestellt, meist durch Web Container wie JSF. Der Zugriff auf die Geschäftslogik erfolgt dabei z.B. über Web Services. Die andere Möglichkeit stellt die Entwicklung eines eigenen Desktop-Clients dar, wodurch die Präsentationslogik auf den Rechnern selbst implementiert wird. Zusätzlich dazu können diese Clients direkt auf die Geschäftslogik zugreifen und benötigen keine Web Services.[118]

In der Server-Schicht wird die gesamte Geschäftslogik implementiert. Das bedeutet, dass hier Anfragen der Clients verarbeitet werden und auf diese eine Antwort zurückgegeben wird. Diese Logik kann durch Contexts and Dependency Injection implementiert werden, welche im Verlauf des Kapitels näher erläutert wird.

Die letzte Ebene ist die Datenschicht, auf der Informationen gespeichert und verarbeitet werden. Die Speicherung und Verarbeitung von Informationen kann hierbei auf verschiedene Arten erfolgen. Eine

115. vgl. (SELFHTML E.V. (Hrsg.), 2014)
116. vgl. (Weil, 2011, S. 11)
117. vgl. (Weil, 2011, S. 11)
118. vgl. (Weil, 2011, S. 11)

Möglichkeit stellt die Realisierung der Datenschicht in einer relationalen Datenbank dar, aber auch andere Ansätze, wie ein dateibasiertes System sind hier denkbar.[119]

### Enterprise Java Beans (EJB)

Die Enterprise Java Bean stellt eine grundlegende Komponenten einer JavaEE-Anwendung dar. Sie enthält die Geschäftslogik und kann mit anderen EJB's verknüpft werden, um komplexe Anwendungen zu erstellen.[120]

EJBs sind Java Klassen, deren Methoden über ein Java Servlet oder Java Server Faces aufgerufen und durch eine passende Annotation kenntlich gemacht werden. Dadurch kann über ein Webinterface mit den Daten und Methoden einer Bean interagiert werden. Session Beans werden in drei Kategorien (siehe Abbildung 21) unterteilt, durch die der Lebenszyklus einer einzelnen Bean definiert wird.


Abb. 21: Kategorien der Session Beans

Jede EJB muss per Annotation einer dieser Kategorien zugeordnet werden. Eine Stateless-Bean wird durch das Hinzufügen der Annotation @Stateless oberhalb der Java Klasse definiert, wie in Abbildung 22 zu sehen ist. Die übrigen EJB Annotationen, @Statefull und @Singleton, erfolgen analog. Stateless-Beans haben keinen Zustand und werden Clients beliebig zugeordnet. Solche Beans sind sinnvoll, um dem Client Geschäftslogik zur Verfügung zu stellen, welche beim Aufruf einen genau definierten Zustand aufweist. Dies können auch einfache statische Methoden sein, wie beispielsweise die Rückgabe eines Datums vom Server, die dem Client zur Verfügung gestellt werden.

```
1 @Stateless
2 public class DateConnector () {
3
4     public static Date getDate () {
5         return new Date ();
6     }
7 }
```

Abb. 22: Beispiel für eine Stateless-EJB

Bei einer Statefull-Bean erhält jeder Client eine EJB zugewiesen und behält diese auch über einen definierten Zeitraum, beispielsweise die Dauer einer Client-Sitzung. Ein beliebtes Beispiel, für die Verwendung dieser Art von Beans, stellt ein Warenkorb in einem Webshop dar. Dabei behält jeder Client seinen

119. vgl. (Weil, 2011, S. 11)

120. vgl. (Weil, 2011, S. 267)

eigenen Warenkorb und muss ihn nicht mit anderen Benutzern teilen und kann diesen über verschiedene Seiten des Webshops benutzen. Die Verwendung von Statefull-Beans ist notwendig, damit ein Benutzer nicht bei jedem Seitenaufruf eine komplett neue Sitzung zugewiesen bekommt und zuvor getätigte Arbeitsschritte verloren gehen.

Die Singleton-Bean existiert nur ein einziges Mal innerhalb einer Anwendung und wird jedem Clienten zur Verfügung gestellt. Dies kann beispielsweise die Erzeugung einer ID sein, die für alle Clients durchgeführt werden soll. Durch die Verwendung einer Singleton-Bean ist hier sichergestellt, dass nicht parallel die gleichen ID's vergeben werden können und dadurch Duplikate auftreten.

### Contexts and Dependency Injection (CDI)

Wie auch bei EJB wird bei CDI mit sogenannten CDI-Beans gearbeitet. CDI ist jedoch in der Lage eine CDI-Bean, wie auch eine EJB, in einer JavaEE Anwendung zu verwenden, sodass ohne komplizierte Aufrufe ein Objekt bereitgestellt werden kann. Dadurch wird es auf einfache Weise ermöglicht, Objekte in anderen Objekten aufzurufen, um Objekt-Netze aufzubauen und gleichzeitig die Abhängigkeiten zwischen einzelnen Objekten aufzulösen.

Mit der Verwendung von CDI und EJB genügt es, eine kurze Annotation (@EJB oder @Inject) vor eine Klasse oder Methode (siehe Abbildung 23) einzufügen. Diese Annotation führt dazu, dass die JavaEE Laufzeitumgebung den Aufruf und die Verknüpfung zwischen den verschieden Klassen und Methoden übernimmt, sodass dem Entwickler die Arbeit erleichtert wird.[121]

```
1 @EJB
2 private Service service;

3 @Inject
4 private void setService(Service service){
5       this.service = service;
6 }
```

Abb. 23: Verwendung von EJB und CDI

Bei der Verwendung von CDI wird statt @EJB ein @Inject verwendet. Eine Methode, die per CDI eingebunden wird, darf sämtliche Zugriffsmodfikatoren (private, public, etc.) verwenden, Werte zurückliefern, sowie Parameter entgegennehmen, die mit dieser verknüpft sind. In bestimmten Fällen, kann es vorkommen, dass einige Werte eines Java-Objekts nach der Anbindung noch nicht gefüllt sind. Hierfür existiert die Annotation @Post-Construct, die auf eine Methode angewendet werden kann und exakt einmal aufgerufen wird, wenn das Objekt erzeugt wird. Über die Funktionalität können zusätzliche Informationen abgerufen werden, beispielsweise aus einer Datenbank und die Werte des Objektes ergänzt werden.

121. vgl. (Bien, 2009, S. 37)

### Rich Internet Application (RIA)

Das in SIWOB entwickelte Frontend zur Verwendung des Dokumentenverwaltungssystems ist eine sogenannte RIA. Eine solche Applikation wird mittels eines Internet Browsers (Internet Explorer, Firefox, etc.) ausgeführt und angezeigt. Eine RIA ist vereinfacht ausgedrückt nichts anderes als eine Internet Seite, welche gegenüber normalen Web-Seiten deutlich mehr Funktionalität bietet. Dabei werden Informationen und Programmlogik auf einem Server gespeichert, auf den eine RIA zugreift. Durch diesen Aufbau entstehen einige Vorteile gegenüber herkömmlichen Desktop Anwendungen. Ein mögliches Beispiel für RIAs sind die im "Google Drive" integrierten Anwendungen, wie beispielsweise Textverarbeitung oder Tabellenkalkulation.[122]

Ein Vorteil ist, dass eine Rich Internet Application von jedem Rechner aus erreichbar ist. Dadurch kann jeder Benutzer von jedem Rechner aus auf das System zugreifen und mit den Dokumenten arbeiten. Weiterhin kann auch von Tablets bzw. Smartphones von unterwegs auf diese Informationen zugegriffen werden. Es muss also nicht für jedes Betriebssystem ein eigenes Programm geschrieben werden.

Zusätzlich dazu befinden sich die Informationen nicht auf dem Rechner oder dem Tablet selbst. Deswegen müssen von den Clients keine Backups erstellt werden. Nur der Server muss die Daten sichern, um Verlust vorzubeugen.

Für die Entwicklung von RIAs existieren verschiedene Möglichkeiten, dazu zählen verschiedene Frameworks. Der Unterschied liegt darin, ob ein extra Plugin benötigt wird oder nicht. Die Umsetzung des Frontends von SIWOB wurde in diesem Fall durch Java EE realisiert. Dadurch wird eine Verknüpfung von in Java entwickelten Strukturen und XHTML (Extensible Hypertext Markup Language) zur Visualisierung auf dem Benutzer Client ermöglicht.

### Web Services

Unter dem Begriff Web Service wird im allgemeinen ein Dienst verstanden, der einem Client angeboten wird. Dieser Begriff bildet eine Untermenge des Gesamtkonzeptes einer SOA und kennzeichnet jene Architekturen, die Dienste für Clients anbieten.[123] Die ursprüngliche Idee hinter diesem Konzept ist es, ein Netz aus Diensten zu realisieren, das sich nicht nur durch Mensch-Maschine Kommunikation auszeichnet, sondern die Möglichkeit bietet, dass einzelne Dienste untereinander kommunizieren können.

Um die Interoperabilität zwischen verschiedenen Diensten zu ermöglichen muss ein gemeinsames Protokoll verwendet werden. Dies ist im Fall von Web Services häufig SOAP, welches unabhängig von der verwendeten Programmiersprache zur Implementierung eines Web Service ist. Dies ermöglicht es, dass heterogene Plattform-Netzwerke, die beispielsweise auf unterschiedlichen Programmiersprachen basieren, miteinander kommunizieren können.

---

122. vgl. (Petmecky, 2011, S. 5)

123. vgl. (Burbiel, 2007, S. 21ff.)

SOAP ist dabei an kein Übertragungsprotokoll gebunden, wobei häufig ein Austausch per HTTP auf Basis von TCP/IP genutzt wird.[124] Der Einsatz anderer Übertragungsprotokolle, wie beispielsweise SMTP oder UDP, ist aber auch denkbar. Der Vorteil von HTTP besteht in der Verwendung durch Webserver, die eine gute Verbreitungsmöglichkeit durch die technische Infrastruktur ermöglichen.

## 5.2 Software-Prüfung

An der Prüfung einer Software sind mehrere Parteien mit verschiedenen Sichtweisen beteiligt, wobei die beteiligten Personen auch Mitarbeiter desselben Unternehmens sein können. Zum Einen der Hersteller einer Software, der eine Prüfung in Auftrag gibt. Dieser ist an einem reibungslosen Prüfungsverfahren ohne negative zeitliche oder finanzielle Auswirkungen auf den Entwicklungsprozess seiner Software interessiert. Zum Anderen der Prüfer, der eine Software-Prüfung hinsichtlich inhaltlicher Korrektheit durchführt. Eine Prüfung, die mit einer minimalen Anzahl von Rückfragen und daraus resultierenden Unterbrechungen des Prüfprozesses stattfinden kann, verringert die Bearbeitungszeit, erhöht die Effizienz und spart somit Zeit und Kosten.

Für die Software-Prüfung können daraus zwei wesentliche Erfolgsfaktoren abgeleitet werden. Die Praxis zeigt, dass eine Software-Prüfung aus Sicht des Herstellers besser einschätzbar wird, wenn schon während der Entwicklungsarbeiten wesentliche Anforderungen der Prüfer berücksichtigt werden können. Dabei wären Hinweise sowohl zu inhaltlichen als auch formalen Aspekten nützlich. Prüfer hingegen könnten mit einer Spezifikation ihrer Anforderungen an die Dokumentation einer Software frühzeitig Impulse für das Entwicklungsverfahren geben.

### 5.2.1 Sichere Systeme und Fehlerquote

Eine vom Menschen neu programmierte Software ist grundsätzlich nicht fehlerfrei. Erste Maßnahmen zur Fehlerbehebung erfolgen bereits in Form von ersten Testverfahren durch die Programmierer, durch die zahlreiche Fehler gefunden werden. Allein während der Entwicklung werden durch diese Vorgehensweise ungefähr 30-50 Fehler je 1.000 Zeilen Programmcode, sogenannten Lines of Code (LOC), aufgespürt und vor einer Veröffentlichung eliminiert werden. Trotzt dieser Anstrengungen der Entwickler und ersten Tests, verbleiben im Schnitt 1-3 unentdeckte Fehler je 1.000 LOC in einer ausgelieferten Software, von denen schätzungsweise 10% schwerwiegende Restfehler darstellen. Im Rückschluss bedeutet es, das 0,1-0,3 Fehler pro 1.000 Zeilen Code zu einem Absturz des entwickelten Systems führen können. Wird diese grob abgeschätzte Quote der schwerwiegenden Fehler auf gängige Software oder hardwarenahe Software hochgerechnet, die ohne weiteres aus mehreren 10.000 Zeilen Code bestehen, erhöht sich die absolute Anzahl möglicher schwerwiegender Fehler. Durch zusätzliche kostenintensive Maßnahmen können diese schweren Fehler um den Faktor zehn weiter gesenkt werden. Trotzdem verbergen sich dann 0,01-0,03 schwerwiegende Fehler je 1.000 Zeilen Code in einer Software.[125]

---

124. vgl. (Finger·&·Zeppenfeld,·2009,·S.·56)

125. vgl. (Bommer,·Spindler,·&·Barr,·2008,·S.·6f.)

Ab wann gilt ein System somit als sicher? Vigenschow definiert für hardwarenahe Software, sogenannte Real Time Embedded System (RTES), dass diese als sicher gelten, sofern keine Fehler auftreten, solange das Gerät für seine Aufgaben benutzt wird. Weiterhin darf beim Auftreten eines Fehlers das laufende System nicht in einen gefährdenden Zustand geraten und beispielsweise eine Person schädigen. Der TÜV verschärft diese Definition dahingehend, dass im Falle eines auftretenden Fehlers nicht zu einer Gefährdung kommen darf. Weiterhin muss der Fehler durch das ausführende System in einer gewissen Toleranzzeit erkannt und angemessen darauf reagiert werden.[126]

Allerdings weisen RTES besondere Schwierigkeiten auf, die vor allem auf die Echtzeitausführung von Befehlen zurückzuführen sind. RTES sind unter anderem reaktive Systeme, die in einem nicht deterministischen Umfeld arbeiten. Dies kann beispielsweise der Fall sein, wenn Menschen in den Ablauf eines Arbeitsprozesses integriert sind. Hierbei sind die Reihenfolge und der Zeitpunkt von Ereignissen nicht vollständig planbar. Unter anderem gibt die Umwelt die Geschwindigkeit der Arbeit vor, an die sich ein RTES halten muss, da Menschen im Gegensatz zu Maschinen nicht immer mit gleichbleibender Leistungsfähigkeit arbeiten können. An diesem Punkt spielt die Nebenläufigkeit und Verteilung von Prozessen eine wichtige Rolle, da die Komplexität eines RTES dadurch weiter steigt.[127] Die Software-Prüfung steht vor einer besonders großen Herausforderung, da auftretende Fehler unmittelbar zu einer Gefährdung von Menschen führen können und das System wenig Zeit hat, um in einen sicheren Zustand für den Menschen überzugehen.

Durch die IEC werden Normen und Standards bereitgestellt, in denen definiert wird, ab welchem Zustand ein System als sicher gilt. Hier ist vor allem die IEC 61508 zu nennen ("Functional Safety"), die als umfassender Standard für sicherheitskritische Systeme dient und in diesem Bereich zumeist als Grundlage verwendet wird. Im besonderen wird in Teil 3 des Standards eine dokumentierte Vorgehensweise für durchzuführende Tests definiert.[128] Welche Vorgehensweise zum Testen angewendet werden soll, hängt vom Potential der Gefährdung durch das System ab und wird in SIL Klassen unterteilt. Es existieren nach der IEC 61508 vier SIL Abstufungen, wobei vier die höchste und eins die niedrigste Sicherheitsstufe darstellt. Je kritischer ein System eingestuft wird, desto aufwendiger fallen die Vorgehensweisen und die darin enthaltenen Testmethoden aus, um ein System zu überprüfen.[129]

Sofern beim Auftreten eines Fehlers in einer Software ein Mensch und dessen Gesundheit oder Leben gefährdet werden könnte, wird von sicherheitskritischen Systemen gesprochen. Die Einstufung eines zu entwickelnden Systems sollte deshalb bereits in einer führen Projektphase erfolgen, beispielsweise in der Phase in der die Machbarkeit ermittelt wird oder eine Risikoanalyse erfolgt. Grundsätzlich werden jedoch für sicherheitskritische Systeme die gleichen Testmethoden verwendet, wie sie auch für normale ungefährliche Systeme eingesetzt werden. Zusätzlich müssen weitere Aufgaben ausgeführt werden, um den

126. vgl. (Vigenschow, 2010, S. 144)
127. vgl. (Vigenschow, 2010, S. 145)
128. vgl. (Liggesmeyer, 2009, S. 386f.)
129. vgl. (Börcsök, 2011, S. 32ff.)

gestellten Anforderungen an ein sicherheitskritisches System gerecht zu werden. Hierunter fällt eine detaillierte Sicherheitsanalyse, die im Rahmen des Risikomanagements erfolgen muss. Weiterhin muss die Durchführung dieser Tests anhand definierter Modelle erfolgen (bsp.: V-Modell), aus denen vollständige Testdokumentation mit lückenloser Rückverfolgbarkeit zwischen Anforderungen und Testfällen resultieren, sodass ein Beleg über das Erreichen eines definierten Überdeckungsgrades (Maß über den Grad der Vollständigkeit einer Testtechnik) vorgelegt werden kann. Daneben muss belegt werden, dass alle aufgestellten Sicherheitsanforderungen vollständig erfüllt sind und dass vor allem Defekte bzw. Ausfälle mit Hilfe geeigneter Maßnahmen korrekt behandelt werden und keine Person zu Schaden kommt. Alle gesammelten Testdaten, ermittelten Ergebnisse und verwendeten Testumgebungen müssen abschließend langfristig aufbewahrt werden. Der weitgehende Unterschied beim Testen besteht hierbei in der Gründlichkeit der Durchführung der ausgewählten Testmethoden.[130]

### 5.2.2 Fehler-Entstehung, -Verstärkung und Kosten

Durch das Testen soll die Qualität einer Software gesteigert werden, wobei die Anforderungen an die Fehlerquote unterschiedlich streng ausfallen können. Vor allem wenn bei einem Fehler Personen gefährdet oder sogar verletzt werden können, erhöht sich die Erwartung an die Software. Aus den Projektphasen (siehe Abbildung 24), der Anforderungsformulierung, der Entwurfsphase und der Codierung resultieren die Fehler einer Software. Die ersten 10% der Fehler entstehen bei der Formulierung von Anforderungen, die beispielsweise aus falschen Annahmen über das gewünschte Verhalten oder Formulierungsschwächen resultieren. Der größte Anteil der Fehler (90%) entsteht jedoch in den Phasen des Entwurfs (40%) und der Phase der Codierung (50%).[131]

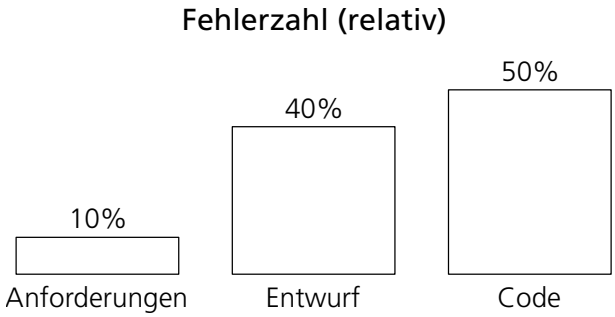



Abb. 24: Fehlerzahl in den verschiedenen Projektphasen[132]

Ohne Zweifel basiert ein Anteil der Fehler in der Entwurfsphase auf den fehlerhaften Anforderungen, da diese zunächst als korrekt und ausreichend detailliert angenommen werden. Die Umstand der Folgefehler zieht sich bis in die Phase der Codierung fort, durch den der Entwurf in eine echte Software umgesetzt wird. Neben den Folgefehlern können aber auch Defizite im Entwurf, wie etwa das Fehlen notwendiger Funktionalität, ein Grund für Fehler sein. In der Codierungsphase sind dies meist

130. vgl. (Bath·&·McKay,·2011,·S.·18f.)

131. vgl. (Frühauf·et·al.,·2007,·S.·30ff.)

132. in Anlehnung an (Liggesmeyer,·2009,·S.·33)

Implementierungsschwächen, wenn beispielsweise Prüfungen über falsche Grenzen erfolgen. Neben den genannten Beispielen existieren noch viele weitere Fehlerquellen, wie etwa Integrationsfehler oder auch Änderungsfehler, die aus verschiedenen Versionen stammen, auf die bei der Fehlersuche geachtet werden muss.[133]

Pressman formuliert den Umstand der Fehlerverstärkung in einem formalen Modell (siehe Abbildung 25), welches sich durch seine individuelle Betrachtungsweise auf jedes Testmodell anwenden lässt. Wird beispielsweise das V-Modell (siehe 5.2.4) als Basis zur Fehlerverstärkung verwendet, existieren insgesamt 8 Entwicklungsschritte, bei denen die Fehlerverstärkung berüchtigt werden muss. Der erste Schritt stellt einen Sonderfall dar, da hier keine Fehler von vorherigen Schritten weitergeben werden können, sondern nur neue Fehler generiert werden. In Abhängigkeit der verwendeten Testmethoden und dem Aufwand, mit dem diese durchgeführt werden, steigt die prozentuale Leistungsfähigkeit für die Fehlererkennung, wodurch die Fehleranzahl verringert wird. In den folgenden Schritten werden Fehler in die nächsten Entwicklungsschritte weitergegeben, wobei einige Fehler unentdeckt bleiben und somit unverändert weitergegeben werden. Andere Fehler führen hier zu einer Fehlerverstärkung, da durch eine Kopplung das Fehlerpotential vergrößert wird. Die Prozesskette wird schlussendlich bis zum letzten Entwicklungsschritt durchgeführt. Hierbei werden in den ersten Schritten, in denen alle konzeptionellen Arbeiten an einem Projekt durchgeführt werden, mehr Fehler erzeugt und weitergegeben, da hier die Grundlagen des Software-Produkts gelegt werden. Erst in den letzten Phasen, in den verschiedene Tests durchgeführt werden, findet ausschließlich die Fehlererkennung statt, so dass die Gesamtanzahl der übergegeben Fehler stetig sinkt. Mit Hilfe des Modells kann eine Abschätzung über die verbleibenden Fehler in einem Software-Produkt getroffen werden und ob die Testmaßnahmen im ausreichenden Umfang angesetzt wurden. Da dieses Modell auf Schätzungen basiert, ist es für die Genauigkeit der Prognose unerlässlich die tatsächlich erzielten Werte aus einen Projekt aufzuzeichnen und mit den Schätzungen zu vergleichen, damit eine Präzisierung für weitere Modellberechnungen möglich wird.[134]

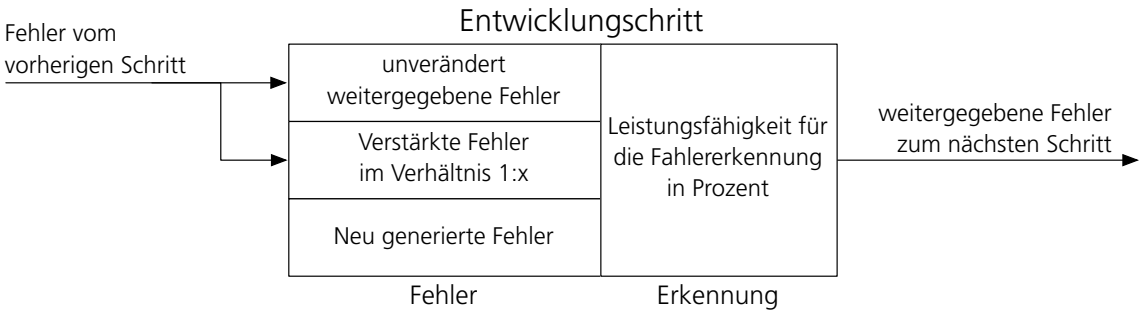



Abb. 25: Modell der Fehlerverstärkung in der Software-Entwicklung[135]

133. vgl. (Wratil·&·Kieviet,·2010,·S.·171f.)

134. vgl. (Pressman,·2005,·S.·752ff.)

135. in Anlehnung an (Pressman,·2005,·S.·752)

Demgegenüber stehen Kosten, die aufgewendet werden müssen, um die Fehler aus den verschiedenen Projektphasen zu beheben. Systematisches und planvolles Testen, unter Verwendung verschiedenster Testmethoden, verbessert grundsätzlich die Fehlererkennung. Außerdem reduzieren sich durch eine frühzeitige Erkennung mögliche resultierende Fehler, die bis zur Codierung mit "verschleppt" werden können. Eine detaillierte Betrachtung der anteiligen relativen Kosten wird in Abbildung 26 dargestellt, aus der deutlich wird, dass eine Verschleppung von Fehlern aus frühen Projektphasen zu exponentiell höheren Kosten für die Fehlerbeseitigung in späteren Projektphasen führen können.[136]

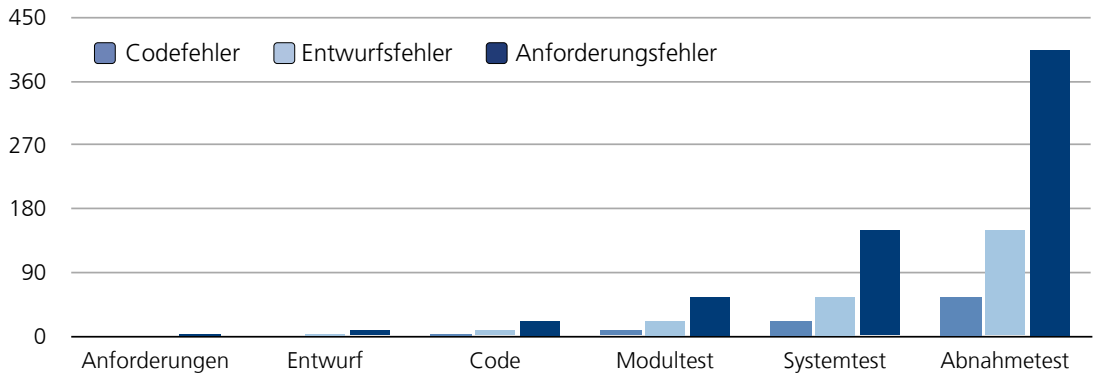



Abb. 26: Aufschlüsselung relativer Kosten zur Fehlerbeseitigung in den Projektphasen[137]

Hierbei kann in unterschiedliche Arten von Fehlern unterschieden werden, Codefehler, Entwurfsfehler und Anforderungsfehler, die in Bezug zur Phase der Umsetzung stehen. Durch die Verschleppung von Fehlern aus frühen Phasen der Projektentwicklung resultieren höhere Kosten zu deren Beseitigung. Liegen die relativen Kosten in den ersten Phasen, in denen Fehler entstehen (siehe Abbildung 24), im einfachen Verhältnis zu den Entwicklungskosten vor, so wachsen diese in den späteren Testphasen um ein Vielfaches an. Dies liegt unter anderen daran, dass die Durchführung von Tests Kosten aufwerfen, da Mitarbeiterzeit für die Erstellung und Durchführung der Tests eingeplant werden muss. Sollten Fehler erst in den letzten Projektphasen identifiziert werden, in denen die Software gegen die Anforderungen im Beisein des Auftraggebers getestet wird, fallen die relativen Kosten, derartige Fehler zu beseitigen, 40 - 400 mal höher aus, da hier im Regelfall einschneidende Veränderungen im Projekt zur Problemlösung erfolgen müssen. Bei sicherheitskritischen Systemen haben Hersteller oft keine Möglichkeit Fehler erst nach der Inbetriebnahme zu beheben. Hier muss die Fehlerfreiheit im Rahmen der Unbedenklichkeit gegenüber potentiell gefährdeten Personen eingehalten werden, sodass diese Fehler vor der Inbetriebnahme beseitigt werden müssen. In diesem Fall steht für den Hersteller auch kein Geld in Aussicht, da mit einem unfertigen Produkt kein Gewinn erwirtschaftet werden kann. In diesen Situationen sind bereits zahlrei-

136. vgl. (Frühauf·et·al.,·2007,·S.·19f.)

137. in Anlehnung an (Pressman,·2005,·S.·748) & (Frühauf·et·al.,·2007,·S.·20)

che Unternehmen gescheitert, bis hin zum Bankrott. Es sollte somit das Anliegen jedes Herstellers sein, gründlich und gewissenhaft zu arbeiten und das bereits ab der Phase der Anforderungen.[138]

### 5.2.3 Testmethoden

Liggesmeyer schlägt ein ausführliches Klassifikationsschema für Software-Prüfung vor und trifft eine grobe Unterscheidung zwischen dynamischen und statischen Testprinzipien. Diese sind im Detail in entsprechender Quelle nachzulesen. Dynamische Testtechniken weisen hierbei eine hohe praktische Bedeutung auf, wobei dies nicht für alle Techniken gilt. Besonders wichtig sind funktionsorientierte und einige kontrollflussorientierte Testtechniken. In der Klasse der diversifizierenden Techniken besitzt insbesondere der Regressionstest eine hohe praktische Relevanz.[139]

Durch das Testen sollen Fehler identifiziert und korrigiert werden, um die Qualität und Sicherheit zu erhöhen. Der Ablauf des Testens umfasst Planung, Design, Spezifikation, Durchführung und Auswertung.[140] Alle Vorgänge und Ergebnisse eines Tests müssen akribisch dokumentiert, analysiert und interpretiert werden, um versteckte Fehler zu entdecken, die anschließend korrigiert werden müssen.[141]

#### 5.2.3.1 Statische Methoden (Reviews)

Mängel in einer Software sind durch menschliche Fehler entstanden und können daher durch den Menschen wirksam erkannt werden.[142] Bei einem statischen Test wird die zu prüfende Software nicht ausgeführt, sondern manuell auf Fehler mit Hilfe verschiedener Methoden geprüft. Hierbei wird grundsätzlich keine Computersystem benötigt und es werden auch keine Testfälle für den Test ausgewählt, jedoch ist in dieser Testphase der Einsatz von Werkzeugen zur Unterstützung sinnvoll und erwünscht. Weiterhin kann keine Aussage über die Korrektheit oder Zuverlässigkeit der Software erfolgen.[143] Der Hauptteil einer statische Analyse entfällt dabei auf die Fehlersuche, neben der Analyse und Bewertung. Ein statischer Test kann solange wiederholt werden, bis formulierte Kriterien an die Software erfüllt sind.[144]

Der Nutzen statischer Analysen besteht vor allem in der frühzeitigen Fehlererkennung, die möglich ist, bevor ein ausführbares Programm vorliegt. Grundsätzlich gilt in dieser Phase der Software-Entwicklung, je früher Fehler identifiziert werden, desto geringer fallen die Kosten für die Beseitigung aus. Durch Analyse-Werkzeuge findet viel automatisierte Unterstützung statt, durch die Ergebnisse gewonnen und beispielsweise eine frühzeitige Risikoanalyse genauer durchgeführt werden kann. Hierbei besteht die Schwierigkeit in der Interpretation der Ergebnisse, die durch automatisierte Auswertungen erstellt werden und nicht immer einfach lesbar ausfallen. Darüber hinaus kann die Einhaltung von Codierungsstan-

138. vgl. (Liggesmeyer, 2009, S. 30ff.)
139. vgl. (Liggesmeyer, 2009, S. 37f.)
140. vgl. (Scholz, 2005, S. 192f.)
141. vgl. (Sneed et al., 2012, S. 11ff.)
142. vgl. (Frühauf et al., 2007, S. 23)
143. vgl. (Liggesmeyer, 2009, S. 43)
144. vgl. (Frühauf et al., 2007, S. 89)

dards überprüft werden, um die Wartungsfähigkeit einer entwickelten Software zu gewährleisten. Allerdings werden durch statische Analysen keine konkreten Fehler entdeckt, sondern nur verdächtige Stellen aufgespürt und müssen letztendlich durch einen Test belegt oder widerlegt werden.[145]

Vorteile:

- Es können alle Entwicklungsergebnisse (Code, Testdaten, Anleitungen, etc.) geprüft werden.
- Es werden nicht nur Schwachstellen der Software identifiziert, sondern auch die Schwächen der Entwickler, wodurch die Selbsteinschätzung realistischer wird.
- Durch Diskussionen werden Mängel identifiziert, die bei der Entwicklung im einzelnen nicht aufgefallen sind.
- Statische Methoden können bereits durchgeführt werden, bevor das Programm lauffähig ist, wodurch frühzeitig Probleme gelöst werden können.

Nachteile:

- Je mehr Querverweise im Programm vorliegen, desto unübersichtlicher und schwieriger wird es, die Prüfung durchzuführen.
- Fehler können gezielt auf Entwickler zurückgeführt werden und könnten diesen blockieren.
- Der Aufwand für die Vorbereitung und die Durchführung der Test ist nicht unerheblich und kann nicht nebenher erfolgen.

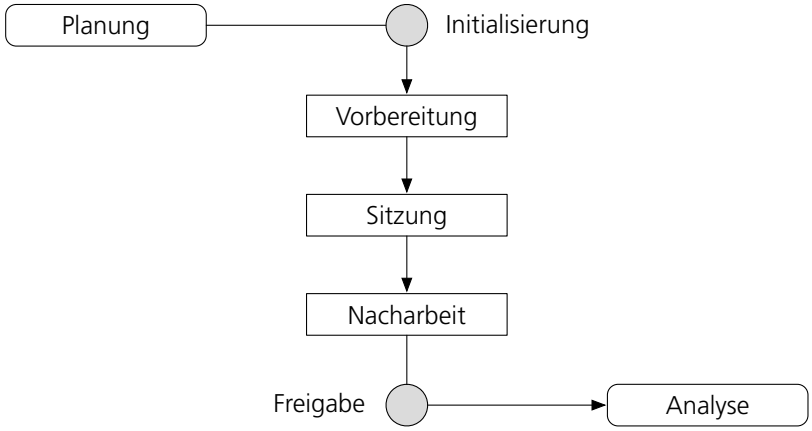



Abb. 27: Schematischer Ablauf eines statischen Tests[146]

Den Start eines statischen Testverfahrens (siehe Abbildung 27) bildet die Initialisierung, das Ende die Freigabe der erarbeiteten Ergebnisse. Zwischen der Initialisierung und der Freigabe werden bei dieser Testmethodik drei wesentliche Schritte sequentiell abgearbeitet, die Vorbereitung, die Sitzung und die Nacharbeit, wobei der Ablauf in mehreren Durchgängen erfolgen kann. Bevor jedoch ein statischer Test durchgeführt werden kann, bedarf es einer ausreichenden Planung, der zu testenden Aspekte und formulierten Fragen, die nach dem Test beantwortet werden müssen. Weiterhin müssen alle Unterlagen ge-

145. vgl. (Bath·&·McKay,·2011,·S.·116f.)

146. in Anlehnung an (Frühauf·et·al.,·2007,·S.·89)

sammelt und Einladungen an Personen versendet werden, die im Verlauf eines Reviews beteiligt sein müssen. Dies können beispielsweise Gutachter, Protokollanten, Programmierer der Software und weitere Personen sein. In der Phase der Initialisierung werden alle Dokumente und Informationen an die Testpersonen übergeben, die für den Test notwendig sind. Jeder Gutachter, sofern mehrere beteiligt sind, muss anschließend in der Phase der Vorbereitung in einem festgelegten Zeitraum das ihm zur Verfügung gestellte Material sichten und auf Fehler prüfen. Jeder Fehler, wie etwa Rechtschreibfehler, Abweichungen von Konventionen, Richtlinien, etc., werden vom Gutachter an entsprechender Stelle markiert oder mit Kommentaren notiert, sofern es sich um einen schwerwiegenden Fehler handelt, der Aufklärungsbedarf benötigt. Alle schwereren Fehler sind Bestandteil der anschließenden Sitzung und werden dort zwischen Gutachter und Hersteller diskutiert und in einem Bericht gebündelt notiert, der somit eine Liste der identifizierten Probleme darstellt. In der Phase der Nacharbeit hat der Hersteller die Möglichkeit, die erarbeitete Liste mit den Ergebnissen und die Markierungen zu korrigieren, sodass eine neue verbesserte Version vorliegt. An dieser Stelle kann ein neuer Prüfzyklus gestartet oder das geprüfte Projekt freigegeben werden, sofern die Gutachter mit dem Ergebnis der Vorbereitung und der Sitzung zufrieden sind bzw. keine weiteren Fehler vorliegen.[147]

### 5.2.3.2 Dynamische Methoden (Tests)

Der Ablauf des Testens beinhaltet verschiedene Phasen die nacheinander durchgeführt werden. In der Planungsphase werden Testfälle mit ausgewählten Bedingungen formuliert, in denen Eingabewerte und erwartete Ergebnisse definiert werden. Alle Testfälle müssen in der Phase der Durchführung anhand eines erstellten Ablaufplans ausgeführt werden. Hierbei ist es möglich, Testfälle zu koppeln, sofern Ergebnisse als Eingabewerte für andere Testfälle dienen. Die Ergebnisse der Tests werden dokumentiert und anschließend ausgewertet.[148]

"Die Vorstellung, einen Mechanismus auszuprobieren und aus den Beobachtungen zu folgern, ob er in Ordnung ist, entspricht der Intuition der meisten Menschen. Den Test kann man daher als 'natürliches' Prüfverfahren bezeichnen."[149]

Bei dynamischen Testverfahren wird die zu testende Software in einer realen Betriebsumgebung ausgeführt. Die zu testende Software wird hierbei mit zuvor definierten Eingaben und Werten ausgeführt und das Ergebnis gemäß der Spezifikation geprüft. Allerdings sind dynamische Testtechniken nur Stichprobenverfahren, weshalb kein Testverfahren die Korrektheit der getesteten Software beweisen kann.[150]

Der Nutzen dynamischer Tests besteht in der automatisierten Suche nach Fehlern in einer Software, die sonst nur mühsam oder unter Umständen gar nicht identifiziert würden. Es können hierbei auch Fehler identifiziert werden, die entfernt von der Ursache zu Problemen oder Abweichungen führen. Gleichzeitig

147. vgl. (Frühauf·et·al.,·2007,·S.·89ff.)
148. vgl. (Scholz,·2005,·S.·190ff.)
149. vgl. (Frühauf·et·al.,·2007,·S.·22)
150. vgl. (Liggesmeyer,·2009,·S.·39)

sind dynamische Tests sehr kosteneffektiv. Trotz möglicher hoher Anschaffungskosten decken die durch dynamische Tests automatisch gefundenen Fehler zumeist die Kosten, die entstünden, wenn diese erst in späteren Phasen identifiziert würden, wie etwa in der Produktion. Diese Testmethoden können bereits im Hintergrund während der Erstellungsphase ausgeführt werden und erhöhen dadurch unter anderem das Vertrauen in die erstellte Software. Fehlerhafter oder zu verbessernder Code kann bereits in der Entwicklungsphase identifiziert bzw. für detaillierte spätere Testmethoden vorgemerkt werden. Damit dynamische Testmethoden effizient durchgeführt werden können, müssen diese durch entsprechende Methoden und Implementierungen mit dem Quellcode verbunden werden. Gerade bei zeitkritischen Systemen (Echtzeitsysteme) kann es dadurch zu Verfälschungen der Ergebnisse kommen, da die Testmethoden die Performance des zu testenden Systems beeinträchtigen können.[151]

Vorteile:

- Ein dynamischer Test ist beliebig reproduzierbar, sofern das geprüfte Programm und das System nach deterministischen Regeln arbeiten.
- Nach Planung und Vorbereitung, können dynamische Test mit vertretbarem Aufwand durchgeführt und vor allem wiederholt werden.
- Bei einem Test wird die Zielumgebung, bestehend aus Hardware und Software, mit geprüft.
- Ein dynamischer Test ist eine Simulation der Praxis, wodurch das Systemverhalten für die Testperson sichtbar wird.

Nachteile:

- Beim Testen werden nicht alle möglichen Zustände geprüft, wodurch die Aussagekraft der Testergebnisse überschätzt werden kann.
- Es können nicht alle Anwendungssituationen durch einen Test nachbildet werden, insbesondere diejenigen, die nicht vorhersehbar sind.
- Ein Test zeigt lediglich, dass ein Fehler existiert, jedoch nicht die Fehlerursache.

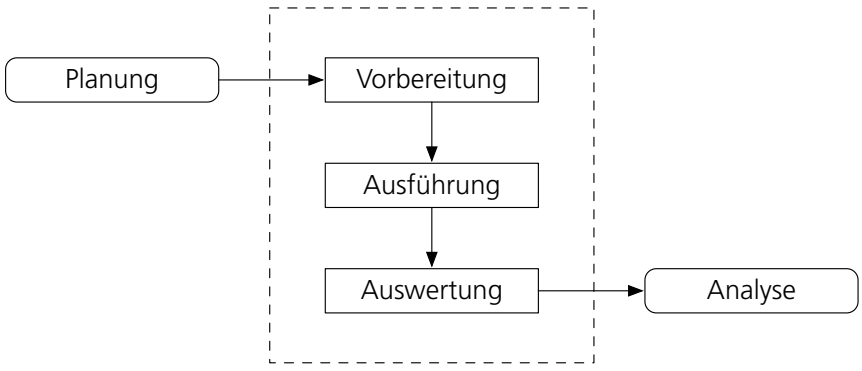



Abb. 28: Schematischer Ablauf eines dynamischen Tests[152]

151. vgl. (Bath·&·McKay,·2011,·S.·128ff.)

152. in Anlehnung an (Frühauf·et·al.,·2007,·S.·37)

Ein dynamischer Test gliedert sich grundsätzlich in drei Schritte auf (siehe Abbildung 28), die Vorbereitung, Ausführung und Auswertung, die in dieser Reihenfolge sequentiell abgearbeitet werden. Bevor ein Testzyklus über eine Software erfolgen kann, muss eine genaue Planung aufgestellt werden. Hierbei wird definiert, was das Ziel des Tests sein soll, welche Personen daran beteiligt sind und vor allem in welchen Umfang getestet wird. Weiterhin muss formuliert werden, wie die gesetzten Ziele erreicht werden sollen und welche Ergebnisse nach dem Test zu erwarten sind. Als erstes werden in der Testvorbereitung die grundsätzlich durchzuführenden Testfälle inklusive Reihenfolge erstellt und in einer Testvorschrift formuliert sowie die notwendige Hardware bestimmt und für den Test bereitgestellt. Im Anschluss daran erfolgt die Ausführung der spezifischen Testfälle. Hierbei führt ein Tester die Anweisungen gemäß der formulierten Testvorschrift aus und notiert die Ergebnisse detailliert in einem Testprotokoll. Anschließend werden Ist-Ergebnisse des Test mit den erwarteten Soll-Ergebnissen verglichen und mögliche Abweichungen zwischen Ist und Soll als potentielle Fehler in einem Testbericht notiert. Hierbei ist es wichtig, identifizierte Fehler unmittelbar zu protokollieren, damit diese nicht im Verlauf des Testvorgangs vergessen werden und später Schaden anrichten können. Alle Informationen aus der Durchführung und der Dokumentation der Ergebnisse werden im Testbericht gesammelt und zur genauen Analyse vorgelegt. Es wird geprüft, welche Fehler entstanden sind und wie diese behoben werden können, um daran anschließend weitere Testzyklen durchzuführen, bis die Ergebnisse den Erwartungen entsprechen.[153]

Dynamische Testmethoden können in drei grobe Kategorien unterteilt werden, strukturorientiert (White-Box) oder funktionsorientiert (Black-Box), in die jeweils weitere Testprinzipien eingeordnet werden können. Weiterhin existieren Grey-Box Testmethoden, durch die White-Box und Black-Box Tests kombiniert werden sollen, um dadurch die Vorteile beider Methoden auszunutzen. Dies wird erreicht, indem für den Entwurf der Testfälle Teile der Implementierung mit der Spezifikation bereits während der Entwicklung verbunden werden. Beispielsweise kann dadurch ein Integrationstest über die Schnittstellen der verschiedenen entwickelten Module oder Klassen einer Software erfolgen.[154]

#### 5.2.3.2.1 Strukturorientiert

Bei strukturorientieren Testmethoden, den sogenannten White-Box- oder Glass-Box-Tests, dient die Kenntnis über die innere Struktur der erstellten oder in Entwicklung befindlichen Software als Grundlage zur Erzeugung der Testfälle. Die Auswahl der Testfälle richtet sich nach den möglichen Abläufen, durch die bei der Ausführung der Software iteriert werden kann. In den Testfällen werden Informationen definiert, durch die festgelegt wird, welche Entscheidungen bzw. Abläufe durch den Test geprüft werden. Eine Ausführung gleicher Testfälle mit den definierten Informationen führen durch diese Testmethoden, beispielsweise in einer Verzweigung, zur selben Entscheidung. Eine Reproduktion bzw. kontinuierliches Wiederholen eines Tests inkl. der daraus resultierenden Ergebnisse zur Überprüfung stellt ein effizientes

153. vgl. (Frühauf et al., 2007, S. 38ff.)

154. vgl. (Vigenschow, 2010, S. 26f.)

Mittel während und nach der Software-Entwicklung dar. Das zentrale Element für die Auswahl von Testfällen beim Glass-Box Testmethoden stellt der Ablaufpraph der Software dar.[155]

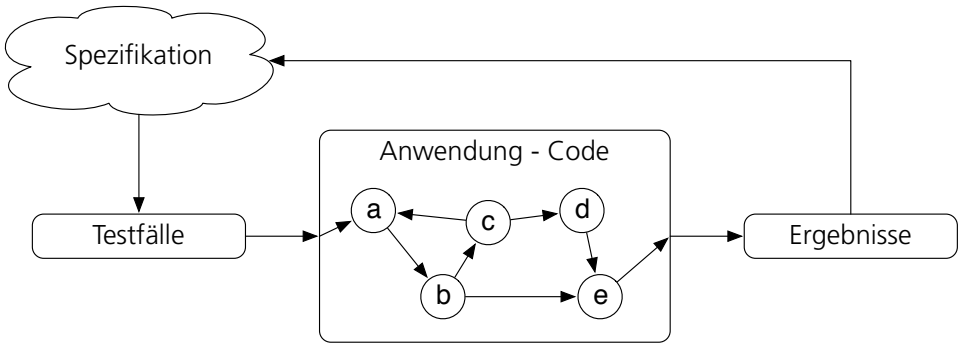



Abb. 29: Funktionsweise eines White-Box Test

Für White-Box Testmethoden (siehe Abbildung 29) werden ablauforientierte Testfälle auf Basis der Programmlogik und der Spezifikation definiert, mit denen die interne Struktur des Systems untersucht werden kann. Ein derartiger Testfall bezieht sich dabei auf einen ausgesuchten Bereich des Systems, wobei der Testentwurf auf Basis der Implementierung erfolgt und bei der Erstellung der Tests der Quelltext vorliegt. Dadurch kann die innere Struktur, wie etwa Anweisungen, Bedingungen, Pfade, etc., des zu testenden Objektes untersucht und das Verhalten bewertet werden.[156]

Mit Hilfe von White-Box Testmethoden und der Kenntnis über die innerer Struktur einer Software durch den vorliegenden Quellcode wird diese Struktur getestet und das Verhalten bewertet. Durch diese Tests soll die Struktur möglichst vollständig geprüft werden, wobei der Überdeckungsgrade definiert, wie genau der Test ausgeführt wird. Der Überdeckungsgrad stellt hierbei das Verhältnis der ausgeführten Anweisungen in einem Test zur Gesamtzahl aller möglichen Anweisungen dar. Die Qualität dieser Testmethoden steigt mit zunehmendem Überdeckungsgrad, der maximal 100% erreichen kann, wodurch gleichzeitig die Komplexität steigt, da mehr Testfälle berücksichtigt werden müssen.[157]

Die zu Grunde liegende Codestruktur beim White-Box Test stützt sich in häufigen Fällen auf die Testmethode automatisierter Unit Tests. Bei der Erstellung derartiger Unit-Tests ist besonders zu beachten, das vorliegende Gesamtmodell in möglichst kleine Teilmodelle aufzuspalten, um die Anzahl möglicher zu testender Zustände in jedem Teilmodell auf ein überschaubares Maß zu reduzieren. Dadurch wird die Testkomplexität des Gesamtsystems auf kleinere, weniger komplexe Probleme heruntergebrochen. Für jedes erstellte Teilmodell müssen allerdings die Schnittstellen für die Inputs und Outputs eindeutig definiert werden. Anschließend muss über alle Teilmodelle ein vollständiger Unit Test definiert werden, der alle er-

155. vgl. (Frühauf·et·al.,·2007,·S.·57ff.)

156. vgl. (Scholz,·2005,·S.·193)

157. vgl. (Vigenschow,·2010,·S.·26f.)

stellten Teile berücksichtigt und im Test abdeckt, sodass dieser dann für jedes Teilmodell durchgeführt werden kann.[158]

Strukturorientierte Testmethoden liefern kein Vorgehen zur Erstellung von Testfällen, sondern müssen in irgendeiner Form die programmierte Struktur der Anwendung mit diesen Fällen im Testdurchlauf abdecken. Die Spezifikation liefert hier nur eine Grundlage über die erwarteten Ergebnisse, die aus einem Test resultieren sollten.[159]

Es wird bei strukturorientierten Testverfahren in vier grobe Kategorien unterschieden in die sich die Testmethoden einordnen lassen. Abbildung 30 visualisiert die Hierarchie dieser vier Kategorien und durch welche diese subsumiert werden. Beispielsweise inkludiert der Zweigüberdeckungstest (C1) die Testmethoden des Anweisungsüberdeckungstests (C0) und gilt damit im gleichen Zuge als zuverlässiger. Diese Subsumption lässt sich äquivalent auf die restlichen Überdeckungstests anwenden.[160]

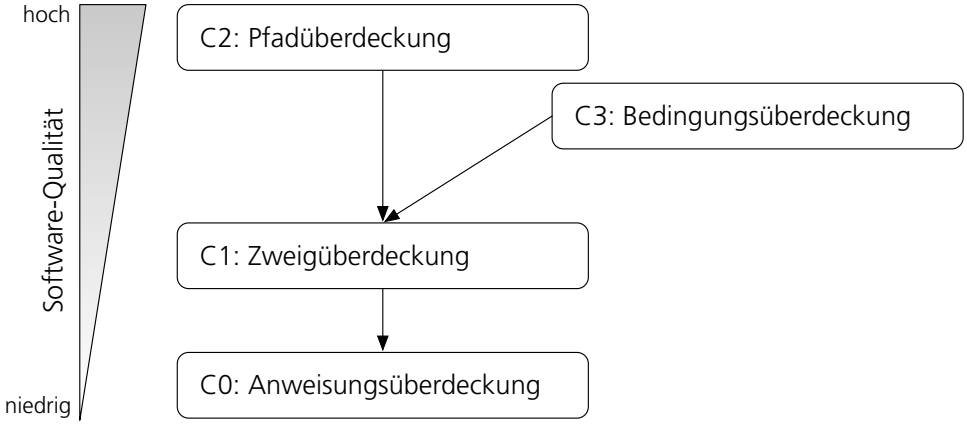



Abb. 30: Subsumption der strukturorientierten Testkategorien[161]

### Anweisungsüberdeckung

Der Test auf Anweisungsüberdeckung, oder auch C0 Test, stellt die einfachste Prüfmethode dar, bei der gewährleistet werden muss, dass alle Anweisungen bzw. Knoten (siehe Abbildung 29) durch den Test mindestens einmal ausgeführt werden. Wenn alle Anweisungen (Knoten) des Testobjekts überprüft wurden, ist ein Überdeckungsgrad von 100% erzielt und somit vollständig. Allerdings werden durch diesen Test zahlreiche Wege (Kanten), die beispielsweise durch Abfragen entstehen können nicht überprüft, sodass die Aussagekraft des Ergebnisses stark reduziert wird. Die Anweisungsüberdeckung ist lediglich eine oberflächliche, aber schnelle Prüfmethode, die gewährleistet, dass keine ungenutzten Anweisungen vorliegen.[162]

158. vgl. (Vigenschow, 2010, S. 283ff.)
159. vgl. (Liggesmeyer, 2009, S. 84f.)
160. vgl. (Wratil & Kieviet, 2010, S. 178)
161. in Anlehnung an (Bath & McKay, 2011, S. 96)
162. vgl. (Liggesmeyer, 2009, S. 85ff.)

### Zweigüberdeckung

Damit auch alle Kanten getestet werden können, muss ein Test auf Zweigüberdeckung, oder auch C1 Test, erfolgen. Hierbei muss durch den Test gesichert sein, dass alle möglichen Entscheidungen, Bedingungen, etc., also die Kanten (siehe Abbildung 29) im Testdurchlauf mindestens einmal abgearbeitet werden. Damit inkludiert diese Testmethode die Anweisungsüberdeckung (C0), da nach Abarbeitung aller Pfade zwangsweise auch alle Knoten mindestens einmal iteriert wurden. Sofern alle möglichen Pfade der Software durch einen Test geprüft wurden, ist ein Überdeckungsgrad von 100% erreicht. Mit Hilfe der Zweigüberdeckung wird Programm-Code identifiziert, der nicht durch einen Test abgedeckt wird, sodass dieser Anteil entweder im Programm optimiert oder ein neuer Testfall hierfür erstellt werden kann. Aber wie bei der Anweisungsüberdeckung liefern diese Testmethoden keine Aussage über die Fehlerfreiheit des Testobjektes, auch wenn ein Überdeckungsgrad von 100% erreicht wird. Verkettete oder komplexe Entscheidung sowie variable Belegungen können nicht berücksichtigt werden, da einige Konten bzw. Kanten nur ein einziges mal getestet werden.[163]

### Pfadüberdeckung

Mit Hilfe der Pfadüberdeckung, oder auch C2 Test, werden diese Sequenzen von Anweisungen (Knoten und Kanten) einer Anwendung geprüft. Eine Sequenz setzt sich aus einer beliebigen Anzahl von Anweisungen, Bedingungen, Entscheidungen, etc. zusammen, die gebildete werden können, um vom Start bis zum Ende eines Kontrollflusses zu gelangen. Das Ziel ist es hierbei, alle möglichen Pfade des Kontrollflusses (siehe Abbildung 29) durch diese Testmethode zu erfassen, damit nach Möglichkeit alle denkbaren Zustände überprüft werden. An diesem Punkt entsteht das Problem der Komplexität, die zumeist durch Schleifenanweisungen entsteht, wobei jede Iteration durch eine Schleifenanweisung einen weiteren Pfad in diesem Überdeckungstest darstellt. Meist beinhalten derartige Anweisungen iterative Beschränkungen oder frühzeitige Abbruchbedingungen. Unter Umständen existieren auch Anweisungen durch die ohne Wiederholungsanzahl iteriert werden muss. Ein Pfad gilt in diesem Zusammenhang als identisch, sofern eine Sequenz mit den gleichen Anweisungen über die gleichen Knoten und Kanten erfolgt. Da normale Software so gut wie nie ohne Schleifenanweisungen, Abfragen oder Verschachtelungen programmiert werden kann, entstehen dadurch millionenfach zu testende Pfade, die auf Basis des Kontrollflussgraphen erzeugt werden. Auf diese Weise können Pfade entstehen, die nie in einem Testverlauf abgearbeitet werden, da die logische Programmstruktur dies nicht erlaubt. Folglich kann auch nicht immer ein Überdeckungsgrad von 100% erzielt werden. Ungeachtet dessen weist die Testmethode der Pfadüberdeckung die beste Quote der Fehlererkennung auf, die durch sehr viel Rechenleistung erkauft werden muss und dadurch in den meisten Fällen nicht realisierbar ist.[164]

### Bedingungsüberdeckung

Der Bedingungsüberdeckungstest, oder auch C3 Test, berücksichtigt als einzige strukturorientierte Methode den Wahrheitsgehalt einer Belegung für eine Variable. Bereits in kleinen Programmen sind zahlrei-

163. vgl. (Liggesmeyer, 2009, S. 88ff.)

164. vgl. (Liggesmeyer, 2009, S. 136ff.)

che atomare Entscheidungen enthalten, durch die eine gewünschte Funktionalität erst erzielt werden kann. Entscheidungen oder auch Abfragen können hierbei in einfacher Form mit wenigen Zustände auftreten, aber auch komplexe zusammenhängende Prüfungen oder Bedingungen existieren meist in zahlreicher Form. Hierbei muss für einen erfolgreichen Test auf Bedingungsüberdeckung jede Entscheidung auf den Wahrheitsgehalt ***true*** und ***false*** getestet werden. Das bedeutet, dass gültige Belegungen für jede dieser Entscheidungen formuliert werden müssen, egal ob einfach oder komplex zusammengesetzt, damit jede vorkommende Variable einmal auf diese beiden Zustände getestet wird. Ein Test auf Bedingungsüberdeckung erreicht in diesem Sinne einen Überdeckungsgrad von 100%, sofern jede Variable mit beiden Wahrheitsgehalten erfolgreich getestet wurde. Problematisch wird dieser Test bei zusammengesetzten Bedingungen, da diese zur Ausführung des Mikroprozessors durch den Compiler in Atomare Bedingungen zerlegt und logisch ineinander verschachtelt ausgeführt werden. Hierbei wählt der Compiler eine möglichst optimale Reihenfolge, in der die Anweisungen geprüft werden, sodass Anweisungen gegebenenfalls nicht mit den geforderten Belegungen geprüft werden können, da ein Abbruch bereits auf viel höherer Ebene erfolgt. Ungeachtet dieser Tatsache können durch die Bedingungsüberdeckung einfache und auch komplexe logische Sachverhalte in einem Programm mit verhältnismäßig gutem Aufwand überprüft werden.[165]

#### 5.2.3.2.2 Funktionsorientiert

Bei funktionsorientierten Testmethoden dienen die funktionalen Anforderungen an das Programm (Spezifikation) als Grundlage, d.h. von der Interaktion des Programms mit seiner Umgebung und seiner Wirkung auf diese. Die Auswahl der Testfälle richtet sich nach den Eingaben, Ausgaben und ihrer funktionellen Verknüpfung, die in der Spezifikation formuliert sein müssen.[166] Grundsätzlich handelt es sich bei allen funktionsorientierten Testmethoden um Black-Box Tests. Dies gilt aber nicht im Umkehrschluss, da auch Black-Box Testmethoden existieren, die nicht als funktionsorientiert kategorisiert werden, wie etwa der Back to Back Test der im Kapitel 5.2.3.2.3 kurz beschrieben wird.[167]

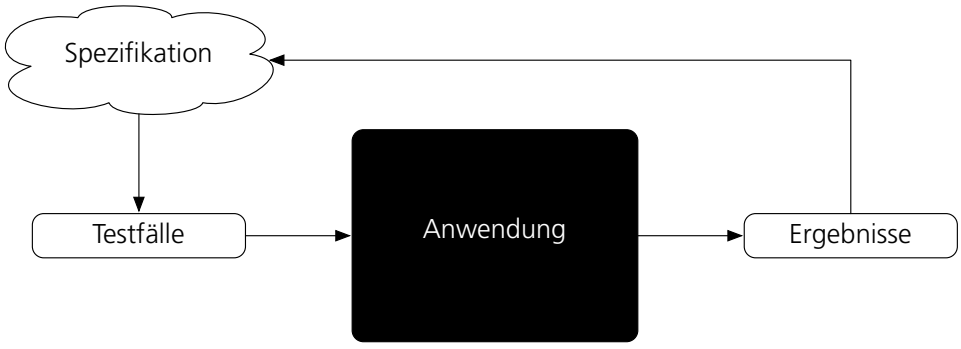



Abb. 31: Funktionsweise eines Black-Box Test

165. vgl. (Liggesmeyer, 2009, S. 93ff.)
166. vgl. (Frühauf et al., 2007, S. 45ff.)
167. vgl. (Liggesmeyer, 2009, S. 51)

Für einen Black-Box Test (siehe Abbildung 31) werden Testfälle gemäß der gestellten Spezifikation definiert, wobei die interne Struktur der Anwendung grundsätzlich uninteressant für den Prüfer ist. Es soll geprüft werden, ob die Anwendung im Rahmen der formulierten Spezifikation korrekt funktioniert.[168]

Für die Erzeugung und Auswahl von Testfällen aus einer Spezifikation werden meist drei Verfahren unterschieden, die Funktionsüberdeckung, Eingabeüberdeckung und Ausgabeüberdeckung, durch die der Aufbau und der Umfang von Testfällen definiert wird.[169]

### Funktions-, Eingabe- und Ausgabeüberdeckung

Das Ziel der Funktionsüberdeckung ist es mit Hilfe der ausgewählten Testfälle jede Funktion eines Programms mindestens einmal auszuführen und das Ergebnis zu überprüfen. Hierfür wird werden gemäß der vorliegenden Spezifikation alle enthaltenen Funktionen ermittelt und in einer Liste zusammengefasst. Gleichzeitig werden in zwei separaten Listen die möglichen Eingaben und erwarteten Ausgaben, die das Programm verarbeiten und liefern soll, anhand der Spezifikation ermittelt. Aus der Lister aller gesammelten Funktionen wird eine noch nicht durch einen Testfall abgedeckte Funktion selektiert und die dazu gehörenden Eingaben sowie erwarteten Ausgaben gesammelt. Anhand dieser Daten wird zu dieser Funktion ein passender Testfall formuliert und parallel die Funktionen in der Lister markiert, die auch durch diesen Testfall abgedeckt werden. Dieser Kreislauf des Selektierens einer Funktion, Ermittelns der zugehörigen Daten und Erstellens eines passenden Testfalls wird solange wiederholt, bis jede Funktion durch einen der formulierten Testfälle abgedeckt wird. Grundsätzlich verhält sich das Erstellen der Testfälle für die Eingabe- und Ausgabeüberdeckung gleich, nur dass bei diesen beiden Verfahren als Basis zur Erstellung der Testfälle die Listen mit den möglichen Eingaben und erwarteten Ausgaben als Grundlage zur Erstellung herangezogen werden. Im Ergebnis aller drei Methoden liegt eine Sammlung verschiedener Testfälle vor.[170]

Nach dem Entwurf der Testfälle auf Basis der Spezifikation oder den gestellten Anforderungen erfolgt der eigentliche Test, bei dem das Black-Box System die definierten Eingaben erhält und diese nacheinander ausführt. Anschließend werden die Testergebnisse bewertet, die bei entsprechenden Inputs entstehen, indem diese mit den erwarteten Ergebnissen aus der Spezifikation verglichen werden. Sofern keine Auffälligkeiten bzw. Abweichungen auftreten gilt ein Test als erfolgreich.[171]

### Äquivalenzklassen und Grenzwerte

Ein Problem bei der Erstellung von Testfällen besteht in der Auswahl repräsentativer Wertebereiche, sowohl für die Eingabedaten, als auch für die Ausgabedaten. Meist ist es nicht realisierbar alle möglichen Belegungen der Variablen in einem Programm zu überprüfen, da dies in einem unendlichen Berechnungsaufwand münden würde. Um dieses Problem zu lösen werden, die Wertebereiche einer Variable in

168. vgl. (Scholz, 2005, S. 192)
169. vgl. (Frühauf et al., 2007, S. 45f.)
170. vgl. (Frühauf et al., 2007, S. 46ff.)
171. vgl. (Vigenschow, 2010, S. 26f.)

Klassen aufgeteilt, die gemäß ihrer Belegung zu einem gleichen Ergebnis (***true oder false***) führen, wobei lediglich zwei Kategorien (***gültig und ungültig***) existieren. Die eine Klasse beinhaltet die Bereiche mit gültigen Eingabewerten, die zu einem korrekten Ergebnis führen, die andere Klasse beinhaltet die ungültigen Werte, die zu einem fehlerhaften Ergebnis führen.

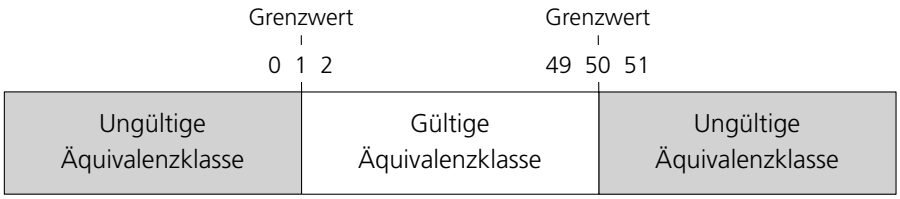



Abb. 32: Beispiel für den Wertebereich einer Variable[172]

Kann beispielsweise der Eingabebereich für eine Variable zwischen 1 und 50 (***1*** ≤ ***Wert*** ≤ ***50***) liegen (siehe Abbildung 32), so existiert für diesen Fall eine gültige und zwei ungültige Äquivalenzklassen. Die Eingabewerte von ***1*** ≤ ***Wert*** ≤ ***50*** liefern ein gültiges Ergebnis, die Eingabewerte ***Wert < 1*** und ***Werte > 50*** liefern ein ungültiges Ergebnis. Damit eine Eingabe- und Ausgabeüberdeckung erzielt wird, muss mindestens für jede vorhandene Äquivalenzklasse ein Testfall formuliert und geprüft werden. Äquivalenzklassen sind grundsätzlich einfach aus einer Spezifikation zu bilden, sofern diese umfangreich und gut formuliert wurde. Eine weitere Möglichkeit stellt die Betrachtung von Grenzwerten dar, durch die die Methode der Äquivalenzklassen erweitert werden kann, da zumeist Variablengrenzen Ursache zahlreicher Fehler in einem Programm sind, wie beispielsweise Abfragen mit < und ≤. Hierbei werden für diese kritischen Übergänge zusätzliche Testfälle formuliert, die diese Grenzen berücksichtigen und die ebenfalls gemäß der Äquivalenzklassen eingeordnet werden. Dadurch werden sensible Programmbereiche intensiver überprüft, wodurch ein höheres Maß an Sicherheit entsteht. Allerdings werden viele der möglichen Werte für eine Variable unter den Äquivalenzklassen pauschalisiert, wodurch gegebenenfalls Ausnahmefehler von seltenen gültigen Belegungen nicht erkannt werden können. Weiterhin können Abhängigkeiten und Wechselwirkungen zwischen Variablen schwer oder gar nicht identifiziert werden, da diese durch die Spezifikation nicht abgedeckt und somit formuliert werden können.[173]

#### 5.2.3.2.3 Diversifizierend

Mit diversifizierenden Testmethoden werden verschiedene Versionen einer Software gegeneinander getestet und die Ergebnisse zwischen den Versionen verglichen. Es entfällt somit die Bewertung der Korrektheit der Ergebnisse gegen die Spezifikation durch spätere Versionen einer Software, die häufig sehr aufwendig ausfallen kann. Die Versionen einer Software können durch parallele Entwicklungen basierend auf einer Spezifikation entstehen oder zeitlich nacheinander im Sinne von Software-Versionen realisiert worden sein.[174]

---

172. in Anlehnung an (Bath·&·McKay,·2011,·S.·39)
173. vgl. (Frühauf·et·al.,·2007,·S.·50ff.)
174. vgl. (Liggesmeyer,·2009,·S.·180)

### Back to Back-Test

Der Back to Back-Test (siehe Abbildung 33) testet mehrere Versionen einer Software gegeneinander, die auf einer gleichen Spezifikation basieren und durch unabhängige Personen parallel entwickelt wurden. Die unterschiedlichen Versionen erhalten gleiche Testdaten, wobei die Korrektheit durch den Vergleich der resultierenden Ergebnisse gezeigt werden soll.


Abb. 33: Schematischer Ablauf eines Back to Back Tests[175]

Sind alle Ergebnisse identisch, gilt die Software als korrekt. Sollten Abweichungen in den Ergebnissen aufgedeckt werden, müssen die Ursachen identifiziert und korrigiert werden. Für die Entwicklung diversitärer Programmversionen fallen höhere Entwicklungskosten an. Durch diese Art der Testmethode können sehr viele Testfälle in kurzer Zeit verarbeitet werden. Nachteilig an dieser Methode ist die theoretische Möglichkeit gleicher Fehler trotz unabhängiger Programmierung. In diesem Fall liefert ein Test identische Ergebnisse und die Software wird fälschlicherweise als korrekt angenommen, obwohl diese fehlerhaft ist.[176]

### Regressionstest

Mit Hilfe vom Regressionstest sollen Nebenwirkungen von Software-Modifikationen identifiziert werden, die sich negativ auf die Funktionalität auswirken könnten.

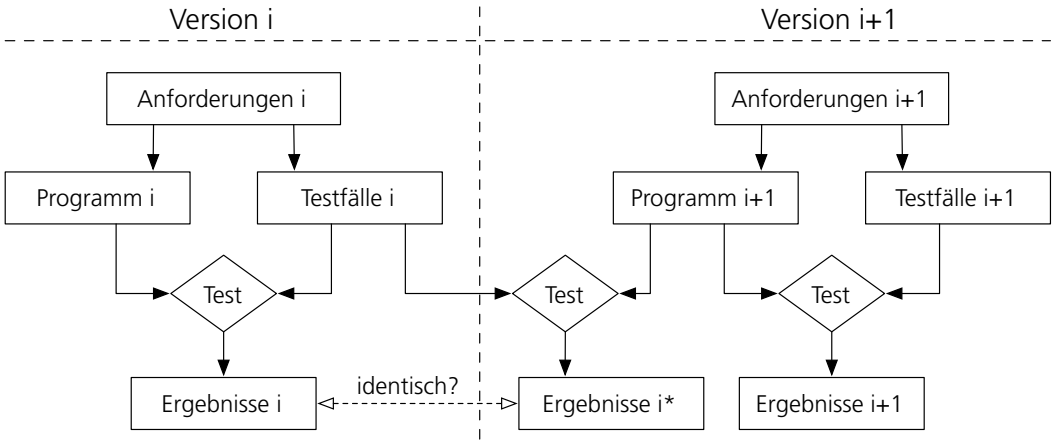



Abb. 34: Schematischer Aufbau eines Regressionstests[177]

175. in Anlehnung an (Liggesmeyer, 2009, S. 182)
176. vgl. (Liggesmeyer, 2009, S. 180ff.)
177. in Anlehnung an (Frühauf et al., 2007, S. 33)

Die Basis für einen Regressionstest (siehe Abbildung 34) stellt ein Test der ersten Version (i) einer Software dar, für den Testfälle (i) aus den Anforderungen (i) erstellt werden und mit Hilfe derer das Programm (i) getestet wird. Sofern die Ergebnisse (i) aus diesem Test keine Auffälligkeiten zeigen oder die Testpersonen keine Einwände äußern, gilt dieser Test als erfolgreich abgeschlossen. Neue Funktionen oder Verbesserungen führen zumeist dazu, dass neue Versionen (i+1) einer Software entstehen, die verständlicherweise auch überprüft werden müssen. Damit diese neue Version getestet werden kann, müssen für die Veränderungen aus den Anforderungen (i+1) neue Testfälle (i+1) formuliert und das neue Programm (i+1) mit den alten Testfällen (i) und den neuen Testfällen (i+1) überprüft werden. Anschließend werden die Ergebnisse des Tests verifiziert. Sofern die neuen Ergebnisse (i+1) bestätigt werden können und die Ergebnisse (i*) mit den alten Ergebnissen (i) der Vorgängerversion identisch sind, kann der Test als erfolgreich gewertet werden. Da dieser Test mit jeder neuen Version wiederholt werden muss, sollte die Durchführung der Testfälle soweit wie möglich automatisiert werden, da eine manuelle Prüfung nicht wirtschaftlich ist. Der Vorteil besteht hierbei darin, dass für jede Version lediglich die Veränderungen auf Basis der neuen Anforderungen in Testfällen formuliert werden müssen. Bestehende Funktionen einer Software werden dadurch in jeder Testphase überprüft, ob diese weiterhin wie vorgesehen arbeiten oder durch die Veränderungen fehlerhaft ausgeführt werden. Weiterhin lassen sich mit Hilfe von Automatisierungen neue Versionen schnell und effizient Testen.[178]

### 5.2.3.3 Formale Methoden

Durch statische oder dynamisches Testes kann die hundertprozentige Korrektheit einer Software im Rahmen der Spezifikation nicht mit Sicherheit nachgewiesen werden. Mit diesen Testmethoden wird die korrekte Funktionsweise einer Software überprüft, sie sind allerdings nicht vollständig und beinhalten ein Restrisiko. Für einen vollständigen Beweis der Korrektheit müssen formale Beweistechniken verwendet werden, wie etwa die formale Verifikation, die sehr aufwendig sind.[179]

Ein System gilt als korrekt bezüglich einer Eigenschaft, wenn die Funktionen des System den Spezifikationen entsprechen. Durch mathematische Verifikationsverfahren und -techniken kann die Korrektheit überprüft und gewährleistet werden. Es soll überprüft werden, ob das System mit den definierten Grenzen der Systemspezifikation zusammenpasst und diese einhalten kann. Durch die Validierung wird ein System auf die gestellten Anforderungen des Auftraggebers geprüft und ob das System diese erfüllt.[180]

Durch formale Verifikationsmethoden kann die Korrektheit eines Systems überprüft werden und ob das zu testende System die definierten Grenzen der Systemspezifikation erfüllen kann. Ein im Bereich der Verifikationsmethoden bekanntes Verfahren ist das Model Checking, das eine vollautomatische Verifikation einer Systembeschreibung (etwa eines Quelltextes) gegen eine vorliegende Spezifikation durchführen kann. Die Überführung der Spezifikation in eine Repräsentationsform aus temporallogischen Ausdrücken ist dabei Bedingung für einen automatischen Test. Mit Hilfe der temporallogischen Ausdrücke werden

178. vgl. (Liggesmeyer, 2009, S. 180ff.)
179. vgl. (Liggesmeyer, 2009, S. 439f.)
180. vgl. (Scholz, 2005, S. 188f.)

die Kriterien gemäß der Systemspezifikation definiert. Der Model Checker prüft hierbei sämtliche möglichen Systemzustände, die aufgrund unterschiedlicher Variablenbelegungen entstehen können und vergleicht jeden einzelnen Zustand mit der vorgegeben Spezifikation. Existiert kein Zustand, der der Spezifikation widerspricht, gilt die Systembeschreibung hinsichtlich der gegebenen Spezifikation als verifiziert. Wird ein System hinsichtlich seiner Spezifikation erfolgreich formal verifiziert, so ist die Korrektheit dieses Systems nachgewiesen. Der große Nachteil besteht in der Komplexität dieser Methode, die mit wachsender Zahl an Variablen mehr Rechenleistung und -zeit benötigt. Da für eine Variable alle theoretisch möglichen Belegungen getestet werden müssen und dies in Kombination mit allen Variablen, ist dieses Verfahren auch bei einfachen Systembeschreibungen mit einem unverhältnismäßig hohen Rechenaufwand verbunden, wodurch für den formalen Beweis auf Korrektheit der Software sehr viel Rechenzeit notwendig werden kann.[181]

### 5.2.4 Teststufen

Das V-Modell (siehe Abbildung 35) verknüpft das Vorgehen bei der Software-Entwicklung mit entsprechenden qualitätssichernden Methoden. Auf der linken Seite befinden sich die Stufen der Entwicklung, denen entsprechend auf der rechten Seite Testmethoden zugeordnet werden. In die einzelnen Teststufen der rechten Seite ordnen sich die verschiedenen Testmethoden ein, die je nach Komplexität und Anforderungen an das Produkt gewählt werden. Hierbei wird das Modell in Richtung des geschriebenen "V" abgearbeitet. Die einzelnen Komponenten des V-Modells lassen sich gemäß der Projektgroße weiter verfeinern, wobei in diesem Kapitel nur die rechte Seite des Modells betrachtet wird.[182]

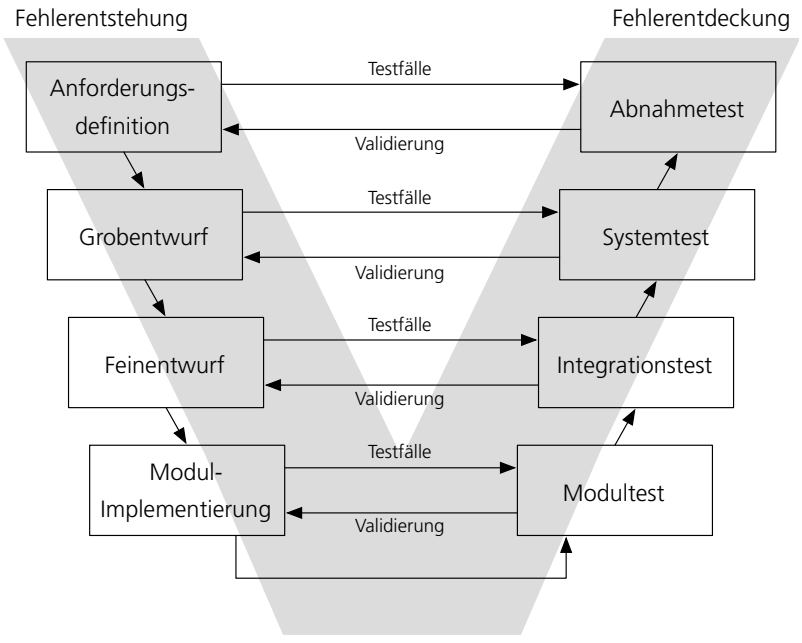



Abb. 35: V-Modell grobe Modulbetrachtung[183]

181. vgl. (Scholz, 2005, S. 196f.)

182. vgl. (Vigenschow, 2010, S. 173)

183. in Anlehnung an (Vigenschow, 2010, S. 173)

### Modultest

Im Rahmen des Modultests werden die kleinsten lauffähigen Einheiten einer Software getestet, die zumeist einzelne Klassen darstellen. Da diese häufig von anderen Klassen oder Funktionen abhängig sind, ist es hier notwendig geeignete Methoden bereitzustellen, die diese Abhängigkeiten ersetzen. Hierfür werden im Normalfall Test-Units (Testeinheiten), kleine Testeinheiten, zu den entsprechenden Klassen formuliert. Mit Hilfe dieser Testeinheiten kann die Funktionalität der Module einzeln nacheinander überprüft werden. Unit-Tests werden im Regelfall bereits während der Entwicklungsphase durch die Entwickler formuliert und gegen die entsprechenden Module getestet, durch die eine erste Kontrolle über die Lauffähigkeit möglich wird und erste Nachweise auf Korrektheit vorliegen.[184]

### Integrationstest

Die nächste Stufe, der Integrationstest, bezieht sich auf das Zusammenwirken abhängiger einzelner Module, die bereits im Modultest überprüft wurden. Hierbei wird das System schrittweise aus den einzelnen Modulen aufgebaut und in regelmäßigen Abständen der Modulverbund mit entsprechenden Testverfahren auf Funktionsfähigkeit untersucht. Es existieren zwei Ansätze, die das Vorgehen und die Reihenfolge des Modultests definieren, die Bottom-Up- und die Top-Down-Methode. Bei der Bottom-Up Methode werden von der untersten Ebene ausgehend abhängige Module getestet, die zu einzelnen Programmeinheiten zusammengefasst werden können. Sofern diese Tests erfolgreich sind, werden Stufe für Stufe Programmeinheiten zu Teilsystemen, Teilsysteme zum Gesamtsystem verknüpft und jeweils überprüft. Der Vorteil besteht hier in der frühzeitigen Prüfung auf Funktionalität mit der Hardware, da hier häufig Probleme mit den Schnittstellen entstehen. Im Falle der Top-Down Methode wird das Gesamtsystem in einzelne Teilsystem aufgespalten und getestet, bis am Ende wieder die einzelnen Programmeinheiten vorliegen. Hierbei kann frühzeitig die Funktionalität des erwarteten Gesamtsystems überprüft werden, wodurch das Produkt und deren geplante Abläufe besser eingeschätzt werden können. Der Schwerpunkt dieser Tests liegt auf den Schnittstellen der jeweiligen Stufen und einzelnen Komponenten, wodurch Ergebnisse über komplette Abläufe vorliegen. [185]

### Systemtest

Über den Systemtest wird das fertige System gegen die kompletten Anforderungen getestet, sowohl den funktionalen, als auch den nicht funktionalen, in denen alle Leistungsinformationen definiert stehen. Auf diese Weise wird ein Nachweis (Verifikation) über das erstellte System geliefert, welcher belegt, dass die die Informationen gemäß der Anforderungen mit Hilfe der Programmfunktionen korrekt verarbeitet werden. Unter den Systemtest fallen auch Leistungs- oder Stresstests die gegebenenfalls gemäß der Anforderungen erfüllt werden müssen. Ein Systemtest erfolgt dabei in einer möglichst realistischen Umgebung, wobei das System mit definierten Testdaten versorgt wird, damit die resultierenden Testdaten möglichst nah am Produktsystem liegen. Vor allem Leistungs- und Stresstest sind hier wichtig, da diese am Testsystem ohne realen finanziellen Schaden durchgeführt werden können. Hier werden Fragen der

184. vgl. (Liggesmeyer, 2009, S. 371f.)

185. vgl. (Scholz, 2005, S. 74f.)

Sicherheit geklärt, da eine Simulation ausserhalb von Normbedingungen möglich wird, wie etwa Überlastung, bei der das System in einen sicheren Zustand übergehen sollte.[186]

### Abnahmetest

Der Abnahmetest ist der letzte Test der durchgeführt wird, bevor eine Software ausgeliefert wird. Durch diesen soll festgestellt werden, ob das erstellte System die gewünschten Aufgaben (Verifikation) des Kunden erfüllt, für die es in Auftrag gegeben wurde. Einerseits kann das Produkt im Beisein des Auftraggebers oder des Kunden getestet werden, sodass dieser jegliche Aktion beobachten und einschätzen kann, ob er mit den Ergebnissen zufrieden ist. Der Test kann hierbei bereits auf einem Produktivsystem des Kunden unter Verwendung von echten Daten erfolgen, wodurch ein besseres Verständnis im Sinne des Kunden erzielt wird. Eine weitere Möglichkeit ist die Abnahme des Produkts auf Basis der Testergebnisse aus den vorherigen Teststufen, insbesondere des Systemtests. Diese Art der Akzeptanzüberprüfung ist der letzte Bestandteil, nach dem ein Produkt im Akzeptanzfall als fertiggestellt gilt und somit die Übergabe sowie die Bezahlung erfolgen und die Garantiezeit beginnt.[187]

## 5.2.5 Software-Alterung

Die Entwicklung von Software erfolgt im Regelfall unter der Prämisse des fehlenden Verschleißes, da das fertige Software-Produkt als losgelöste Einheit behandelt wird, die nicht an ein Hardware-System gebunden ist. Diese Fehlannahme führt zu idealisierten Betrachtungen in der Planung und Erstellung von Software, vor allem aber auch in der Durchführung von Tests, da sich das Einsatzumfeld einer Software im stetigen Wandel befindet. Hierunter fällt beispielsweise der schlichte Verschleiß von Hardware, ohne die eine Software nicht ausgeführt werden kann. Aber auch die kontinuierliche Entwicklung einer Software, um diese durch zusätzliche Funktionen zu erweitern, stellen mögliche Veränderungen im zeitlichen Verlauf dar.[188]

In diesem Zusammenhang hat sich der Begriff Software-Alterung oder auch "Software Aging" gebildet, der beschreibt, dass Software über die Zeit des Einsatzes verschiedenen Einflüssen ausgesetzt ist, aus denen ein fehlerhaftes Verhalten resultieren kann. Durch alternde Hardware oder Weiterentwicklungen, die eine Software komplizierter gestalten, wird eine Alterung im Sinne des Software-Lebenszyklus hervorgerufen, die bereits in der Entwicklung berücksichtigt werden muss.[189]

Vor allem bei hardwarenaher Software (embedded Systems) spielt die Software-Alterung eine wichtige Rolle, da Verschleiß von Hardware zu Messabweichungen und infolgedessen zu verändertem Systemverhalten führen kann. Beispielsweise könnte eine steigende Fragmentierung des Speichers durch anfallendes Datenvolumen oder häufige Datenzugriffe zu längeren Schreib- und Lesezyklen führen, die nicht durch Tests berücksichtigt und simuliert wurden, womit durch das veränderte Zeitverhalten fehlerhafte

186. vgl. (Liggesmeyer, 2009, S. 376f.)
187. vgl. (Frühauf et al., 2007, S. 79)
188. vgl. (Engels, Goedicke, Goltz, Rausch, & Reussner, 2009, S. 393)
189. vgl. (Wratil & Kieviet, 2010, S. 189f.)

Zustände entstehen könnten. Es besteht vor allem die Problematik, mechanischen Verschleiß in Software berücksichtigen zu müssen. Dies kann in keinem Fall zu 100% gewährleistet werden, da sich Verschleiß nach äußeren Einflüssen richtet, die sich gleichwohl über den Zeitverlauf verändern können. Im schlimmsten Fall können derartige Fehler zu einer Gefährdung von Personen führen und müssen durch geeignete Methoden in den verschiedenen Testphasen berücksichtigt werden. Eine Möglichkeit besteht hierbei mit Hilfe von Eigendiagnose gegen die Alterung von Software vorzugehen, indem beispielsweise regelmäßige Tests auf sensible Hardware durchgeführt werden, um die Leistungsfähigkeit oder die Bereitschaft des Systems zu überprüfen. Wenn bei diesen regelmäßigen Tests Abweichungen identifiziert werden, sollte das technische Gerät im Idealfall in einen sicheren Zustand übergehen, wodurch eine Gefährdung von Personen ausgeschlossen werden kann. Eine andere Möglichkeit ist die dynamische Anpassung der Software an die diagnostizierten Werte, durch die ein weiterer Betrieb im Rahmen der sicheren Ausführung ermöglicht, jedoch die Geschwindigkeit reduziert wird. Weiterhin wäre es denkbar, in zyklischen Abständen Optimierungen am produktiv arbeitenden System durchzuführen, sofern die Auslastung dies zulässt oder die gemessenen Werte dies zwingend erfordern. So könnte eine periodisch durchgeführte Defragmentierung einer verlängerten Schreib- und Lesezeit entgegenwirken.[190]

Weitere Möglichkeiten gegen Software-Alterung vorzugehen bestehen im Refactoring oder dem Redesign von Software. Refactoring beschreibt in diesem Kontext die manuelle oder automatisierte Verbesserung des Programm-Codes (Restrukturierung), ohne dabei die Funktionsweise nach außen zu verändern. Das Ziel dieser Methode besteht darin, das Design einer Software so zu verändern, dass auf der einen Seite die Fehleranalyse und Wartung erleichtert und auf der anderen Seite die Möglichkeit der funktionalen Erweiterung vereinfacht wird. Mit Hilfe definierter Refactoring-Vorgaben können komplexe Programmstrukturen ohne Veränderung der Funktionsweise umgestaltet werden. Beispielsweise kann die Lesbarkeit verbessert werden, wodurch sich das Verständnis für die Funktionsweise des Codes vereinfacht. Weiterhin verstärkt das Refactoring die Modularität von Software und reduziert im gleichen Zuge Redundanzen, da Code durch die Modularisierung an vielen Stellen mehrfach verwendet werden kann und nicht viele Male implementiert vorliegt. Unterstützung in der Durchführung von Refactorings besteht in der Durchführung von Regressionstests (siehe Kapitel 5.2.3.2.3), mit denen belegt werden kann, dass die Funktionsweise des restrukturierten Programms die Gleiche ist, wie im Original und sich somit keine Fehler eingeschlichen haben. Die Konsequenz in der Entwicklung einer Software durch diese Methode ist eine kontinuierliche Qualitätsverbesserung, durch die nicht neu in den Entwicklungsablauf eingestiegen werden muss. Entgegen dem normalen Entwicklungsprozess von Software (siehe Kapitel 5.2.4), bei dem die Prozessphasen sequenziell durchlaufen werden, erlaubt das Refactoring jede Phase im Einzelnen immer wieder abzuarbeiten.[191]

Hingegen stellt das Redesign von Programm-Code eine komplette Überholung der Codebasis dar, mit dem Ziel eine neue Version zu erschaffen. Diese Methode kommt immer dann zum Einsatz, wenn das

190. vgl. (Wratil·&·Kieviet,·2010,·S.·189f.)

191. vgl. (Hoffmann,·2013,·S.·396ff.)

Altsystem starr und unflexibel geworden ist und somit keine Voraussetzungen für Verbesserungen oder Erweiterungen bietet. Häufig findet aber ein Redesign auch schon in der ersten Entwicklung statt, beispielsweise wenn gravierende Mängel in den höheren Testphasen festgestellt werden (Systemtest oder Abnahmetest). Das grundlegende Ziel für ein Redesign ist in allen Fällen immer dasselbe. Vor allen die Entledigung von ausgedienten oder sogar fehlerhaften Code-Altlasten stehen hier im Vordergrund, wodurch Code effizienter und weniger fehleranfällig ausfallen soll. Hierbei fließen maßgeblich die Erfahrungen der vorherigen Entwicklungen sowie der durchgeführten Tests ein. Aber auch das Nachreichen fehlender Funktionen oder die Integration neuer Prüfroutinen zur weiteren Absicherung und Stärkung der Zuverlässigkeit stellen wichtige Aspekte dar. Im Ergebnis soll durch ein Redesign eine neue Version vorliegen, die den neuen Ansprüchen mehr gerecht wird.[192]

## 5.2.6 Workflow für den Software Test

Viele Vorgehensweisen zum Testen von Software orientieren sich in den Grundzügen am V-Modell (siehe Abbildung 35) und setzten die Reihenfolge, sowie die Ausdehnung der Testmethoden mehr oder weniger extensiv um, sodass sich viele Abläufe mit diesem Modell decken. In Abbildung 36 wird dieser Sachverhalt grob skizziert. Beispielsweise können durch den Einsatz verschiedener Methoden auf der Ebene von Programmeinheiten (Modul- oder Komponententests) nur logische Programmfehler aus der Kodierungsphase identifiziert werden, jedoch keine Fehler die auf der Ebene der Anforderungen entstanden sind.[193]

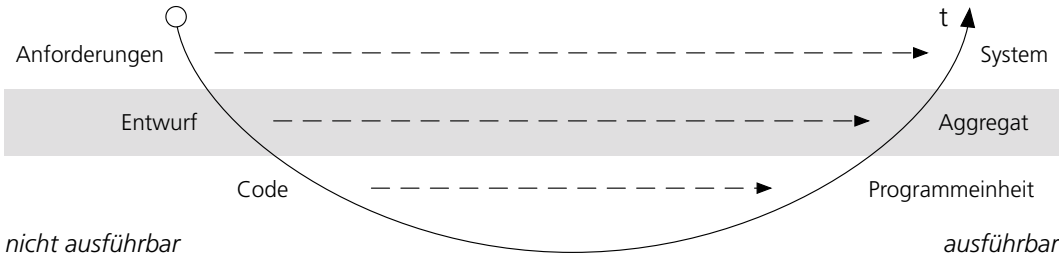



Abb. 36: Modellhafte Darstellung der Abstraktionsebenen in einem Software-Projekt[194]

### 5.2.6.1 Bottom-Up Methode

Für die Prüfung von Software existieren zwei Konzepte die gegensätzlich zueinander verfahren, dem Top-Down Konzept auf der einen und dem Bottom-Up Konzept auf der anderen Seite, aus denen mögliche Workflows abgeleitet werden können. Grundsätzlich ist hierbei zu beachten, dass Fehler nur auf der Abstraktionsebene identifiziert werden können, auf denen diese auch entstehen. In vielen Software-Projekten erfolgt das Testen auf der "rechten Seite" dieses abstrakten Modells, da hier eine erste ausführbare Form der Ziel-Software vorliegt, die mit Hilfe automatisierter Testverfahren überprüft werden kann. Diese

192. vgl. (Hoffmann, 2013, S. 406ff.)
193. vgl. (Frühauf et al., 2007, S. 19)
194. vgl. (Frühauf et al., 2007, S. 132)

Verfahrensweise entspricht dem Bottom-Up-Prinzip, bei dem vom kleinsten Element bis hin zum gesamten Programm getestet wird.

Nach Scholz, Vigenschow und Liggesmeyer existieren vier Stufen für den Testablauf (siehe Abbildung 37), die sich auf die Abstraktionsebenen von Abbildung 36 übertragen lassen, wobei durch die jeweilige Stufe die Auswahl der entsprechenden Testmethoden definiert wird. Weiterhin stellt diese Reihenfolge eine hohe Kongruenz zum V-Modell her, welches heute in vielen Software-Entwicklungen Anwendung findet.[195]

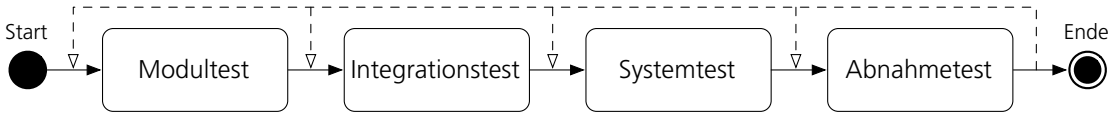



Abb. 37: Bottom-Up Workflow zur Prüfung von Software[196]

In den ersten drei Phasen des Bottom-Up Workflows findet eine Verifikation statt, durch die festgestellt werden soll, ob die erstellte Software die Aufgaben gemäß den Anforderungen korrekt verarbeitet. Die Basis des Bottom-Up Prinzips stellt der Modultest dar (siehe Kapitel 5.2.4), bei dem die einzelnen Komponenten, beziehungsweise Programmeinheiten der Software überprüft werden. Meistens erfolgen diese Tests mit Hilfe von Unit-Tests, durch die eine Überprüfung gegen die Implementierung und somit den kleinsten Programmeinheiten stattfindet.[197] Auf der Stufe des Integrationstests werden logische Modulverbunde in unterschiedlichen Verbindungskomplexitäten gebildet und in regelmäßigen Abständen gegen den Entwurf des Programms getestet. Durch die Verbindung der einzelnen Komponenten zu komplexen Modulen ist es möglich, die Schnittstellen, in denen zahlreiche Informationen zwischen diesen Komponenten ausgetauscht werden, auf Fehler zu überprüfen.[198] Nachdem alle Komponenten zu sinnvollen Modulen zusammenfasst wurden und das Gesamtsystem vorliegt, erfolgt der Systemtest gegen die Anforderungen. Hierbei werden alle Anforderungen an die Software mit Hilfe möglichst realistischer Testdaten auf Erfüllung geprüft, sodass auch Leistungs- und Stresstest aussagekräftig ausfallen.[199] In der letzten Phase, dem Abnahmetest, findet eine Validierung des Systems statt, sodass belegt werden kann, ob das System die gestellten Aufgaben auch erfüllen kann. Hier stellt sich heraus, ob letztendlich am gewünschten System vorbei gearbeitet wurde, wenn beispielsweise wichtige Eigenschaften fehlen oder unnötige Funktionen umgesetzt wurden. Im Idealfall entspricht der Funktionsumfang genau den Wünschen des Kunden. Die verschiedenen Phasen können immer wieder von neuem durchgearbeitet werden, wobei auch ein Rückschritt über mehrere Phasen möglich ist, sollten beispielsweise Lücken in den Testfällen identifiziert werden. Aber auch der nachträgliche Einsatz weiterer Testmethoden zur Komplettierung

195. vgl. (Scholz, 2005, S. 202f.) & vgl. (Vigenschow, 2010, S. 173) & vgl. (Liggesmeyer, 2009, S. 371f.)
196. in Anlehnung an (Vigenschow, 2010, S. 173)
197. vgl. (Liggesmeyer, 2009, S. 371f.)
198. vgl. (Frühauf et al., 2007, S. 74f.)
199. vgl. (Liggesmeyer, 2009, S. 376f.)

der Testergebnisse wäre ein weiteres Szenario in vorherige Prozessphasen zurückzukehren. Letztendlich wird mit jeder Phase die Wirksamkeit der Prüfung erhöht, existierende Fehler lokalisiert und bis zu einem gewünschten Restfehlergrad reduziert. Allerdings ist ein Rückschritt immer mit zusätzlichen Kosten verbunden, die bei der Planung mit berücksichtigt werden sollten. Im Idealfall ist ein Test so gut strukturiert, vorbereitet und durchgeführt, so dass ein Rückschritt in vorherige Testphasen nicht notwendig wird.[200]

### 5.2.6.2 Top-Down Methode

Zusätzlich zu den Testverfahren des Bottom-Up Ansatzes zur Fehlersuche auf der jeweiligen Abstraktionsebene, kann auf der "linken Seite" (siehe Abbildung 36) durch geeignete statische Testmethoden getestet werden. Diese Tests verfahren nach dem Top-Down-Prinzip (siehe Abbildung 38), von der groben Vorstellung der Software (Anforderungen), bis zur detaillierten Umsetzung (Quellcode), da diese keinen ausführbaren Code benötigen.

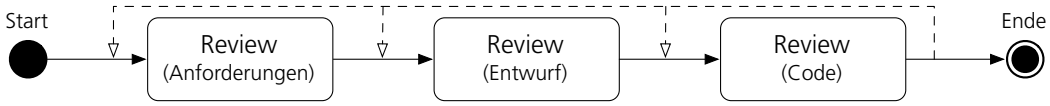



Abb. 38: Top-Down Workflow zur Prüfung von Software[201]

In der ersten Phase erfolgt ein Review über die erhobenen und spezifizierten Anforderungen auf qualitative Eigenschaften wie etwa Vollständigkeit, Korrektheit und Verständlichkeit. Reviews bieten eine Möglichkeit, Anforderungsdokumente auf Mängel zu überprüfen und tragen maßgeblich zur Erhöhung der Anforderungsqualität bei. Das Ergebnis ist eine erste potentiell Fehlerliste, über die im Rahmen einer Diskussion entschieden wird, welche Fehler behoben werden müssen. Grundsätzlich wird in der zweiten und dritten Phase, dem Review gegen den Software-Entwurf und dem Programm-Code, nach dem gleichen Schema verfahren, bei dem jeder dieser Teile gründlich im analogen Format inspiziert und über das Ergebnis diskutiert wird. Jede dieser einzelnen Phasen kann in mehreren Iterationen durchgearbeitet oder auch mehrere Phasen zurückgeschritten werden, sollten Mängel auf grundlegende Fehler der vorherigen Phase zurückzuführen sein, so dass die Wirksamkeit der Prüfung erhöht wird und letztendlich eine Reduktion der Fehler bis zu einem gewünschten Grad erfolgen kann.[202]

Fehler gilt es in möglichst frühen Phasen der Projektentwicklung zu erkennen und zu beseitigen, da die Kosten zur Fehlerbehebung mit dem Projektverlauf, bis hin zur Veröffentlichung des Produktes und darüber hinaus, exponentiell ansteigen (siehe Kapitel 5.2.2). Reviews (siehe Kapitel 5.2.3.1) sind in den ersten Projektphasen das Mittel der Wahl, um Fehler aus den frühen Phasen der Software-Entwicklung zu reduzieren oder zu vermeiden, da ohne eine Programmausführung Fehler in Anforderungen und den ersten Entwürfen identifiziert werden können, wodurch sich in diesen Bereichen kostenintensive Folgefehler reduzieren lassen. Weiterhin lassen sich mit den Reviews auch Codefehler ermitteln. Es ist vor allem wichtig

200. vgl. (Frühauf·et·al.,·2007,·S.·79)
201. in Anlehnung an (Frühauf·et·al.,·2007,·S.·19)
202. vgl. (Frühauf·et·al.,·2007,·S.·132ff.)

durch die Reviews die Anzahl der Fehler in den Anforderungen und den Entwürfen zu minimieren, da durch diese in späteren Phasen der automatisieren Tests, den Struktur- und Funktionstests, immens höhere Kosten zur Behebung entstehen.[203]

### 5.2.6.3 Hybrid Methode

Eine Verknüpfung der beiden verschiedenen Ansätze (siehe Abbildung 39), Top-Down und Bottom-Up, entspricht in diesem Fall einer optimieren Vorgehensweise, durch die sich die Vorteile beider Methoden miteinander verknüpfen lassen.

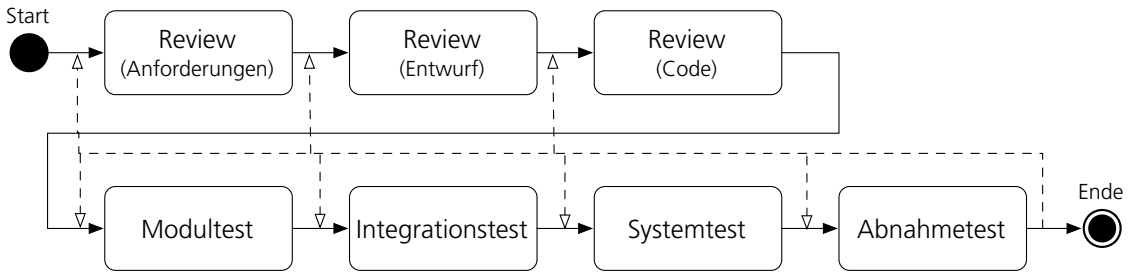



Abb. 39: Verknüpfung von Top-Down und Bottom-Up Worklow zur Software-Prüfung

Mit diesem Ansatz können bereits in den Fehlerentstehungsphasen, den Anforderungen, dem Entwurf und der Code-Entwicklung, Mängel identifiziert und beseitigt werden. Die Reduktion dieser Fehler sollten im Fokus jedes Entwicklers liegen, da die Kosten für eine spätere Beseitigung (siehe Kapitel 5.2.2) immens anwachsen. Mit Hilfe geeigneter Review Techniken oder externer Gutachter, die größere Erfahrung in diesem Bereich haben, können hier eine Vielzahl von Problemen von den Anforderungen über den Entwurf Schritt für Schritt reduziert werden, so dass bereits nach der Codierung ein relativ robustes Programm vorliegt. Gleichwohl erfolgt eine Review des Codes, da gemäß der statischen Methoden (siehe Kapitel 5.2.3.1) viele Fehler, die durch den menschlichen Gedankengang erzeugt wurden am Besten durch diesen wieder identifiziert werden können. In diesem Kontext spielen vor allem semantische Fehler, sogenannte Logikfehler eine wichtige Rolle, die durch Automatismen schwer oder mit erheblichem Aufwand identifiziert werden können. Erst nach diesen ausführlichen Schritten finden automatisierte Tests der einzelnen Module statt, da Belegungen von Variablen oder Methodendurchläufe durch diese dynamischen Verfahren (siehe Kapitel 5.2.3.2) stark erleichtert werden und eine größere Abdeckung von möglichen Zuständen erlauben, die ein Mensch nie erreichen kann. In besonders kritischen Fällen kann auch der formale Nachweis (siehe Kapitel 5.2.3.3) auf Korrektheit eines Programms notwendig sein, je nachdem welche Anforderungen an das System und das Einsatzfeld gestellt werden. Auch die einzelnen Verknüpfungen der Module, von kleinen bis zu großen Modulkonstrukten oder umgekehrt, werden durch die verschiedenen Integrationsstufen überprüft und erfolgen wie auch die vorherigen Prüfverfahren losgelöst von der später eingesetzten Hardware. Ab der Phase des Systemtests erfolgt eine Verbindung von Software mit Hardware, die unter besonderen Gesichtspunkten getestet wird. Sofern ein Test

203. vgl. (Frühauf·et·al.,·2007,·S.·20)

nicht zufriedenstellend ist, kann wie es auch in den Bottom-up und Top-Down Ansätzen der Fall ist, in vorherigen Phasen zurück geschritten werden, bis ein befriedigender Korrektheitsgrad basierend auf den gestellten Anforderungen an das System nachgewiesen werden kann. Diese kann beispielsweise der Nachweis sein, dass eine Software in keinen für den Menschen gefährdenden Prozess verbleibt, sondern in angemessner Zeit in einen sicheren Zustand übergeht. Der Abnahmetest stellt in diesen Zusammenhang die Nachweise für die aufgestellten Kriterien gemäß der Anforderungen dar, die erfüllt sein müssen, damit eine Abnahme des fertigen Systems erfolgen kann.

### 5.2.6.4 Workflow abgeleitet aus dem Hybrid-Modell

Aus den bisherigen Betrachtungen lässt sich ein idealisierter Workflow (siehe Abbildung 40) für die Software-Prüfung ableiten, der die aufgezeigten Methoden zum Testen von Software subsumiert. Zusätzlich impliziert dieses Modell die in Kapitel 5.2.2 (Fehler-Entstehung, -Verstärkung und Kosten) beschriebenen Theorieren, sodass eine effiziente Entwicklung und Prüfung realisiert werden kann, sofern die einzelnen Schritte eingehalten werden. Mit Hilfe einer Validierung im praktischen Betriebsumfeld von Herstellern und Prüfern könnte das Modell abschließend verifiziert und gegebenenfalls weiter optimiert werden. Der Workflow lässt sich hierbei in verschiedene, aufeinanderfolgende Aktivitäten aufgliedern, die jeweils einen Beitrag zum gesamten Prozess leisten.

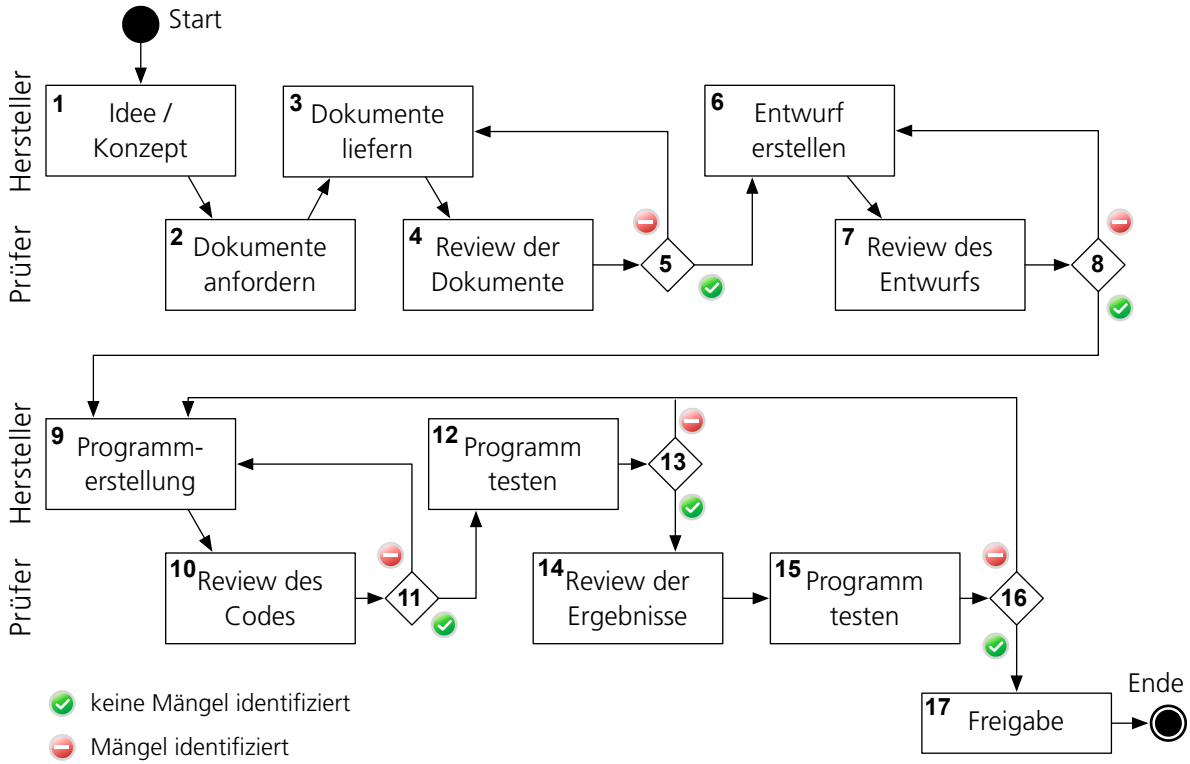



Abb. 40: Workflow zur Prüfung von Software

1. Wenn ein Hersteller eine Idee oder ein Konzept für eine Entwicklung hat, nimmt dieser Kontakt mit Prüfer des Instituts auf.

2. Der Prüfer formuliert im gemeinsamen Gespräch, welche Informationen er insgesamt vom Hersteller benötigt (bspw.: Anforderungen, Entwurf, ...) und in welcher Reihenfolge der Hersteller diese abliefern muss, damit der Workflow der Software-Prüfung reibungslos erfolgen kann.

3. Der Hersteller formuliert zu der Idee, bzw. dem Konzept die gewünschten Anforderungen (funktionale / nicht funktionale), gemäß der Kriterien des Prüfers und liefert diese ab.

4. Der Prüfer führt ein statisches Review über die ausgelieferten Anforderungen durch und prüft somit vorab auf offensichtliche Mängel.

5. Sollten Prüfer Mängel in den Anforderungen identifizieren, werden diese dem Hersteller mitgeteilt und er muss diese zuerst nachbessern, sodass die Schritte 3 bis 5 wiederholt werden müssen. Wenn keine Mängel festgestellt werden, wird mit der nächsten Aktivität fortgefahren.

6. Der Hersteller formuliert zu den Anforderungen einen ersten Entwurf, wie etwa Use-Case Diagramme, Abhängigkeiten, etc. anhand dessen eine genaue Funktionalität abgeschätzt werden kann und stellt diese dem Prüfer zur Verfügung.

7. Der Prüfer führt ein statisches Review über die eingereichten Entwürfe und prüft wiederum auf Mängel.

8. Sollten Prüfer Mängel in den Entwürfen identifizieren, werden diese dem Hersteller mitgeteilt und er muss diese zuerst nachbessern, sodass die Schritte 6 bis 8 wiederholt werden müssen. Wenn keine Mängel festgestellt werden, wird mit der nächsten Aktivität fortgefahren.

9. Auf Basis der Anforderungen und des Entwurfs, erstellt der Hersteller die Software zu seinem Produkt, gemäß den Kriterien des Prüfers und reicht den erstellten Code ein. Dies umfasst außerdem aussagekräftige Methoden- und Variablennamen, Kommentierungen, Testroutinen und weitere zusätzliche Dokumente, die den Programm-Code erläutern.

10. Der Prüfer führt zuerst ein statisches Review über die eingereichten Programm-Code durch und prüft diesen auf offensichtliche Mängel.

11. Sollten Prüfer Mängel im Programm-Code identifizieren, werden diese dem Hersteller mitgeteilt und er muss diese zuerst nachbessern, sodass die Schritte 9 bis 11 wiederholt werden müssen. Wenn keine Mängel festgestellt werden, wird mit der nächsten Aktivität fortgefahren.

12. Auf Basis eigener formulierter Testfälle muss der Hersteller, mit Hilfe geeigneter dynamischer Testverfahren, den erstellen Programm-Code auf fehlerhaftes Verhalten prüfen und über die Testergebnisse eine umfangreiche Dokumentation erstellen.

13. Sollte der Hersteller Mängel im Programm-Code identifizieren, werden diese dokumentiert und er muss diese zuerst nachbessern, sodass die Schritte 9 bis 13 wiederholt werden müssen. Wenn keine Mängel festgestellt werden, übergibt der Hersteller den Programm-Code mit seiner bisherigen Testdokumentation an den Prüfer.

14. Der Prüfer führt ein statisches Review über die Testdokumentation des Herstellers durch und wertet diese aus.

15. Anschließend wird die Software mit Hilfe zusätzlicher Testfälle und geeigneter dynamischer Testmethoden vom Prüfer getestet.

16. Sollten Prüfer Mängel im Programm-Code identifizieren, werden diese dem Hersteller mitgeteilt und er muss diese zuerst nachbessern, sodass die Schritte 9 bis 16 wiederholt werden müssen. Wenn keine Mängel festgestellt werden, wird mit der nächsten Aktivität fortgefahren.

17. Der Prüfer dokumentiert abschließend die Software als ausreichend geprüft und gibt diese frei, sodass der Gesamtprozess der Zertifizierung fortgeführt werden kann.

Der geprüfte Programm-Code kann in anknüpfenden Testverfahren, bei der das Zusammenspiel von Hardware und Software im Fokus liegen, eingesetzt werden, um die Praxistauglichkeit belegen zu können.

## 5.3 Qualitätsindex

Auf Basis von qualitativen Erhebungen, sowohl bei Herstellern als auch bei Prüfinstituten, wurden beispielhafte Prozesse zur Baumustermusterprüfung abgleitet. Die anschließende Analyse und Bewertung der Prüfabläufe, hinsichtlich der Realisierbarkeit durch Software, ermöglicht eine Filterung geeigneter Aktivitäten, bei denen eine Abbildung und Unterstützung durch IT-Systeme möglich bzw. realistisch erscheint. Aus den qualitativen Erhebungen sollten detaillierte Informationen über Vorgehensweisen bei der Erstellung von Software-Prüfdokumentationen erworben und dabei insbesondere grundlegende Problematiken bei der Erstellung angemessener Dokumente identifiziert werden.

Mit dem Ziel einer Bewertung von Software-Prüfdokumentationen und der damit verbundenen methodischen Vorbereitung der späteren Entwicklungsarbeiten, wurden Kriterien zur Qualitätsbewertung festgelegt. Anhand dieser Kriterien soll die Bestimmung der Qualität (Qualitätsindex) von Dokumentformaten zur Abbildung von Software-Prüfvorgängen möglich werden.

Hierbei orientiert sich der Aufbau des Kriterienkataloges für den Qualitätsindex an bewährten Strukturen des Software-Engineering. Es werden Qualitätskriterien vorgestellt, die bei der Qualitätsbewertung von Dokumentformaten genutzt werden sollen. Dadurch wird eine Vereinheitlichung in Form und Layout unter den einzelnen Kriterien des Index realisiert, sodass eine Gesamtbewertung über alle Kriterien möglich wird.[204] Weiterhin wird auf Standards zurückgegriffen, wir beispielsweise die IEEE 930[205] oder die IEC 13849-3[206], anhand derer die einzelnen Kriterien zur Qualitätsbestimmung ermittelt wurden.

204. vgl. (Röder, Franke, Müller, & Przybylski, 2009)
205. vgl. (IEEE Computer Society, 1998)
206. vgl. (Deutsches Institut für Normung, 2008b)

Der Qualitätsindex soll Kriterien zur Bestimmung der Qualität von Dokumentformaten für die Abbildung von Software-Prüfvorgängen schriftlich festhalten. Durch derartige Kriterien kann die Eignung von Dokumentformaten untersucht werden, inwieweit diese für die Dokumentation von Software-Spezifikationen geeignet sind. Das Dokument liefert eine erste Übersicht der Qualitätskriterien zur Beurteilung der Qualität von Formaten von Software-Prüfdokumenten. Aufgrund des flexiblen Aufbaus kann das Dokument weiter ergänzt werden, sofern neue Kriterien identifiziert werden, mit denen eine genauere Qualitätsmessung möglich wird.

Zur besseren Übersicht und Bewertung wird eine Aufteilung in verschiedene Kategorien getroffen, sodass eine Zuordnung von Qualitätskriterien zu diesen Gruppen und eine weitere Differenzierung möglich wird. Für die Zuordnung in diese Gruppen existieren zwei Konzepte, die gegensätzlich zueinander verfahren, dem Konzept der Spezialisierung auf der einen und dem Konzept der Generalisierung auf der anderen Seite. Bei der Spezialisierung wird versucht ein allgemeines Qualitätskriterium so weit zu spezialisieren, dass eine eindeutige Zuordnung in eine der vorgegeben Kategorien möglich wird. Hierbei kann eine Abspaltung in einzelne unabhängige Kriterien erfolgen, damit eine bessere Einordnung möglich wird. Sobald ein Kriterium sich nicht mehr weiter spezialisieren lässt oder aufgrund der weiteren Spezialisierung in mehreren Kategorien eingeordnet werden müsste, ist die Spezialisierung beendet. Beim Konzept der Generalisierung hingegen wird versucht, ein spezielles Qualitätskriterium weiter zu verallgemeinern und es auf diese Weise in eine generalisierende Gruppe einzuordnen. Durch die Methode der Generalisierung werden doppelte Kriterien lokalisiert und isoliert, sodass diese nicht mehrfach aufgelistet werden.

Jedes identifizierte Qualitätskriterium wird durch eine kurze Beschreibung eingeleitet, sodass ein Grundverständnis des Merkmals gegeben ist und deutlich wird, welche Aufgabe dieses erfüllen soll. Daneben wird die Funktion beschrieben, die dieses Merkmal realisieren muss, damit es einen positiven Betrag zur Gesamtqualität des Index leisten kann.

Auf Basis der Funktionsbeschreibung kann eine Antwort über die Erfüllung des Merkmals abgegeben werden. Zu diesem Zweck wird durch die Beurteilung knapp formuliert, was beim zugehörigen Merkmal bewertet werden soll. Eine Beantwortung erfolgt durch die tabellarische Auflistung von verschiedenen Fragen, die vorgegeben sind. Es wird in drei Beantwortungskategorien unterschieden.

1. In der ersten und einfachsten Variante werden zwei Fragen formuliert, die eindeutig mit einem Ja oder Nein zu beantworten sind und dazu die passende Wertungshöhe identifiziert werden kann.

2. In der zweiten Variante ist es notwendig mehr als zwei Fragen zu beantworten, bei der mehrere Fragen mit Ja und nur eine Frage mit Nein beantwortet werden können. In diesem Fall müssen die Punkte der Ja und Nein Antworten zusammen addiert werden.

3. In der letzten Variante ist eine Beantwortung in aufsteigender Reihenfolge möglich, bei der abgeschätzt werden muss in welchem prozentualen Ausmaß das entsprechende Merkmal zutrifft und dazu die passende Wertungshöhe entnommen werden.

Durch die jeweilige Antwort werden die zutreffenden Punkte ermittelt und in die Ergebniszeile der Bewertungstabelle überführt (im 2. Fall bereits aufaddiert). Das Einzelergebnis darf jedoch nicht die maximale Punktezahl (5 Punkte) überschreiten.

Nachdem alle Qualitätsmerkmale überprüft wurden, muss die Wertungshöhe in die Tabelle eingetragen werden. Am Ende wird ein prozentualer Wert ausgewiesen, der die Gesamtqualität des Dokumentformates widerspiegelt. Je höher dieser Wert ausfällt, desto besser geeignet erscheint das evaluierte Format zur Abbildung der Informationen von Software-Prüfvorgängen. Je niedriger dieser Wert ausfällt, desto weniger geeignet ist das evaluierte Format im Umkehrschluss. Das Maximum stellt hierbei hundert Prozent, das Minimum null Prozent dar.

### 5.3.1 Allgemeine Informationen zum Projektstand

#### Können Meta-Informationen abgebildet werden? (#01-01)

**Beschreibung:** Dieses Qualitätsmerkmal bewertet die Möglichkeit der Integration von Meta-Informationen zu einem Projekt. Meta-Informationen liefern einen zusätzlichen Informationsbeitrag, der eine Arbeit mit den Inhalten vereinfachen kann. Hierunter fallen insbesondere Informationen, die nur einmalig eingegeben werden müssen, wie beispielsweise ein Firmenname oder ein Projektname.

**Funktion:** Meta-Informationen können verwendet werden, um mehr Komfort bereitzustellen, beispielsweise indem bekannte Daten verwendet werden, um Dokumente auszufüllen, wenn ein Dokument erzeugt oder eine Email formuliert werden muss.

Neben diesen statischen Informationen existieren aber auch dynamische Informationen, die sich im Verlauf eines Projektes verändern können. Auch diese Informationen werden verwendet, um den Benutzer erhöhten Komfort während der Bearbeitung eines Projektes zu liefern. Beispiele hierfür stellen die Anzeigen von Versionen des Projektes dar oder der Hinweis auf inhaltliche Änderungen von verschiedenen Dokumenten.

*Meta-Informationen können unter anderem folgendes Wissen beinhalten:*

- Name der Firma (statisch)
- Projektname (statisch)
- beteiligte Prüfer (statisch)
- Start und Ende der Prüfung (statisch)
- Individuelle Vorgangserfassungsnummer (statisch)
- Version (dynamisch)
- Status (dynamisch)

Meta-Informationen sind grundsätzlich für den Benutzer nicht direkt veränderbar und werden teilweise versteckt abgelegt, damit eine einfache Manipulation von außen abgewehrt werden kann. Änderungen erfolgen ausschließlich über Eingabemasken oder Automatismen, um diese angemessen zu pflegen.

**Beurteilung:** Es wird validiert, ob Meta-Informationen zu einem Projekt integriert werden können.

**Beantwortung:**

| Fragen | Antwort | Wertung |
|---|---|---|
| Können Meta-Informationen dynamisch integriert werden? | Ja | 1 Punkt |
| Können Meta-Informationen statisch integriert werden? | Ja | 2 Punkte |
| Können Meta-Informationen integriert werden? | Ja | 2 Punkte |
| Können Meta-Informationen integriert werden? | Nein | 0 Punkte |
| | **Ergebnis (Max. 5 P.)** | |

### Kann der Status des Projektes nachgehalten werden? (#01-02)

**Beschreibung:** Dieses Qualitätsmerkmal bewertet die Möglichkeit der Abbildung eines Projektstatus. Anhand des Status wird es möglich, die Bearbeitungsphase und den Stand des Projektes im zeitlichen Verlauf darzustellen und zu dokumentieren.

**Funktion:** Der Status eines Projektes symbolisiert seinen Zustand und zeigt dem Benutzer an, in welcher Bearbeitungsphase sich dieses befindet.

*Mögliche Status eines Projektes:*

- Antrag
- Entwurf
- Prüfung
- Zertifizierung

Anhand dieses symbolischen Auszeichnungsmerkmals können zu erledigende Aufgaben dem jeweiligen Status zugeordnet werden. Dadurch verfügt der Benutzer, zu jeder möglichen Phase, über eine Zusammenfassung, beispielsweise in Form einer Checkliste, die verarbeitet werden muss, um zum nächsten Status eines Projektes wechseln zu können.

**Beurteilung:** Es wird validiert, ob der Status zu einem Projekt nachgehalten werden kann.

**Beantwortung:**

| Fragen | Antwort | Wertung |
|---|---|---|
| Kann der Status nachgehalten werden? | Ja | 5 Punkte |
| Kann der Status nachgehalten werden? | Nein | 0 Punkte |
| | **Ergebnis (Max. 5 P.)** | |

## Kann die Version des Projektes abgebildet werden? (#01-03)

**Beschreibung:** Dieses Qualitätsmerkmal bewertet die Möglichkeit, die Version eines Projektes erfassen zu können. Zu diesem Zweck muss das Projekt eine eindeutige Nummer beinhalten, welche die Version identifiziert.

**Funktion:** Eine Nummer zur Visualisierung des Versionsstandes sollte fortlaufend generiert werden und muss eindeutig sein. Dies kann in Form einer aufsteigenden Nummerierung erfolgen, die mit der Ziffer eins beginnt. Die Versionsnummer muss immer dann aktualisiert werden, wenn Inhalte des Projektes geändert wurden, indem die Nummer um den Wert eins inkrementiert wird.

Durch die Versionierung wird es schneller möglich nachzuvollziehen, ob und welche Teile des Projektes geändert wurden (abhängig von der Tiefe der Versionierung). Hierzu muss das Projekt lediglich oberflächlich überflogen und auf eine Veränderung der Versionsnummer überprüft werden.

**Beurteilung:** Es wird validiert, ob die Version zu einem Projekt nachgehalten werden kann.

**Beantwortung:**

| Fragen | Antwort | Wertung |
|---|---|---|
| Kann die Version nachgehalten werden? | Ja | 5 Punkte |
| Kann die Version nachgehalten werden? | Nein | 0 Punkte |
| | **Ergebnis (Max. 5 P.)** | |

## Kann die Korrespondenz abgebildet werden? (#01-04)

**Beschreibung:** Dieses Qualitätsmerkmal bewertet die Möglichkeit der Integration von Korrespondenz, wodurch der Informationsfluss zwischen zwei Parteien bereitgestellt werden kann.

**Funktion:** Die Korrespondenz bildet alle ausgetauschten Informationen zwischen zwei Parteien ab, sodass sichtbar wird, welche Informationen ausgetauscht wurden. Neben den Inhalten der Korrespondenz muss das Datum des Versandes oder des Einganges dokumentiert werden und sofern die jeweilige Korrespondenz Anhänge beinhaltet auch diese.

Die Historie der Korrespondenz liefert einen schnellen Einblick über den Verlauf ausgetauschter Informationen, sodass nachvollzogen werden kann, welche Informationen zu einem definierten Zeitpunkt versendet und / oder eingetroffen sind.

Korrespondenz kann hierbei auf verschiedene Arten entstehen, beispielsweise durch Austausch von Emails, Briefen, Telefonaten, FTP-Transfer und andere Kanälen, sollte aber grundsätzlich in ein digitales Format überführt werden, um dieses in der Historie speichern zu können.

**Beurteilung:** Es wird validiert, ob Korrespondenz zu einem Projekt nachgehalten werden kann.

Beantwortung:

| Fragen | Antwort | Wertung |
|---|---|---|
| Kann die Korrespondenz vollständig abgebildet werden? | 100% | 5 Punkte |
| Kann die Korrespondenz überwiegend abgebildet werden? | 50% - 99% | 3 Punkte |
| Kann die Korrespondenz teilweise abgebildet werden? | 1% - 49% | 2 Punkte |
| Kann die Korrespondenz nicht abgebildet werden? | 0% | 0 Punkte |
| | Ergebnis (Max. 5 P.) | |

## 5.3.2 Allgemeine Eigenschaften der Dokumentation

### Kann das Format verschiedene Bereiche abbilden? (#02-01)

**Beschreibung:** Dieses Qualitätsmerkmal bewertet die Möglichkeit, Inhalte in verschiedenen Bereichen der Dokumentation zu erfassen. Eine Aufspaltung in verschiedene Bereiche ermöglicht die Bündelung von gleichartigen Inhalten in einer Dokumentation.

**Funktion:** Das separierte Speichern von Informationen in der Dokumentation gewährleistet eine klare Trennung gleichartiger Inhalte in dafür vorgesehenen Bereiche. Eine sinnvolle Aufteilung muss mindestens in zwei Bereiche erfolgen, da diese klar voneinander zu trennen sind.

*Mögliche Dokumentbereiche:*

- Bereich für Software-Spezifikation (siehe Kapitel 5.3.3)
- Bereich für Quellcode (siehe Kapitel 5.3.4)

Jeder einzelne Dokumentbereich stellt unterschiedliche Anforderungen, die für eine Wiedergabeform der mitgelieferten Dokumente nutzbringend sind. Software-Spezifikationen weisen beispielsweise einen speziellen Aufbau im Textformat sowie Formulierungen auf und müssen sehr viele Inhaltsinformationen aufnehmen. Wohingegen Quellcode in einer speziellen, stark strukturierten Form formuliert wird, sodass diese maschinenlesbar interpretiert werden kann.

Durch die Trennung in verschiedene Bereichsgruppen für anfallende Dokumente wird es dem Leser und Ersteller erleichtert die gesamten Information besser zu strukturieren, wodurch die Inhalte logisch und physisch leichter unterschieden werden können.

Durch eine Bereichstrennung wird auch die Anwenderfreundlichkeit mit wachsender Komplexität und steigendem Anzahl und Inhalt der Dokumente stark erhöht.

**Beurteilung:** Es wird validiert, ob die Dokumentation Inhalte in verschiedenen Bereiche erfassen kann.

Beantwortung:

| Fragen | Antwort | Wertung |
|---|---|---|
| Kann die Dokumentation weitere Bereiche abbilden? | Ja | 1 Punkt |
| Kann die Dokumentation Quellcode abbilden? | Ja | 2 Punkte |
| Kann die Dokumentation Software-Spezifikationen abbilden? | Ja | 2 Punkte |
| Kann die Dokumentation keine verschiedenen Bereiche abbilden? | Ja | 0 Punkte |
| | **Ergebnis (Max. 5 P.)** | |

### Kann eine Dokumenthierarchie abgebildet werden? (#02-02)

**Beschreibung:** Dieses Qualitätsmerkmal bewertet die Möglichkeit, eine Dokumenthierarchie abzubilden. Mit Hilfe einer Dokumenthierarchie kann die Ablagestruktur von Dokumenten zur besseren Übersicht dargestellt werden.

**Funktion:** Das Nachhalten einer Dokumentstruktur ist immer dann hilfreich, wenn viele einzelne Dokumente vorgehalten werden müssen. Dadurch kann die Übersichtlichkeit erhöht werden, da nicht alle Dateien auf derselben Ebene abgelegt werden.

Eine derartige Ablagestruktur wird zumeist mit Hilfe von hierarchisch organisierten Ordnern realisiert, wodurch Dokumente besser sortiert und kategorisiert werden können. So können verschiedenartige Dokumenttypen in dafür vorgesehene Strukturordner, wie beispielsweise ein Ordner für Quellcode oder Abbildungen, abgelegt werden.

**Beurteilung:** Es wird validiert, ob eine Dokumenthierarchie abgebildet werden kann.

Beantwortung:

| Fragen | Antwort | Wertung |
|---|---|---|
| Kann eine Dokumenthierarchie abgebildet werden? | Ja | 5 Punkte |
| Kann eine Dokumenthierarchie abgebildet werden? | Nein | 0 Punkte |
| | **Ergebnis (Max. 5 P.)** | |

### Können sicherheitsrelevante Bereiche abgebildet werden? (#02-03)

**Beschreibung:** Dieses Qualitätsmerkmal bewertet die Möglichkeit, Inhalte in sicherheitsrelevanten Bereichen der Dokumentation zu erfassen. Eine Trennung in spezielle Bereiche ermöglicht die Bündelung von gleichartigen Inhalten in einer Dokumentation.

**Funktion:** Sicherheitsrelevante Bereiche bündeln diejenigen Informationen, die im Falle von problematischen Verhaltensweisen (bspw. Programm-Fehlern) zu einer negativen Auswirkung im Zusammenhang mit dem Benutzer führen könnten. Ein Beispiel hierfür stellt ein unvorhersehbares Fehlverhalten des Prüf-

objektes dar, welches zu einem Personenschaden führt. Aus diesem Grund stellen sicherheitsrelevante Bereiche eine erhöhte Wichtigkeit für die Prüfung durch den Prüfer des Prüfinstitutes dar.

Eine eindeutige visuelle Trennung von Inhalten und Zuordnung zu sicherheitsrelevanten Bereichen erleichtert dem Prüfer diese wichtigen Informationen schnell lokalisieren zu können, sodass eine Prüfung ggf. effizienter durchgeführt werden kann.

**Beurteilung:** Es wird validiert, ob sicherheitsrelevante Bereiche abgebildet werden können.

**Beantwortung:**

| Fragen | Antwort | Wertung |
|---|---|---|
| Können sicherheitsrelevante Bereiche abgebildet werden? | Ja | 5 Punkte |
| Können sicherheitsrelevante Bereiche abgebildet werden? | Nein | 0 Punkte |
| | **Ergebnis (Max. 5 P.)** | |

### Können nicht sicherheitsrelevante Bereiche abgebildet werden? (#02-04)

**Beschreibung:** Dieses Qualitätsmerkmal bewertet die Möglichkeit, Inhalte in nicht sicherheitsrelevanten Bereichen der Dokumentation zu erfassen. Eine Trennung in spezielle Bereiche ermöglicht die Bündelung von gleichartigen Inhalten in einer Dokumentation.

**Funktion:** Nicht sicherheitsrelevante Bereiche bündeln diejenigen Informationen, die im Falle von problematischen Verhaltensweisen (bspw. Programm-Fehlern) zu keiner negativen Auswirkung im Zusammenhang mit dem Benutzer führen könnten. Ein Beispiel hierfür stellt der Ausfall eines Monitors dar, sodass der Benutzer keine Informationen mehr ablesen kann.

Aus diesem Grund stellen nicht sicherheitsrelevante Bereiche eine weniger hohe Wichtigkeit für die Prüfung durch den Prüfer des Prüfinstitutes dar.

Eine eindeutige visuelle Trennung von Inhalten und Zuordnung zu nicht sicherheitsrelevanten Bereichen erleichtert dem Prüfer diese weniger wichtigen Informationen schnell trennen zu können, sodass eine Prüfung ggf. effizienter durchgeführt werden kann.

**Beurteilung:** Es wird validiert, ob nicht sicherheitsrelevante Bereiche abgebildet werden können.

**Beantwortung:**

| Fragen | Antwort | Wertung |
|---|---|---|
| Können nicht sicherheitsrelevante Bereiche abgebildet werden? | Ja | 5 Punkte |
| Können nicht sicherheitsrelevante Bereiche abgebildet werden? | Nein | 0 Punkte |
| | **Ergebnis (Max. 5 P.)** | |

## Kann Tracking abgebildet werden? (#02-05)

**Beschreibung:** Dieses Qualitätsmerkmal bewertet die Möglichkeit Tracking abzubilden. Tracking ermöglich die visuelle Abbildung der Zusammengehörigkeit von Spezifikationsstrukturen (Anforderungen) mit den entsprechenden Programmfunktionen im Quellcode.

**Funktion:** Damit eine visuelle Darstellung von Tracking grundsätzlich möglich wird, muss zuvor Forward-Tracking (Spezifikation -> Code) und / oder Backward-Tracking (Code -> Spezifikation) in der Spezifikation und dem Quellcode, mit Hilfe von eindeutigen Merkmalen zur Identifikation der einzelnen Strukturen, erfolgen (siehe Abbildung 41).

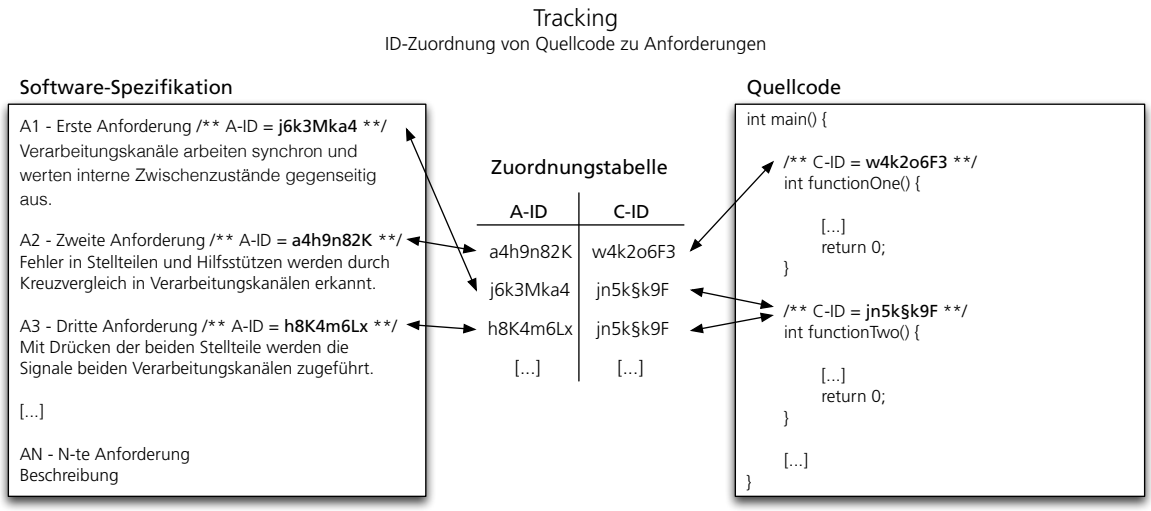



Abb. 41: Tracking mit Zuordnung von Anforderungen zum Quellcode über Zuordnungstabelle

Die Merkmale dienen als Zuordnung zwischen den Anforderungen der Software-Spezifikation und den einzelnen Funktionen des Quellcode. Anhand dieses eindeutigen Merkmals zur Identifikation kann zu einer gewünschten Anforderung eine schnelle visuelle Zuordnung zu der erstellten Funktionen im Quellcode erfolgen und umgekehrt.

Eine visuelle Verknüpfung von Dokumenten erleichtert die Prüfung von Spezifikation gegen den Quellcode, da zu einer Anforderung hierdurch eine schnelle Möglichkeit existiert, die passenden Funktionen aufzufinden und dahingehend auf Vollständigkeit und Korrektheit zu überprüfen.

**Beurteilung:** Es wird validiert, ob Tracking in der Dokumentation abgebildet werden kann.

**Beantwortung:**

| Fragen | Antwort | Wertung |
|---|---|---|
| Kann Tracking abgebildet werden? | Ja | 5 Punkte |
| Kann Tracking abgebildet werden? | Nein | 0 Punkte |
| | Ergebnis (Max. 5 P.) | |

## Können Informationen zu Normen abgebildet werden? (#02-06)

**Beschreibung:** Dieses Qualitätsmerkmal bewertet die Möglichkeit, Normen, Vorschriften oder Richtlinien in der Dokumentation erfassen zu können. Normen, Vorschriften oder Richtlinien dienen dem Leser als Ausgangsbasis zur Interpretation der Inhalte, auf deren Basis Spezifikationsdokumente oder Quellcode erstellt wurden.

**Funktion:** Normen, Vorschriften oder Richtlinien stellen meist komplexe Dokumente dar, die verwendet werden, um die einzelnen Dokumente der gesamten Dokumentation zu erstellen, beispielsweise die Software-Spezifikation.

Damit die Arbeit mit den Dokumenten erleichtert werden kann, müssen alle verwendeten Normen, Vorschriften oder Richtlinien, soweit dies zugelassen wird, vorgehalten werden.

**Beurteilung:** Es wird validiert, ob Informationen zu Normen in der Dokumentation erfasst werden können.

**Beantwortung:**

| Fragen | Antwort | Wertung |
|---|---|---|
| Können Informationen zu Normen erfasst werden? | Ja | 5 Punkte |
| Können Informationen zu Normen erfasst werden? | Nein | 0 Punkte |
| | **Ergebnis (Max. 5 P.)** | |

## Können Kommentare abgebildet werden? (#02-07)

**Beschreibung:** Dieses Qualitätsmerkmal bewertet die Möglichkeit, Kommentare in der Dokumentation erfassen zu können. Kommentare ermöglichen es dem Leser, Notizen zu einzelnen Dokumenten oder Inhalten zu verfassen, die als Grundlage von Verbesserungen dienen.

**Funktion:** Kommentare stellen Anmerkungen dar, mit denen Verbesserungen, Probleme, Aufgaben und weitere Informationen, in Bezug auf einzelne Dokumente und Inhalte der Dokumentation, formuliert werden können.

Auf der einen Seite können Leser mit Hilfe von Kommentaren detaillierte Verbesserungsvorschläge zu Dokumenten oder den Inhalten formulieren und in einer Liste für den Verfasser bündeln, sodass dieser möglicherweise einen Anmerkungskatalog als Übersicht durchzuführender Aufgaben erhält. Auf der anderen Seite können Verfasser zusätzliche Anmerkungen zu den geforderten Dokumenten oder Inhalten formulieren, wie beispielsweise Inhaltsfragen oder erweiterte Informationen, die dem Leser eine Interpretation erleichtern können. Dadurch könnte auch dem Leser die Möglichkeit geboten werden, eine Zusammenfassung von Anmerkungen zu bündeln. In beiden Fällen erleichtert eine Kommentierung die Interpretation und Bearbeitung der Inhalte.

Eine Kommentarliste kann als Checkliste für den Verfasser und Leser dienen und erleichtert demzufolge die Sammlung von zu erledigenden Aufgaben und insbesondere die Überprüfung auf deren vollständige Bearbeitung.

**Beurteilung:** Es wird validiert, ob Kommentare in der Dokumentation erfasst werden können.

**Beantwortung:**

| Fragen | Antwort | Wertung |
|---|---|---|
| Kann der Verfasser Kommentare verfassen? | Ja | 1 Punkt |
| Kann der Leser Kommentare verfassen? | Ja | 2 Punkte |
| Können Kommentare erfasst werden? | Ja | 2 Punkte |
| Können Kommentare erfasst werden? | Nein | 0 Punkte |
| | **Ergebnis (Max. 5 P.)** | |

## Können Checklisten abgebildet werden? (#02-08)

**Beschreibung:** Dieses Qualitätsmerkmal bewertet die Möglichkeit, Checklisten in der Dokumentation erfassen zu können. Checklisten fassen Arbeitspunkte aus abgeleiteten Tätigkeiten zusammen und bündeln diese im Regelfall in einer listenhaften Aufzählungsform.

**Funktion:** Checklisten dienen dazu, Arbeitsabläufe in Prozessen zu unterstützen. Zu jedem Prozess in einer Prozesskette werden hierzu die notwendigen Arbeitspunkte (Aufgaben) formuliert, die ausgeführt sein müssen, um zum nächsten Prozess in der Prozesskette zu gelangen.

Jede Checkliste beinhaltet somit Aufgaben, die bearbeitet werden müssen oder bereits bearbeitet wurden und dient als Übersicht des Arbeitsstandes im Prozess. Eine bearbeitete Aufgabe kann symbolisch, beispielsweise mit einem Haken, als ausgeführt markiert werden. Alle Aufgeben, die nicht markiert wurden, gelten als unbearbeitet.

Durch Checklisten kann zu jedem Prozessschritt schnell eingesehen werden, welche Aufgaben noch zu bearbeiten sind und weiterhin eine Überprüfung auf Vollständigkeit der auszuführenden Aufgaben erfolgen.

**Beurteilung:** Es wird validiert, ob Checklisten in der Dokumentation erfasst werden können.

**Beantwortung:**

| Fragen | Antwort | Wertung |
|---|---|---|
| Können Checklisten erfasst werden? | Ja | 5 Punkte |
| Können Checklisten erfasst werden? | Nein | 0 Punkte |
| | **Ergebnis (Max. 5 P.)** | |

## Kann die Version der Dokumentation abgebildet werden? (#02-09)

**Beschreibung:** Dieses Qualitätsmerkmal bewertet die Möglichkeit, die Version der Dokumentation erfassen zu können. Zu diesem Zweck muss die Dokumentation eine eindeutige Nummer beinhalten, welche die Version identifiziert.

**Funktion:** Die Nummer zur Visualisierung des Versionsstandes sollte fortlaufend generiert werden und muss eindeutig sein. Dies kann in Form einer aufsteigenden Nummerierung erfolgen, die mit der Ziffer eins beginnt. Die Versionsnummer muss immer dann aktualisiert werden, nachdem Inhalte der Dokumentation verändert wurden, indem die Nummer um den Wert eins inkrementiert wird.

Durch die Versionierung wird es schneller möglich nachzuvollziehen, ob und welche Teile der Dokumentation geändert wurden. Hierzu muss die Dokumentation oberflächlich überflogen und auf eine Veränderung der Versionsnummer überprüft werden.

Weiterhin kann für einen erhöhten Detailgrad die Versionierung des betroffenen Dokumentes eingesehen werden, sofern diese vorhanden ist.

**Beurteilung:** Es wird validiert, ob die Version der Dokumentation erfasst werden kann.

**Beantwortung:**

| Fragen | Antwort | Wertung |
|---|---|---|
| Kann eine Version der Dokumentation erfasst werden? | Ja | 5 Punkte |
| Kann eine Version der Dokumentation erfasst werden? | Nein | 0 Punkte |
| | **Ergebnis (Max. 5 P.)** | |

## Kann eine Änderungshistorie abgebildet werden? (#02-10)

**Beschreibung:** Dieses Qualitätsmerkmal bewertet die Möglichkeit, eine Änderungshistorie abzubilden, durch die Änderungen protokolliert werden können.

**Funktion:** Mit Hilfe einer Änderungshistorie (zumeist in tabellarischer Form) wird dokumentiert, welche Teile (Dokumente) der Dokumentation zwischen verschiedenen Versionen geändert wurden.

Hierzu muss nach jeder Änderung an der Dokumentation ein neuer Eintrag mit einer kurzen Beschreibung der getätigten Änderungen im Historienverzeichnis erfolgen, sodass eine Änderungsübersicht entsteht.

Durch die Änderungshistorie kann schneller überblickt werden, welche Dokumente verändert wurden. Weiterhin kann für einen erhöhten Detailgrad die Änderungshistorie des betroffenen Dokumentes eingesehen werden, sofern diese vorhanden ist.

**Beurteilung:** Es wird validiert, ob eine Änderungshistorie in der Dokumentation erfasst werden kann.

Beantwortung:

| Fragen | Antwort | Wertung |
|---|---|---|
| Kann eine Änderungshistorie erfasst werden? | Ja | 5 Punkte |
| Kann eine Änderungshistorie erfasst werden? | Nein | 0 Punkte |
| | Ergebnis (Max. 5 P.) | |

## 5.3.3 Software-Spezifikation

### Kann ein Deckblatt abgebildet werden? (#03-01)

**Beschreibung:** Dieses Qualitätsmerkmal bewertet die Möglichkeit, ein Deckblatt abbilden zu können, um wichtige grundlegende Informationen zur Software-Spezifikation schriftlich festzuhalten, wie beispielsweise Titel, Autoren oder den Status des Dokumentes.

**Funktion:** Diese Informationen liefern dem Leser eine kompakte Übersicht und müssen auf dem Deckblatt dokumentiert werden. Auf diese Wiese kann zum Beispiel, eine eindeutige Zuordnung zum Autor bzw. Hersteller sichergestellt oder die Version des Dokumentes überprüft werden.

*Das Deckblatt sollte ein Mindestmaß an Informationen aufführen:*

- Titel des Dokuments
- Projekttitel
- Firmenname (ggf. mit Logo)
- Autor oder Autoren
- Version mit Datum
- Status

**Beurteilung:** Es wird validiert, ob ein Deckblatt in der Software-Spezifikation erfasst werden kann.

Beantwortung:

| Fragen | Antwort | Wertung |
|---|---|---|
| Kann das Deckblatt vollständig erfasst werden? | 100% | 5 Punkte |
| Kann das Deckblatt überwiegend erfasst werden? | 50% - 99% | 3 Punkte |
| Kann das Deckblatt teilweise erfasst werden? | 1% - 49% | 2 Punkte |
| Kann kein Deckblatt erfasst werden? | 0% | 0 Punkte |
| | Ergebnis (Max. 5 P.) | |

## Kann die Version der Software-Spezifikation abgebildet werden? (#03-02)

**Beschreibung:** Dieses Qualitätsmerkmal bewertet die Möglichkeit, den Versionsstand der Software-Spezifikation abbilden zu können. Mit Hilfe einer Versionierung wird für den Leser sichtbar, ob die Inhalte der Software-Spezifikation geändert wurden.

**Funktion:** Die Software-Spezifikation muss eine eindeutige Nummer beinhalten, welche die Version identifiziert. Hierzu muss innerhalb des Software-Spezifikation der Versionsstand eindeutig abbildet werden.

Eine Nummer zur Visualisierung des Versionsstandes sollte fortlaufend generiert werden und muss eindeutig sein. Dies kann in Form einer aufsteigenden Nummerierung erfolgen, die mit der Ziffer eins beginnt. Die Versionsnummer muss immer dann aktualisiert werden, wenn Inhalte der Software-Spezifikation geändert wurden, indem die Nummer um den Wert eins inkrementiert wird.

Die Versionsnummer muss mindestens auf dem Deckblatt dargestellt werden. Für eine feinere Abstufung kann auch eine Versionsnummer je Kapitel erzeugt werden.

Durch die Versionierung wird es schneller möglich nachzuvollziehen, welche Teile der Software-Spezifikation geändert wurden (abhängig von der Tiefe der Versionierung). Hierzu muss die Spezifikation lediglich oberflächlich überflogen werden.

**Beurteilung:** Es wird validiert, ob die Version der Software-Spezifikation erfasst werden kann.

**Beantwortung:**

| Fragen | Antwort | Wertung |
|---|---|---|
| Kann die Version der Software-Spezifikation erfasst werden? | Ja | 5 Punkte |
| Kann die Version der Software-Spezifikation erfasst werden? | Nein | 0 Punkte |
| | **Ergebnis (Max. 5 P.)** | |

## Kann ein Verzeichnis mit Änderungshistorie abgebildet werden? (#03-03)

**Beschreibung:** Dieses Qualitätsmerkmal bewertet die Möglichkeit, ein Versionsverzeichnis abbilden zu können, in dem Änderungen zwischen den verschiedenen Software-Spezifikationen dokumentiert werden.

**Funktion:** Mit Hilfe eines Versionsverzeichnisses (zumeist in tabellarischer Form) wird dokumentiert, welche Stellen bzw. Inhalte zwischen den verschiedenen Software-Spezifikationen geändert wurden.

Nach jeder Änderung in der Software-Spezifikation muss ein neuer Eintrag mit Versionsnummer und einer kurzen Beschreibung der getätigten Änderungen im Versionsverzeichnis erfolgen.

Der letzte Eintrag einer Version sollte mit der Versionsnummer auf dem Deckblatt kongruent sein und insbesondere keine Lücken zwischen den verschiedenen Versionen aufweisen. Die Änderungshistorie muss somit fortlaufend dokumentiert werden.

Durch das Versionsverzeichnis mit den Änderungsbeschreibungen kann schneller überblickt werden, welche Bereiche der Software-Spezifikation geändert wurden, wobei der Nutzen vom Detailgrad und der Qualität der Beschreibung abhängig ist.

Weiterhin bietet das Versionsverzeichnis die Möglichkeit, die Vollständigkeit der Historie zu überprüfen, indem die Nummerierung der Versionen auf Lücken in der aufsteigenden Zählung untersucht wird.

**Beurteilung:** Es wird validiert, ob ein Verzeichnis mit Änderungshistorie in der Software-Spezifikation erfasst werden kann.

**Beantwortung:**

| Fragen | Antwort | Wertung |
|---|---|---|
| Kann ein Versionsverzeichnis erfasst werden? | Ja | 5 Punkte |
| Kann ein Versionsverzeichnis erfasst werden? | Nein | 0 Punkte |
| | **Ergebnis (Max. 5 P.)** | |

### Kann ein Inhaltsverzeichnis abgebildet werden? (#03-04)

**Beschreibung:** Dieses Qualitätsmerkmal bewertet die Möglichkeit, ein Inhaltsverzeichnis abbilden zu können, welches dazu dient einen Übersicht der dokumentierten Inhalte der Software-Spezifikation zusammenzustellen.

**Funktion:** Das Inhaltsverzeichnis stellt eine kompakte Auflistung (siehe Tabelle 5) aller Inhalte der Software-Spezifikation dar, die mittels kompakter Gliederung abgebildet werden. Für jede Gliederungsüberschrift in der Software-Spezifikation muss ein Eintrag im Inhaltsverzeichnis erscheinen und im Wortlaut mit diesem übereinstimmen.

Damit eine Unterscheidung der Gliederungselemente, unabhängig vom formulierten Titel, erfolgen kann, muss eine Nummerierung in aufsteigender Form vorhanden sein. Diese beginnt beim ersten Gliederungselement (anknüpfend an das Inhaltsverzeichnis), wobei für jedes vorhandene Element die Nummer um den Wert eins inkrementiert wird, bis das letze Element erreicht ist.

Gliederungsüberschriften können im Dokument abgestuft werden, wodurch verschiedene Inhaltsebenen entstehen. Mit jeder neuen Unterebene wird die Nummerierung der darüber liegenden Ebene übernommen und um eine differenzierte Gliederungsnummer erweitert, die dem gleichen Konzept der Hauptebene folgt.

Weiterhin muss zu jedem Eintrag die Seitennummer aufgeführt werden, auf der dieser aufzufinden ist. Diese wird durch entsprechende Füllzeichen, zumeist in Form von Punkten, am rechten Seitenrand aufgeführt, sodass eine einfache Zuordnung zum Gliederungselement möglich ist.

| Nummerierung | Gliederungselement | Seite |
|---|---|---|
| 1 | Hauptebene 1..... | 1 |
| 1.1 | Unterebene 1 von Hauptebene 1 ..... | 3 |
| 1.1.1 | Unterunterebene 1 von Unterebene 1 ..... | 4 |
| 1.1.2 | Unterunterebene 2 von Unterebene 1 ..... | 5 |
| 1.2 | Unterebene 2 von Hauptebene 1 ..... | 7 |
| 2 | Hauptebene 2 ..... | 10 |
| ... | ..... | ... |
| N | Hauptebene N ..... | N |

Tabelle 5: Schema einer Gliederungsabbildung in einer Software-Spezifikation

Durch das Inhaltsverzeichnis wird die Anwenderfreundlichkeit mit wachsender Komplexität und steigendem Inhalt des Dokumentes stark erhöht. Es kann eine Überprüfung auf Vollständigkeit der gesamten Software-Spezifikation erfolgen, indem das Inhaltsverzeichnis auf Lücken in der aufsteigenden Zählung untersucht wird.

**Beurteilung:** Es wird validiert, ob ein Inhaltsverzeichnis in der Software-Spezifikation erfasst werden kann.

**Beantwortung:**

| Fragen | Antwort | Wertung |
|---|---|---|
| Kann ein Inhaltsverzeichnis erfasst werden? | Ja | 5 Punkte |
| Kann ein Inhaltsverzeichnis erfasst werden? | Nein | 0 Punkte |
| | **Ergebnis (Max. 5 P.)** | |

## Kann ein Abbildungsverzeichnis abgebildet werden? (#03-05)

**Beschreibung:** Dieses Qualitätsmerkmal bewertet die Möglichkeit, ein Abbildungsverzeichnis abbilden zu können und dient dazu, eine Übersicht der eingebundenen, grafischen Inhalte zu bündeln, die in der Software-Spezifikation verwendet werden.

**Funktion:** Die Darstellung eines Abbildungsverzeichnisses erfolgt in Form einer Liste (siehe Tabelle 6) mit kongruenter Bezeichnung, wie die Abbildung im Textverlauf beschriftet wurde. Für jede Abbildung in der Software-Spezifikation muss ein Eintrag im Abbildungsverzeichnis erscheinen und im Wortlaut mit der Abbildungsbeschriftung übereinstimmen.

| Elemente | Abbildungstitel | Seite |
|---|---|---|
| Abbildung 1: | Titel der ersten Abbildung ..... | 2 |
| Abbildung 2: | Titel der zweiten Abbildung ..... | 5 |
| ... | ..... | ... |
| Abbildung N: | Titel der n-ten Abbildung ..... | 20 |

Tabelle 6: Schema eines Abbildungsverzeichnisses in einer Software-Spezifikation

Die Auflistung erfolgt in fortlaufender Reihenfolge, in der die Abbildungen im Verlauf der Software-Spezifikation verwendet werden. Jeder Eintrag beginnt mit einer eindeutigen Nummerierung in aufsteigender Form und endet mit einer Seitenzahl als Referenz, an welcher Stelle die Abbildung in der Software-Spezifikation aufzufinden ist. Mit jedem weiteren Eintrag wird die Nummerierung um den Wert eins inkrementiert.

Durch das Abbildungsverzeichnis wird die Anwenderfreundlichkeit mit wachsender Komplexität und steigendem Inhalt des Dokumentes stark erhöht. Der Leser erhält durch diese Verzeichnisse die Möglichkeit, den Abbildungsort von gesuchten Objekten schneller wieder auffinden zu können. Dadurch wird die Suche nach speziellen Inhalten stark vereinfacht. Anhand der Nummerierung kann eine Überprüfung auf Vollständigkeit der aufgelisteten Abbildungen erfolgen, indem die Liste(n) auf Lücken in der aufsteigenden Zählung untersucht wird.

Zumeist wird das Abbildungsverzeichnis am Anfang der Software-Spezifikation im Anschluss an das Inhaltsverzeichnis eingefügt und liefert dem Leser einleitend eine Übersicht der verwendeten Abbildungen.

**Beurteilung:** Es wird validiert, ob ein Abbildungsverzeichnis in der Software-Spezifikation erfasst werden kann.

**Beantwortung:**

| Fragen | Antwort | Wertung |
|---|---|---|
| Kann ein Abbildungsverzeichnis erfasst werden? | Ja | 5 Punkte |
| Kann ein Abbildungsverzeichnis erfasst werden? | Nein | 0 Punkte |
| | **Ergebnis (Max. 5 P.)** | |

### Kann ein Tabellenverzeichnis abgebildet werden? (#03-06)

**Beschreibung:** Dieses Qualitätsmerkmal bewertet die Möglichkeit, ein Tabellenverzeichnis abbilden zu können und dient dazu eine Übersicht der eingebundenen tabellarischen Inhalte zu bündeln, die in der Software-Spezifikation verwendet werden.

**Funktion:** Die Darstellung erfolgt in Form einer Liste (siehe Tabelle 7) mit kongruenter Bezeichnung, wie die Tabelle im Textverlauf beschriftet wurde. Für jede Tabelle in der Software-Spezifikation muss ein Eintrag im Tabellenverzeichnis erscheinen und im Wortlaut mit der Tabellenbeschriftung übereinstimmen.

| Elemente | Tabellentitel | Seite |
|---|---|---|
| Tabelle 1: | Titel der ersten Tabelle ..... | 2 |
| Tabelle 2: | Titel der zweiten Tabelle ..... | 5 |
| ... | ..... | ... |
| Tabelle N: | Titel der n-ten Tabelle ..... | 20 |

Tabelle 7: Schema eines Tabellenverzeichnisses in einer Software-Spezifikation

Die Auflistung erfolgt in fortlaufender Reihenfolge, in der die Tabellen im Verlauf der Software-Spezifikation verwendet werden. Jeder Eintrag beginnt mit einer eindeutigen Nummerierung in aufsteigender Form und endet mit einer Seitenzahl als Referenz, an welcher Stelle die Tabelle in der Software-Spezifikation aufzufinden ist. Mit jedem weiteren Eintrag wird die Nummerierung um den Wert eins inkrementiert.

Durch das Tabellenverzeichnis wird die Anwenderfreundlichkeit mit wachsender Komplexität und steigendem Inhalt des Dokumentes stark erhöht. Der Leser erhält durch diese Verzeichnisse die Möglichkeit, den Abbildungsort von gesuchten Objekten schneller wieder auffinden zu können. Dadurch wird die Suche nach speziellen Inhalten stark vereinfacht. Anhand der Nummerierung kann eine Überprüfung auf Vollständigkeit der aufgelisteten Abbildungen oder Tabellen erfolgen, indem die Liste(n) auf Lücken in der aufsteigenden Zählung untersucht wird.

Zumeist wird das Tabellenverzeichnis am Anfang der Software-Spezifikation im Anschluss an das Inhaltsverzeichnis eingefügt und liefert dem Leser einleitend eine Übersicht der verwendeten Tabellen.

**Beurteilung:** Es wird validiert, ob ein Tabellenverzeichnis in der Software-Spezifikation erfasst werden kann.

**Beantwortung:**

| Fragen | Antwort | Wertung |
|---|---|---|
| Kann ein Tabellenverzeichnis erfasst werden? | Ja | 5 Punkte |
| Kann ein Tabellenverzeichnis erfasst werden? | Nein | 0 Punkte |
| | **Ergebnis (Max. 5 P.)** | |

## Kann ein Abkürzungsverzeichnis abgebildet werden? (#03-07)

**Beschreibung:** Dieses Qualitätsmerkmal bewertet die Möglichkeit, ein Abkürzungsverzeichnis abbilden zu können und dient dazu eine Übersicht über die verwendeten Abkürzungen in der Software-Spezifikation zusammenzustellen.

**Funktion:** Die Darstellung erfolgt in Form einer zweispaltigen tabellarischen Auflistung (siehe Tabelle 8), bei der in der linken Spalte die Abkürzung aufgeführt wird und in der rechten Spalte die dazu gehörige Bedeutung in Langform (vollständig ausgeschrieben). Es erfolgt grundsätzlich eine alphabetische Sortierung der aufgeführten Abkürzungen anhand der ersten Spalte in aufsteigender Form.

| Abkürzung | Langform |
|---|---|
| A | Langform der Abkürzung A |
| BB | Langform der Abkürzung BB |
| ... | |
| ZZZ | Langform der Abkürzung ZZZ |

Tabelle 8: Schema eines Abkürzungsverzeichnis in einer Software-Spezifikation

Durch das Abkürzungsverzeichnis wird die Anwenderfreundlichkeit mit wachsender Komplexität und steigendem Inhalt des Dokumentes stark erhöht. Dem Leser wird es ermöglicht, die vollständige Bedeutung hinter einer Abkürzung auf einfache Weise eindeutig nachvollziehen zu können. Dies ist insbesondere dann von Relevanz, wenn sich Abkürzungen in verschiedenen Anwendungsdomänen überschneiden, so dass eine andere Bedeutung für diese vorliegt. Die kompakte Darstellungsform und alphabetische Sortierung ermöglicht es dem Leser, eine Abkürzung schnell nachschlagen zu können, sofern er diese im Kontext des Lesestoffes nicht mehr präsent hat.

Zumeist wird das Abkürzungsverzeichnis am Anfang der Software-Spezifikation im Anschluss an das Inhaltsverzeichnis eingefügt und liefert dem Leser einleitend eine Übersicht der verwendeten Abkürzungen.

**Beurteilung:** Es wird validiert, ob ein Abkürzungsverzeichnis in der Software-Spezifikation erfasst werden kann.

**Beantwortung:**

| Fragen | Antwort | Wertung |
|---|---|---|
| Kann ein Abkürzungsverzeichnis erfasst werden? | Ja | 5 Punkte |
| Kann ein Abkürzungsverzeichnis erfasst werden? | Nein | 0 Punkte |
| | **Ergebnis (Max. 5 P.)** | |

## Kann ein Begriffslexikon abgebildet werden? (#03-08)

**Beschreibung:** Dieses Qualitätsmerkmal bewertet die Möglichkeit, ein Begriffslexikon oder auch Glossar abbilden zu können, durch das die wichtigsten Begriffe der jeweiligen Spezialgebiete dokumentiert werden sollen.

**Funktion:** Die Darstellung erfolgt in Form einer zweispaltigen tabellarischen Auflistung (siehe Tabelle 9), bei der in der linken Spalte ein Begriff, bzw. eine Definition aufgeführt wird und in der rechten Spalte die dazu gehörige ausführliche Bedeutung formuliert steht. Es erfolgt grundsätzlich eine alphabetische Sortierung der aufgeführten Begriffe anhand der ersten Spalte in aufsteigender Auflistung.

| Begriff | Beschreibung |
|---|---|
| A | Beschreibung des Begriffs A |
| B | Beschreibung des Begriffs B |
| ... | |
| Z | Beschreibung des Begriffs Z |

Tabelle 9: Schema eines Begriffslexikon in einer Software-Spezifikation

Durch das Begriffslexikon wird die Anwenderfreundlichkeit mit wachsender Komplexität und steigendem Inhalt des Dokumentes stark erhöht. Es werden hierdurch nicht nur Unklarheiten zwischen Hersteller und Prüfer vermieden, sondern auch Worte mit spezieller Bedeutung im Anwendungskontext erklärt. Weiterhin müssen spezielle Begriffe nicht an mehreren Stellen in der Software-Spezifikation erläutert werden, wodurch unter anderem Inkonsistenzen bei der Interpretation von Fachworten verhindert werden. Die kompakte Darstellungsform und alphabetische Sortierung ermöglicht es dem Leser, spezielle Fachbegriffe, die im Kontext des Lesestoffes formuliert werden, schnell nachschlagen zu können.

Zumeist wird das Begriffslexikon am Anfang der Software-Spezifikation im Anschluss an das Inhaltsverzeichnis eingefügt und liefert dem Leser einleitend eine Übersicht der verwendeten Fachbegriffe.

**Beurteilung:** Es wird validiert, ob ein Begriffslexikon in der Software-Spezifikation erfasst werden kann.

**Beantwortung:**

| Fragen | Antwort | Wertung |
|---|---|---|
| Kann ein Begriffslexikon erfasst werden? | Ja | 5 Punkte |
| Kann ein Begriffslexikon erfasst werden? | Nein | 0 Punkte |
| | **Ergebnis (Max. 5 P.)** | |

## Kann ein Normenverzeichnis abgebildet werden? (#03-09)

**Beschreibung:** Dieses Qualitätsmerkmal bewertet die Möglichkeit, ein Normenverzeichnis abbilden zu können und dient dazu eine Übersicht der angewendeten Normen, Vorschriften oder Richtlinien in der Software-Spezifikation zusammenzustellen.

**Funktion:** Die Darstellung erfolgt in Form einer Auflistung (siehe Tabelle 10), wobei eine Gruppierung nach Normen, Vorschriften oder Richtlinien erfolgen sollte und dazu eine aussagekräftige Beschreibung formuliert steht.

| Normen | Beschreibung |
|---|---|
| DIN Norm 1 | Titel der Norm 1 |
| … | … |
| DIN Norm N | Titel der Norm N |

| Vorschriften | Beschreibung |
|---|---|
| Vorschrift 1 | Beschreibung der Vorschrift 1 |
| | |
| Vorschrift N | Beschreibung der Vorschrift N |

| Richtlinien | Beschreibung |
|---|---|
| Richtlinie 1 | Beschreibung der Richtlinie 1 |
| | |
| Richtlinie N | Beschreibung der Richtlinie N |

Tabelle 10: Schema eines Normenverzeichnisses in einer Software-Spezifikation

Anhand eines Normenverzeichnis wird dem Leser eine Liste der anzuwendenden Normen und Vorschriften aufgezeigt, die in der Software-Spezifikation berücksichtigt wurden.

Das Normenverzeichnis stellt einen Leitfaden der Normen und Vorschriften dar, die ein Hersteller bei der Erstellung seines Produktes berücksichtigen muss, sofern dies durch einen Prüfer zuvor definiert wurde und / oder welche der Hersteller berücksichtigt hat, sofern noch keine Rücksprache mit einem Prüfer erfolgte.

Prüfer eines Prüfinstitutes können durch das Normenverzeichnis prüfen, ob alle vorgegebenen Normen und Vorschriften im Kontext der Spezifikation berücksichtigt wurden und auch im Normenverzeichnis notiert stehen.

**Beurteilung:** Es wird validiert, ob ein Normenverzeichnis in der Software-Spezifikation erfasst werden kann.

Beantwortung:

| Fragen | Antwort | Wertung |
|---|---|---|
| Kann ein Normverzeichnis erfasst werden? | Ja | 5 Punkte |
| Kann ein Normverzeichnis erfasst werden? | Nein | 0 Punkte |
| | Ergebnis (Max. 5 P.) | |

## Kann ein Literaturverzeichnis abgebildet werden? (#03-10)

**Beschreibung:** Dieses Qualitätsmerkmal bewertet die Möglichkeit, ein Literaturverzeichnis abbilden zu können, welches eine Übersicht über die verwendeten Quellen zusammenstellt, auf Basis derer die Software-Spezifikation erstellt wurde.

**Funktion:** Anhand eines Literaturverzeichnis wird dem Leser eine Liste der zu Hilfe genommenen Literatur aufgezeigt, die in der Software-Spezifikation berücksichtigt wurde. Die Darstellung erfolgt in einer alphabetisch sortieren Liste in aufsteigender Aufzählung, wobei ein Eintrag wichtige Merkmale beinhalten muss, um eine Identifikation der Literatur zu ermöglichen.

*Ein vollständiger Eintrag im Literaturverzeichnis beinhaltet wichtige Merkmale:*

- Titel der Quelle
- Autor (Name, Vorname)
- Herausgeber (soweit vorhanden)
- Auflage (sofern mehrere Auflagen vorhanden)
- Verlag
- Erscheinungsort
- Erscheinungsjahr
- URL oder Webadresse (sofern elektronische Quelle)

Sollte der Leser im Bedarfsfall auf eine Originalquelle zurückgreifen müssen, so kann dieser auf Basis der aufgelisteten Merkmale auf die Originalquelle schließen. Dies kann vor allem dann sinnvoll sein, wenn mehrere Auflagen einer Literatur existieren, in der sich verwendete Passagen geändert haben und somit zu einem anderen Ergebnis in der Dokumentation führen.

**Beurteilung:** Es wird validiert, ob ein Literaturverzeichnis in der Software-Spezifikation erfasst werden kann.

Beantwortung:

| Fragen | Antwort | Wertung |
|---|---|---|
| Kann das Literaturverzeichnis vollständig erfasst werden? | 100% | 5 Punkte |
| Kann das Literaturverzeichnis überwiegend erfasst werden? | 50% - 99% | 3 Punkte |
| Kann das Literaturverzeichnis teilweise erfasst werden? | 1% - 49% | 2 Punkte |
| Kann kein Literaturverzeichnis erfasst werden? | 0% | 0 Punkte |
| | **Ergebnis (Max. 5 P.)** | |

### Kann ein Dokumentverzeichnis abgebildet werden? (#03-11)

**Beschreibung:** Dieses Qualitätsmerkmal bewertet die Möglichkeit, ein Dokumentverzeichnis abbilden zu können, welches eine Übersicht über die Dokumente zusammenstellt, die zusätzlich zur Software-Spezifikation erstellt und mit übergeben wurden.

**Funktion:** Anhand des Dokumentverzeichnis wird dem Leser eine Liste der mitgelieferten Dokumente aufgezeigt, die neben der Software-Spezifikation formuliert wurden. Diese Liste dient dazu, den Leser darüber zu informieren, welche weiteren gültigen Dokumente, neben der Software-Spezifikation, berücksichtigt werden müssen, wie beispielsweise ein ausführlicher Plan mit der Beschreibung von Software-Testfällen oder die allgemeine Software-Dokumentation.

**Beurteilung:** Es wird validiert, ob ein Dokumentverzeichnis in der Software-Spezifikation erfasst werden kann.

Beantwortung:

| Fragen | Antwort | Wertung |
|---|---|---|
| Kann ein Dokumentverzeichnis erfasst werden? | Ja | 5 Punkte |
| Kann ein Dokumentverzeichnis erfasst werden? | Nein | 0 Punkte |
| | **Ergebnis (Max. 5 P.)** | |

### Können Seitenzahlen abgebildet werden? (#03-12)

**Beschreibung:** Dieses Qualitätsmerkmal bewertet die Möglichkeit, Seitenzahlen abbilden zu können, die einen genauen Überblick über den Umfang einer Software-Spezifikation liefern.

**Funktion:** In Dokumenten muss eine Seitennummerierung in aufsteigender Form erfolgen, beginnend bei der ersten Seite, wobei für jede weitere Seite die Nummer um den Wert eins inkrementiert wird, bis das Ende des Dokumentes erreicht ist (siehe Abbildung 42).

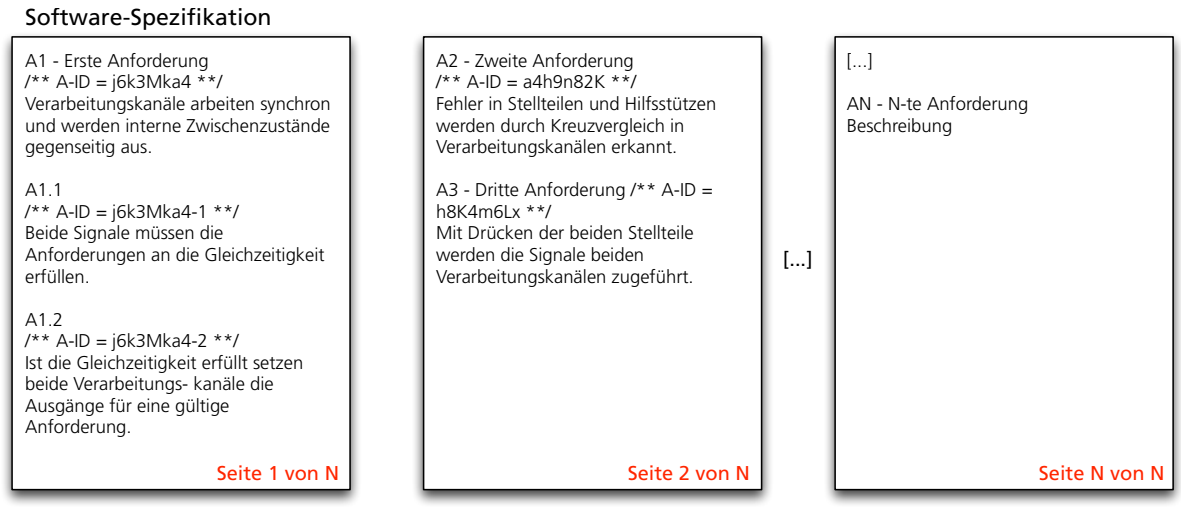



Abb. 42: Beispiel von Seitenzahlen in einer Software-Spezifikation

Anhand der Seitenzahlen wird die Navigation im Dokument erleichtert, da beispielsweise Inhaltsverzeichnis oder Schlagwortverzeichnis auf diese verweisen. Dem Leser wird dadurch das Auffinden von gewünschten Inhalten stark vereinfacht, da er sich nur noch anhand der Referenzierungen orientieren muss.

Dadurch kann eine Überprüfung auf Vollständigkeit der gesamten Software-Spezifikation erfolgen, indem diese auf Lücken in der aufsteigenden Zählung untersucht wird.

Weiterhin bildet eine Nummerierung der einzelnen Seiten die Grundlage für eine Kommunikation zwischen zwei Parteien und bildet somit eine einheitliche Basis zum Informationsaustausch, da diese auf die Seitennummern verweisen kann.

**Beurteilung:** Es wird validiert, ob Seitenzahlen in der Software-Spezifikation erfasst werden können.

**Beantwortung:**

| Fragen | Antwort | Wertung |
|---|---|---|
| Können Seitenzahlen erfasst werden? | Ja | 5 Punkte |
| Können Seitenzahlen erfasst werden? | Nein | 0 Punkte |
| | **Ergebnis (Max. 5 P.)** | |

### Können Zeilennummern abgebildet werden? (#03-13)

**Beschreibung:** Dieses Qualitätsmerkmal bewertet die Möglichkeit, Zeilennummern abbilden zu können, die eine präzise Lokalisierung von Zeilen auf einer definierten Seite in einer Software-Spezifikation ermöglichen.

**Funktion:** In Dokumenten muss eine Zeilennummerierung in aufsteigender Form erfolgen, beginnend bei der ersten Textzeile, wobei für jede weitere Zeile die Nummer um den Wert eins inkrementiert wird, bis das Ende der Seite erreicht ist (siehe Abbildung 43).

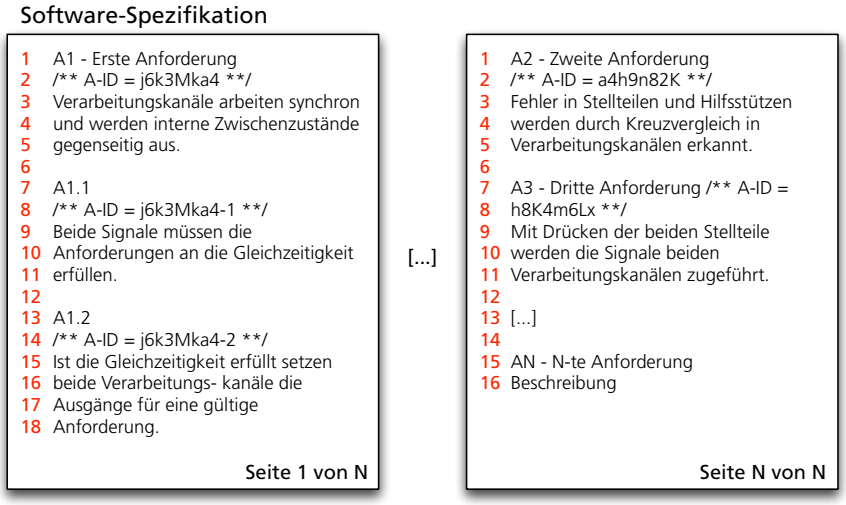



Abb. 43: Beispiel von Zeilennummern in einer Software-Spezifikation

Dadurch können die Inhalte jeder einzelnen Seite auf Vollständigkeit überprüft werden, indem die jeweilige Seite auf Lücken in der aufsteigenden Zählung untersucht wird.

Weiterhin konkretisiert eine Nummerierung der einzelnen Zeilen je Seite, in Kombination mit der Nummerierung der einzelnen Seiten, die Grundlage für eine Kommunikation zwischen zwei Parteien und bildet somit eine präzisierte einheitliche Basis zum Informationsaustausch, da diese auf die Seitennummern verweisen kann.

**Beurteilung:** Es wird validiert, ob Zeilennummern in der Software-Spezifikation erfasst werden können.

**Beantwortung:**

| Fragen | Antwort | Wertung |
|---|---|---|
| Können Zeilennummern erfasst werden? | Ja | 5 Punkte |
| Können Zeilennummern erfasst werden? | Nein | 0 Punkte |
| | Ergebnis (Max. 5 P.) | |

## Können Spezifikationsstrukturen abgebildet werden? (#03-14)

**Beschreibung:** Dieses Qualitätsmerkmal bewertet die Möglichkeit, der Abbildung von einzelnen Spezifikationsstrukturen (Anforderungen), die einen essentiellen Bestandteil einer Software-Spezifikation darstellen. Anforderungen beschreiben die Eigenschaften und Funktionsweise der Software nach denen diese erstellt wird.

**Funktion:** Der zentrale Inhalt und wichtigste Baustein einer Software-Spezifikation besteht aus den formulierten Anforderungen. Mittels dieser Anforderungen werden alle Funktionen des gewünschten Systems detailliert beschrieben und liefern einen Überblick über die Funktionalität der späteren Software. Hierbei kann die Beschreibung auf verschiedenste Weise erfolgen, wie etwa Use-Case-Beschreibungen, Aktivitätsdiagrammen oder anderen Darstellungsformen, die hilfreich sind, um das System funktional zu beschreiben.

Es müssen alle Funktionen vollständig beschrieben werden, wie beispielsweise Funktionen die Hardwarefehler aufdecken und beherrschen, Schnittstellen für ankommende oder ausgehende Informationen zu anderen Systemen oder weitere.

Anhand der formulierten Anforderungen erhält der Leser ein Verständnis über die Funktionsweise und Zusammenhänge der Software und kann auf dieser Grundlage bewerten, ob die Software-Spezifikation ausreichend detailliert ausgeführt wurde.

**Beurteilung:** Es wird validiert, ob Spezifikationsstrukturen in der Software-Spezifikation erfasst werden können.

**Beantwortung:**

| Fragen | Antwort | Wertung |
|---|---|---|
| Können Spezifikationsstrukturen erfasst werden? | Ja | 5 Punkte |
| Können Spezifikationsstrukturen erfasst werden? | Nein | 0 Punkte |
| | **Ergebnis (Max. 5 P.)** | |

## Kann Forward-Tracking abgebildet werden? (#03-15)

**Beschreibung:** Dieses Qualitätsmerkmal bewertet die Möglichkeit, des Forward-Tracking abbilden zu können. Forward-Tracking stellt die Zuordnung von formulierten Anforderungen zu den daraufhin erstellenden Funktionen im Quellcode dar.

**Funktion:** Damit eine Zuordnung zwischen den Anforderungen der Software-Spezifikation und den einzelnen Funktionen des beigefügten Quellcode möglich wird, muss hierzu eine eindeutige Verknüpfung zwischen diesen Informationen formuliert werden (siehe Abbildung 44).

Diese Verknüpfung erfolgt in Form einer eindeutigen Identifikationsnummer (ID), die zu jeder Anforderung erstellt und dieser zugeordnet werden muss. Dies kann beispielsweise als ID-Eintrag zu jeder Anforderung erfolgen. Eine zugeordnete ID darf nur ein einziges Mal in der Software-Spezifikation enthalten sein. Im Quellcode wird die ID der jeweiligen Anforderung der entsprechend zugehörigen Funktion(en) zugeordnet.

Anhand dieses eindeutigen Merkmals zur Identifikation kann zu einer Anforderung eine schnelle visuelle Zuordnung zwischen Anforderung und den dazu erstellten Funktionen im Quellcode erfolgen.

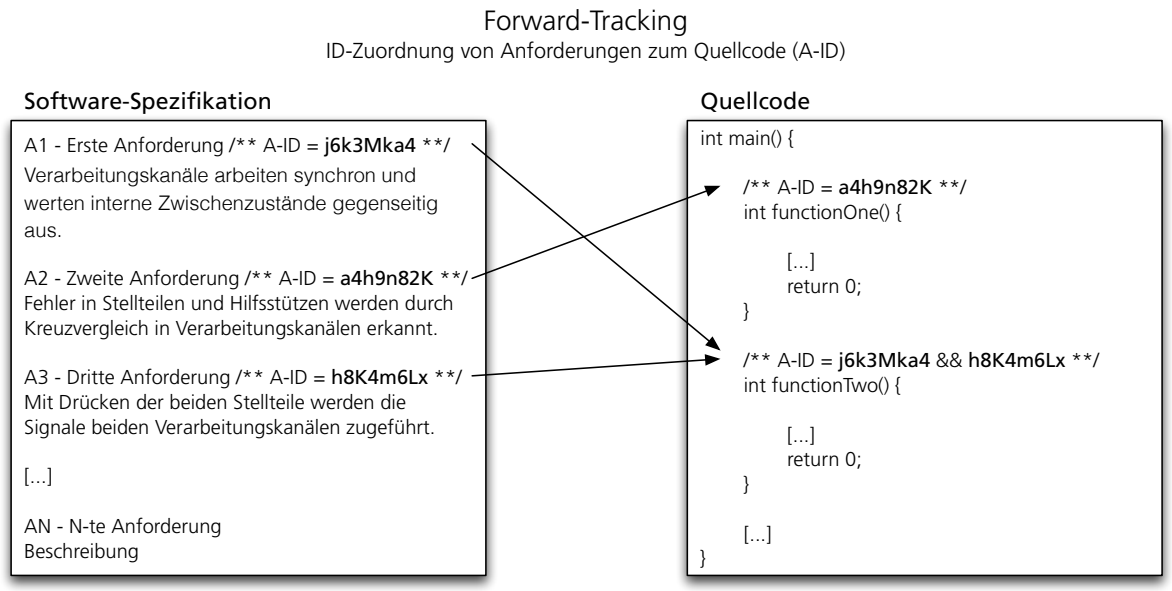



Abb. 44: Forward-Tracking mit Zuordnung von Anforderungen zum Quellcode

**Beurteilung:** Es wird validiert, ob Forward-Tracking in der Software-Spezifikation erfasst werden kann.

**Beantwortung:**

| Fragen | Antwort | Wertung |
|---|---|---|
| Kann Forward-Tracking erfasst werden? | Ja | 5 Punkte |
| Kann Forward-Tracking erfasst werden? | Nein | 0 Punkte |
| | Ergebnis (Max. 5 P.) | |

## Können Fehlerfälle abgebildet werden? (#03-16)

**Beschreibung:** Dieses Qualitätsmerkmal bewertet die Möglichkeit, Fehlerfälle abzubilden. Fehlerfälle oder auch Software-Testfälle stellen eine Sammlung von formulierten Tests dar, mit deren Hilfe die entwickelte Software auf korrekte Funktionsweise überprüft werden kann.

**Funktion:** Eine Software muss während der Programmierung oder spätestens danach auf Fehler hin untersucht werden. Zu diesem Zweck müssen geeignete Testfälle formuliert werden, anhand derer die Software auf korrekte Funktionsweise getestet werden kann.

Insbesondere muss ein Schwerpunkt der Tests auf das Verhalten in einem Fehlerfall erfolgen. Zu diesem Zweck werden Testfälle mit Hilfe verschiedener statischer oder dynamischer Methoden erstellt, wie beispielsweise Tests mit Funktionsüberdeckung oder Tests auf Grenzwerte, anhand derer die Verhaltensweise der Software überprüft werden kann.

Für jede Funktion im Quellcode müssen Testfälle formuliert und den Funktionen zugeordnet werden, um diese zu überprüfen. Die Schwierigkeit besteht darin, ausreichend viele Testfälle zu formulieren, um zu

gewährleisten, dass ausreichend viele Zustände der Software geprüft werden. Hierzu können strukturorientierte Testmethoden eingesetzt werden, auf deren Basis eine Aussage über die wahrscheinliche Sicherheit der Software getroffen werden kann.

Die Abbildung von Testfällen muss nicht zwingend in der eigentlichen Software-Spezifikation erfolgen, sondern kann auch über ein separates Dokument erfolgen, auf welches im Dokumentverzeichnis referenziert wird.

**Beurteilung:** Es wird validiert, ob Fehlerfälle in der Software-Spezifikation erfasst werden kann.

**Beantwortung:**

| Fragen | Antwort | Wertung |
|---|---|---|
| Können Fehlerfälle erfasst werden? | Ja | 5 Punkte |
| Können Fehlerfälle erfasst werden? | Nein | 0 Punkte |
| | **Ergebnis (Max. 5 P.)** | |

### Kann ein Schlagwortverzeichnis abgebildet werden? (#03-17)

**Beschreibung:** Dieses Qualitätsmerkmal bewertet die Möglichkeit, ein Schlagwortverzeichnis abbilden zu können, welches eine Übersicht von Schlagworten zusammenstellt, anhand derer eine Erschließung der Software-Spezifikation möglich wird.

**Funktion:** Durch das Schlagwortverzeichnis, oder auch Index, wird bei Dokumenten mit größerem Umfang die Anwenderfreundlichkeit stark erhöht. Mit wachsender Komplexität und steigendem Inhalt kann es dem Leser schwer fallen, sich in derartigen Dokumenten angemessen zu orientieren. Das Schlagwortverzeichnis bietet die Möglichkeit, wichtige Wörter nachschlagen zu können und insbesondere festzustellen an welchen Stellen (Seitennummer) der Begriff in der Software-Spezifikation aufzufinden ist. Dadurch wird die Suche nach speziellen Inhalten stark vereinfacht.

Zumeist wird das Schlagwortverzeichnis in Form eines Anhanges in der Software-Spezifikation gepflegt und zu Beginn der Spezifikation, in der Einleitung, auf dessen Existenz verwiesen.

**Beurteilung:** Es wird validiert, ob ein Schlagwortverzeichnis in der Software-Spezifikation erfasst werden kann.

**Beantwortung:**

| Fragen | Antwort | Wertung |
|---|---|---|
| Kann ein Schlagwortverzeichnis erfasst werden? | Ja | 5 Punkte |
| Kann ein Schlagwortverzeichnis erfasst werden? | Nein | 0 Punkte |
| | **Ergebnis (Max. 5 P.)** | |

## 5.3.4 Quellcode

### Kann die Version des Quellcodes abgebildet werden? (#04-01)

**Beschreibung:** Dieses Qualitätsmerkmal bewertet die Möglichkeit, den Versionsstand von Quellcode abbilden zu können. Mit Hilfe einer Versionierung wird für den Leser sichtbar, ob die Inhalte der Quelltexte geändert wurden.

**Funktion:** Das Quellcode-Dokument muss eine eindeutige Nummer beinhalten, welche die Version des Dokumentes identifiziert. Hierzu muss innerhalb des Quellcode-Dokumentes der Versionsstand eindeutig abbildet werden.

Eine Nummer zur Visualisierung des Versionsstandes sollte fortlaufend generiert und mindestens im Kopfteil, zu Beginn eines Quellcode-Dokumentes, abgebildet werden. Weiterhin muss die Nummer eindeutig sein. Für eine feinere Abstufung der Version kann auch eine Versionsnummer je Methode erzeugt werden.

Durch die Versionierung wird es schneller möglich nachzuvollziehen, welche Teile der Quelltexte geändert wurden (abhängig von der Tiefe der Versionierung). Hierzu muss der Quelltext lediglich oberflächlich überflogen werden.

**Beurteilung:** Es wird validiert, ob die Version des Quellcodes erfasst werden kann.

**Beantwortung:**

| Fragen | Antwort | Wertung |
|---|---|---|
| Kann die Version des Quellcodes erfasst werden? | Ja | 5 Punkte |
| Kann die Version der Quellcodes erfasst werden? | Nein | 0 Punkte |
| | **Ergebnis (Max. 5 P.)** | |

### Können Programm-Strukturen abgebildet werden? (#04-02)

**Beschreibung:** Dieses Qualitätsmerkmal bewertet die Möglichkeit, Programm-Strukturen abbilden zu können. Programm-Strukturen stellen einzelne oder auch mehrere Funktionen (Methoden) des beigefügten Quellcodes dar.

**Funktion:** Die Visualisierung von Programm-Strukturen soll dem Leser einen schnellen einfachen Einblick in die Funktionen des Quellcodes ermöglichen. Hierzu muss jedes einzelne Quellcode-Dokument, das dem Projekt zugeordnet wird, ausgelesen und angemessen dargestellt werden.

Die Anzeige einzelner oder auch mehrerer Methoden, bis hin zum gesamten Inhalt des Quellcodes, liefern dem Leser einen schnellen Einblick in die Inhalte des damit in Zusammenhang stehenden Quellcode-Dokumentes.

Neben der reinen Anzeige von Inhalten erleichtert diese Art der direkten Visualisierung die Einsicht von Kommentierungen und der Versionsnummer des gewählten Quellcode-Dokumentes, sofern diese nicht bereits in separaten Strukturen abgebildet werden. Sofern Backward-Tracking, die Zuordnung von Quellcode zur Software-Spezifikation, realisiert wurde, kann diese Verknüpfungsinformation verwendet werden, um die betroffenen Anforderungen schnell zu identifizieren.

**Beurteilung:** Es wird validiert, ob die Programm-Strukturen des Quellcodes abgebildet werden können.

**Beantwortung:**

| Fragen | Antwort | Wertung |
|---|---|---|
| Können Programmstrukturen abgebildet werden. | Ja | 5 Punkte |
| Können Programmstrukturen abgebildet werden. | Nein | 0 Punkte |
| | Ergebnis (Max. 5 P.) | |

## Kann Backward-Tracking abgebildet werden? (#04-03)

**Beschreibung:** Dieses Qualitätsmerkmal bewertet die Möglichkeit, Backward-Tracking abbilden zu können. Backward-Tracking stellt die Zuordnung von Quellcode und zu den formulierten Anforderungen in einer Software-Spezifikation dar.

**Funktion:** Damit eine Zuordnung zwischen den einzelnen Funktionen des Quellcodes und den Anforderungen der Software-Spezifikation realisiert werden kann, muss hierzu eine eindeutige Verknüpfung zwischen diesen Informationen formuliert werden (siehe Abbildung 45).

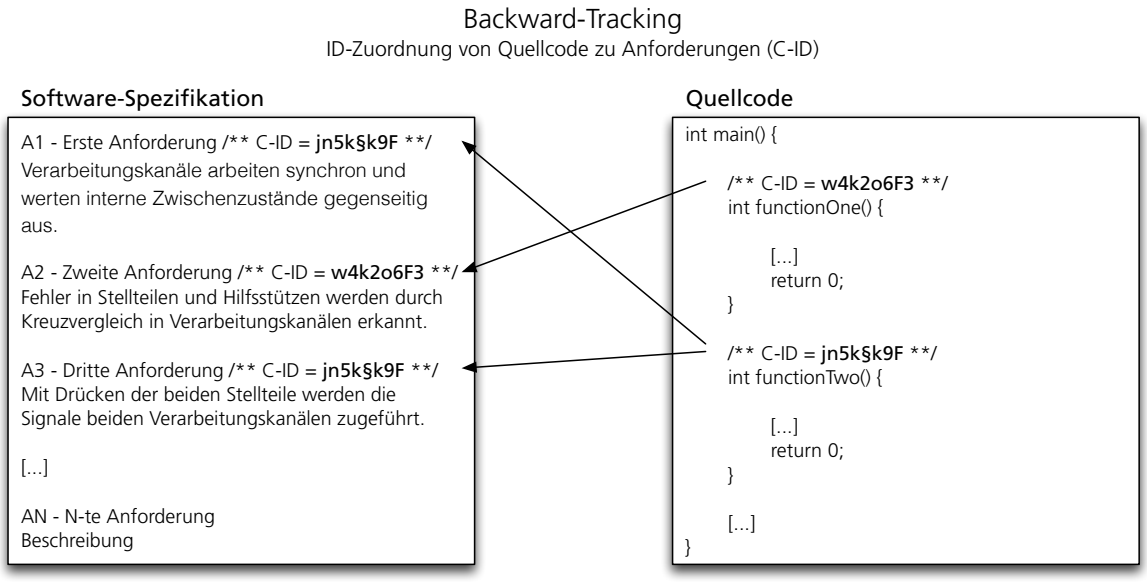



Abb. 45: Backward-Tracking mit Zuordnung vom Quellcode zu Anforderungen

Die Verknüpfung erfolgt in Form einer eindeutigen Identifikationsnummer (ID), die zu jeder Funktion des Quellcodes erstellt und dieser zugeordnet werden muss. Dies kann beispielsweise als ID-Eintrag im Kom-

mentarfeld zu jeder Funktion erfolgen. Eine zugeordnete ID darf nur ein einziges Mal im Quellcode enthalten sein. Abschließend muss die generierte ID der Funktion der zugehörigen Anforderung in der Software-Spezifikation zugeordnet werden.

Anhand dieser eindeutigen Merkmale kann eine schnelle visuelle Zuordnung zwischen einer Funktion im Quellcode und den dazu formulierten Anforderungen erfolgen.

**Beurteilung:** Es wird validiert, ob Backward-Tracking im Quellcode erfasst werden kann.

**Beantwortung:**

| Fragen | Antwort | Wertung |
|---|---|---|
| Kann Backward-Tracking erfasst werden? | Ja | 5 Punkte |
| Kann Backward-Tracking erfasst werden? | Nein | 0 Punkte |
| | **Ergebnis (Max. 5 P.)** | |

### Kann eine Code-Dokumentation abgebildet werden? (#04-04)

**Beschreibung:** Dieses Qualitätsmerkmal bewertet die Möglichkeit, der Abbildung einer Code-Dokumentation. Die Code-Dokumentation liefert eine Zusammenfassung aller kommentierten Funktionen durch den Programmierer in einem kompakten Dokument.

**Funktion:** Die Code-Dokumentation soll dem Leser eine ergänzende Beschreibung der einzelnen Funktionen im Quellcode liefern. Zu diesem Zweck muss zu jeder Funktion mindestens ein Kommentarblock vor der Funktion angelegt werden, mit dessen Hilfe die wichtigsten Merkmale der Funktion beschrieben werden können.

*Der Kommentar einer Funktion sollte wichtige Merkmale enthalten:*

- Funktionsbeschreibung in ausführlicher Form
- Autor der Funktion
- Version der Funktion
- Parameter der Funktion
- Rückgabewerte der Funktion
- Fehlerausgaben

Zu jeder einzelnen Funktion sollten mindestens die o. g. Merkmale detailliert aufgeführt werden. Auf Basis der hinterlegten Merkmale kann abschließend eine Dokumentation generiert werden, sodass der Leser ein kompaktes Begleitdokument für die Interpretation des Quellcodes zur Verfügung gestellt bekommt. Im Regelfall wird diese Dokumentation automatisiert im HTML-Format generiert.

Durch diese Dokumentation soll der Leser die Funktionsweise der ggf. komplexen und wenig transparenten Funktionen besser nachvollziehen können.

**Beurteilung:** Es wird validiert, ob eine Code-Dokumentation im Quellcode erfasst werden kann.

**Beantwortung:**

| Fragen | Antwort | Wertung |
|---|---|---|
| Kann die Code-Dokumentation vollständig abgebildet werden? | 100% | 5 Punkte |
| Kann die Code-Dokumentation überwiegend abgebildet werden? | 50% - 99% | 3 Punkte |
| Kann die Code-Dokumentation teilweise abgebildet werden? | 1% - 49% | 2 Punkte |
| Kann die Code-Dokumentation nicht abgebildet werden? | 0% | 0 Punkte |
| | **Ergebnis (Max. 5 P.)** | |

### Können verschiedene Funktionen gekapselt werden? (#04-05)

**Beschreibung:** Dieses Qualitätsmerkmal bewertet die Möglichkeit einer Trennung von verschiedenartigen Funktionen, wodurch eine Differenzierung in Bereiche einer Software erfolgen kann. Hierbei ist insbesondere die Trennung in sicherheitsrelevante und nicht sicherheitsrelevante Bereiche sinnvoll.

**Funktion:** Die Kapselung von Funktionen soll grundsätzlich in zwei Kategorien erfolgen, in nicht sicherheitsrelevante Funktionen und in sicherheitsrelevante Funktionen. Sicherheitsrelevante Funktionen stellen Methoden dar, die im Falle eines Fehlers zu einer negativen Auswirkung führen könnten, wie beispielsweise einem Personenschaden. Alle übrigen Methode bilden die Gruppe der nicht sicherheitsrelevanten Funktionen, bei denen es zu keiner negativen Auswirkung kommen kann.

Eine Trennung in diese beiden Bereiche ist notwendig, da für eine Prüfung lediglich sicherheitsrelevante Bereiche von Wichtigkeit sind. Aus diesem Grund muss zu jeder sicherheitsrelevanten Funktion zusätzlich das Performance Level (von Stufe a bis e) angegeben werden.

Durch die Trennung entsteht außerdem der Vorteil, dass die weniger wichtigen Funktionen (nicht sicherheitsrelevant) von wichtigen Funktionen (sicherheitsrelevant) klar getrennt werden und effizientere Überprüfung ermöglichen.

**Beurteilung:** Es wird validiert, ob Funktionen gekapselt abgebildet werden können.

**Beantwortung:**

| Fragen | Antwort | Wertung |
|---|---|---|
| Kann in nicht sicherheitsrelevanten Bereichen gekapselt werden? | Ja | 1 Punkt |
| Kann in sicherheitsrelevanten Bereichen gekapselt werden? | Ja | 2 Punkte |
| Können Funktionen voneinander gekapselt werden? | Ja | 2 Punkte |
| Können Funktionen voneinander gekapselt werden? | Nein | 0 Punkte |
| | **Ergebnis (Max. 5 P.)** | |

## 5.4 Dokumentformat zur Abbildung von Prüfdokumentationen

Ausgehend von den gesammelten Informationen und Erkenntnissen erfolgte ein Entwurf für ein Format zur Repräsentation von Informationen innerhalb von Software-Prüfprozessen. Dieser Entwurf sieht ein Container-Format vor, das es erlaubt, Informationen der Software-Prüfung strukturiert zu vereinen, indem es verschiedenste Dokumentarten aufnimmt, wie beispielsweise Spezifikationsdokumente oder Quellcode. Daneben soll das Container-Format zahlreiche Meta-Informationen vorhalten wie Datumsangaben, Projektzustände oder Versionen, durch die Änderungen an Dokumenten sichtbar und für den Prüfer einfacher nachvollziehbar werden.

Ein weiteres Ziel des Forschungsprojektes bestand in der Entwicklung eines konsistenten und qualitätsgesicherten Dokumentformats zur Abbildung aller verwendeten und erzeugten Informationen einer Spezifikation. Das entwickelte Format kann mittels geeigneter Werkzeuge zur Spezifikationsgenerierung eingesetzt werden, um die zugrundeliegende Dokumentarchitektur nutzbar zu machen.

Dieses Dokumentformat wurde für Inhalte von Spezifikationen entwickelt, welche sich an Normen zur Dokumentation von Spezifikation (bspw. der IEEE 930 und ISO 13849-1) orientieren, mit dem Ziel, Informationen für die Software-Prüfung in einem formalen Schema strukturiert abbilden zu können. Das aus dem Dokumentformat abgeleitete Spezifikationsdokument soll menschenlesbar und nicht änderbar sein, aber auch gleichzeitig durch versteckte Informationen angereichert werden können. Durch die versteckten Meta-Informationen soll es beispielsweise möglich sein Kommentarinformationen abbilden und Prüfern oder Herstellen geeignet visualisieren zu können.

### 5.4.1 Container-Format

Die Konzeption des Formates zur Verwaltung von Dokumenten gründet auf qualitativen Befragungen innerhalb eines Prüfinstitutes. Durch dieses Format können die verschiedenartigen Dokumenttypen innerhalb eines Prüfvorgangs in einer einheitlichen Struktur zusammengefasst werden.

Die Ergebnisse der Befragungen stellen hierbei die Grundlage für die Entwicklung eines geeigneten Container-Formates zur Dokumentverwaltung für die verschiedenen Dokumenttypen dar (siehe Abbildung 46), die innerhalb eines Software-Prüfvorgangs Verwendung finden.

Prüfer verwenden eine vereinheitlichte Ordnerstruktur zur Verwaltung aller anfallenden Dokumente einer Prüfung. Sofern ein Auftrag für eine Prüfung entsteht, wird vom Prüfer ein neuer Vorgang mit eindeutiger Vorgangsnummer für den Hersteller und die Prüfung angelegt und damit eine Zuordnung von Informationen zu diesem Prozess erfolgen kann.

Im Laufe der gesamten Prüfung werden unter der jeweiligen Vorgangsnummer alle Dokumente abgelegt, die im Verlauf vom Prüfer generiert oder vom Hersteller geliefert werden, sodass je Prüfprozess ein separater Ordner vorliegt. Dies können Spezifikationsdokumente, Quellcode, Prüfungsergebnisse oder andere verschiedene Dokumentarten sein, die in passende Ordner innerhalb der Container-Struktur aufgenommen werden. Eingehende Emails werden beispielsweise mit den darin enthaltenen Anhängen im

Unterordner Emails im Schriftverkehr abgelegt, sodass jederzeit nachvollziehbar wird, welche Informationen zwischen Hersteller und Prüfer ausgetauscht wurden.

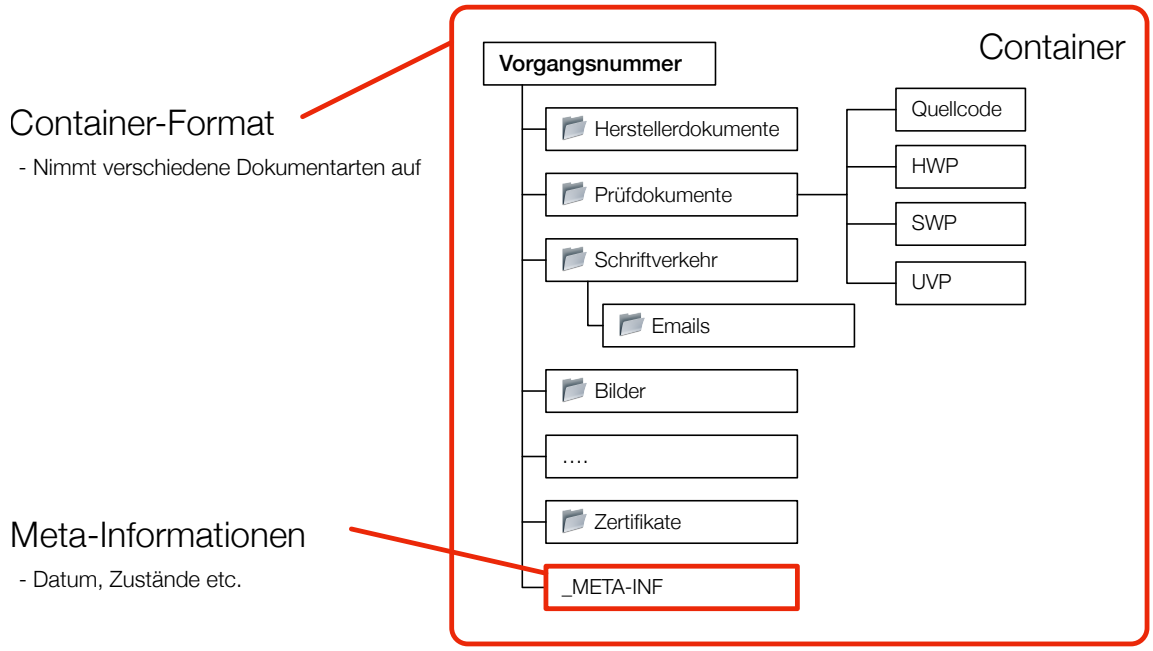



Abb. 46: Container zum Austausch von Inhalten (Import/Export)

Mit der Erstellung eines neues Projektes wird diese Verwaltungsstruktur angelegt und dem Prüfer bereitgestellt, sodass in dieser die meistern Dokumente verwaltet werden können. Damit zusätzlich Flexibilität gewährleistet werden kann, ist eine strukturelle Aufteilung in weitere Unterordner möglich. Diese Aufteilung ist für den Prüfer variabel und kann passend zum jeweiligen Inhalt eines Projektordners erstellt werden.

Alle Dokumente zu einem Prüfvorgang (mit eindeutiger Nummer) werden in diese Ordnerstruktur eingepflegt. Die Inhalte der Ordner werden über passende Meta-Informationen automatisiert verwaltet, die unter anderem eine Liste der Dokumente und Ordner innerhalb eines Ordners bereitstellen. Diese Meta-Dateien (_META-INF) beinhalten zusätzliche Informationen zu einem Ordner oder einer Datei in Objektform, wie beispielsweise relative Pfadangaben, Dokumentstatus, Dateiversionen und weitere, die auf einer reinen physischen Dateiebene nicht bereitgestellt werden können. Dadurch werden die Änderungen an Dokumenten sichtbar und für den Prüfer einfacher nachvollziehbar. Auf Basis dieser Meta-Informationen erhält ein Prüfer einen erweiterten Einblick über die Inhalte zu einem laufenden Projekt, die im Repository (siehe Kapitel 5.5.2) verwaltet und über das User Interface (siehe Kapitel 5.5.3) visualisiert werden. Durch die Verwaltung von Dokumentinformationen über Meta-Dateien, entsteht zusätzlich eine effiziente Möglichkeit Fakten zu abgelegten Daten abzurufen, ohne die Daten selbst aufzurufen, da hierfür nur die Meta-Daten verarbeitet werden müssen.

Weiterhin konnte durch weitere qualitative Befragungen verifiziert werden, dass diese Struktur des Container-Formates in seiner Grundform eingesetzt und zu jeder Prüfung alle anfallenden Dokumente unter einer eindeutigen Vorgangsnummer (VE-Nummer) subsumiert werden.

Eine weitere Funktionalität des Container-Formates besteht in der Unterstützung von Export- und Import-Aufgaben auf die Benutzer im Verlauf eines Prüfprozesse häufiger zurückgreifen, mit dem Ziel Prüfungsinformationen (Dokumente, Quellcode, etc.) auszutauschen. Für diese Aufgaben wurde ein prototypisches Werkzeug realisiert, welches in siehe Kapitel 5.5.4 näher beschreiben wird.

## 5.4.2 Dokumentformat

Nach der Konzeption des Qualitätsindex, den damit verbundenen Kriterien und den durchgeführten Befragungen wurde ein Entwurf für ein formales Dokumentmodell konzipiert. Dieses Format soll sämtliche Eigenschaften und Inhalte eines Prüfvorgangs von Software in einem einzelnen Dokumentmodell abbilden.

Grundsätzlich muss das Dokumentformat drei verschiedene Funktionsmöglichkeiten umsetzen. Zum einen sollte das Dokumentformat eine menschenlesbarer Form unterstützen, zum anderen sollte das Format maschinenlesbar sein, sodass der Vorteil automatisierter Vorgänge in Prüfprozessen ersichtlich wird. Weiterhin ist die Bereitstellung der Container-Informationen von hoher Bedeutung, da sie für die Prüfung notwendige Dokumente enthalten.

Basierend auf den bisher erarbeiteten Informationen wird ein geeignetes Format konzipiert, das innerhalb der Erstellungs- und Bearbeitungsphase einer Software-Spezifikation von Herstellern und Prüfern Verwendung findet. Durch dieses Format werden die verschiedenartigen Informationen einer Software-Spezifikation in einer einheitlichen Struktur zusammengefasst.

Das Dokumentformat soll dem Nutzer eine flexible Handhabung gewährleisten. Dabei gehört die Unveränderlichkeit zu den Hauptaspekten zur Erreichung eines flexiblen Dokumentformats. Dies gilt für unbeabsichtigte Veränderungen, wie beispielsweise die automatischen Aktualisierungen, von Datumsfeldern oder ungewolltes Verschieben von Textblöcken. Weiterhin sollte die Print-Möglichkeit unterstützt werden, damit Benutzer eine ausgedruckte Form (z.B. Ausgabe auf DinA4-Papier) erhalten können.

Aus den vorgestellten Kriterien an ein Dokumentformat kann die wesentliche Herausforderung abgeleitet werden, ein neues Format zu entwerfen, das sowohl die Eigenschaft der Menschen- als auch Maschinenlesbarkeit erfüllt. Zur Ermittlung der besten Eigenschaften erfolgt eine Evaluation für eine geeignete Struktur dieses Formats.

### 5.4.2.1 Evaluation hybrider Spezifikationsformate

Maschinenlesbare Formate zeichnen sich durch einen strukturierten Aufbau aus, bei dem jedes Element eines Spezifikationsdokuments (z.B. eine Überschrift, ein Textabschnitt, eine Zelle einer Tabelle, etc.) eindeutig identifizierbar und damit auch einzeln auslesbar ist. Darüber hinaus stehen sämtliche Elemente in

Beziehung zueinander. So besteht jedes Kapitel aus einer Überschrift und einer Menge von zugeordneten Texten, Grafiken und Tabellen.

Eine Spezifikation in menschenlesbarem Format zeichnet sich hingegen durch eine verständnisfördernde Textformatierung (z.B. Texte werden kleiner dargestellt als Überschriften) aus, in Form eines auf gängigen Papierformaten druckfähigen Dokuments. Die Zusammenhänge zwischen Kapitelüberschriften und Elementen werden in erster Linie visuell erzeugt.

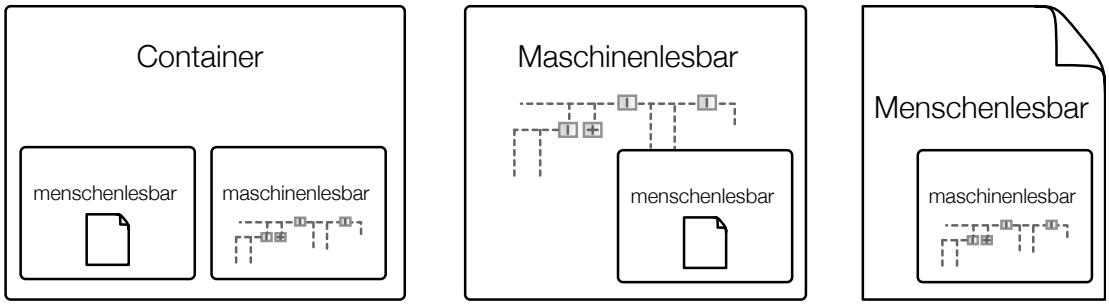



Abb. 47: Vorgehensweisen zu Erzeugung hybrider Spezifikationsformate

Um diese beiden Formate, die auf den ersten Blick gegensätzlich erscheinen, in einem neuen Formattyp miteinander zu vereinen, lassen sich drei mögliche Vorgehensweisen identifizieren, die in Abbildung 47 vorgestellt werden.

Ein menschenlesbares und maschinenlesbares Format, das diese beiden Anforderungen erfüllen kann, lässt sich prinzipiell durch diese verschiedenen Vorgehensweisen erschaffen:

1. Ein neutrales Containerformat nimmt sowohl eine maschinenlesbare, als auch eine menschenlesbare Version mit identischem Inhalt auf.

2. Ein maschinenlesbares Format nimmt die Binärdaten eines menschenlesbaren Formats auf. Hierbei müsste der menschenlesbare Anteil beim Öffnen des Dokuments extrahiert und an eine geeignete Anzeige-Software (PDF-Viewer, Bildbetrachter, Textverarbeitung, etc.) weitergegeben werden.

3. Ein menschenlesbares Format nimmt ein maschinenlesbares Format auf. Dies könnte z.B. in einem unsichtbaren Bereich geschehen.

Aus Sicht der einfachen Benutzbarkeit kommt insbesondere die 3. Vorgehensweise in Frage, da sie ein menschenlesbares Format erfordert. Dieses kann vom Nutzer mit einer gängigen und für verschiedene Plattformen verfügbaren Software unmittelbar und ohne Einsatz spezieller Tools geöffnet und betrachtet werden. Darin enthaltene Informationen, die eine Maschinenlesbarkeit ermöglichen, bleiben dabei zunächst für den Benutzer verborgen. Die versteckten Informationen können mit Hilfe spezieller Software extrahiert und betrachtet werden. Eine derartige Vorgehensweise stellt also eine Spezifikation in zunächst menschenlesbarer Variante zur Verfügung und bringt einen (unsichtbaren) maschinenlesbaren Anteil mit.

#### 5.4.2.2 Hybrides Spezifikationsformat

Ausgehend von der Evaluation möglicher Vorgehensweisen bei der Erstellung eines konkreten Dokumentformats liegt im Ergebnis der Konzeptphase ein Zielformat für eine Spezifikation vor. Die wesentliche Herausforderung besteht darin ein konkretes Format zu entwickeln, das sowohl menschen- als auch maschinenlesbar und gleichzeitig dokumentbasiert ist.

Zu diesem Zweck wurden zunächst zwei voneinander unabhängige Formate entwickelt, wobei eines die Anforderungen an ein maschinenlesbares Format und das andere die Anforderungen an ein menschenlesbares Format erfüllt. Im Anschluss wurde eine Methode entwickelt, die eine Verknüpfung beider Dokumentformate zur Sicherstellung der Inhalte beider Dokumente durchführt.

##### Ausgangsbasis: Spezifikationsschema

Zunächst muss eine klare Struktur der möglichen Inhalte einer Spezifikation konzipiert werden. Diese Entwicklung erfolgt auf Basis der Spezifikationsgrundlagen (siehe Kapitel 5.1.1.2) anhand von Vorschlägen der IEEE 830 1998 sowie der ISO 13849 und wird als XML Schema Definition (XSD)-Schema in ein passendes Dokument überführt. Abbildung 48 visualisiert die konzipierte Grundstruktur des XSD-Schemas und zeigt alle wesentlichen Bestandteile zur Abbildung einer Spezifikation in einer reduzierten Ansicht, wie an der Zeilennummerierung zu erkennen ist.

```
1   <?xml version="1.0" encoding="UTF-8"?>
2   <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
3     <xs:element name="specification">
4       <xs:complexType>
5         <xs:sequence>
6           <xs:element name="document-control"> [...] </xs:element>
72          <xs:element name="introduction"> [...] </xs:element>
186         <xs:element name="overall-description"> [...] </xs:element>
379         <xs:element name="specific-requirements"> [...] </xs:element>
740         <xs:element name="supporting-information"> [...] </xs:element>
802       </xs:sequence>
803       <xs:attribute name="projectName" type="xs:string"/>
804       <xs:attribute name="fileName" type="xs:string"/>
          [...]
814     </xs:complexType>
815   </xs:element>
      [...]
840 </xs:schema>
```

Abb. 48: Auszug XSD-Schema für eine Software-Spezifikation

Die einzelnen Elemente einer Spezifikation werden durch einen komplexen Datentyp verwaltet, mit dessen Hilfe die unterschiedlichen Kapitel zusammengefasst werden, wie beispielsweise die Einleitung ("introduction") oder die allgemeine Beschreibung ("overall-description"). Ebenso werden verschiedene be-

schreibende Attribute bereitgestellt, mit denen projektspezifische Eigenschaften (Projekt-, Dateiname, etc.) definiert werden können. Eine ausführliche Ansicht über das XSD-Schema ist in Anhang 7.1 aufzufinden.

Das XSD-Schema stellt den Ausgangspunkt für sämtliche Dokumentarten dar, die in SIWOB eine Spezifikation aufnehmen oder wiedergeben sollen. Das dabei entwickelte XSD-Dokument definiert für sämtliche Inhalte einer Spezifikation genaue Eigenschaften, wie beispielsweise die minimale Anzahl von funktionalen Anforderungen. Eine konkrete Spezifikation (z.B. einer Maschine) kann gegen ein derartiges Schema validiert werden, sodass bei einer erfolgreichen Prüfung belegt werden kann, dass das vorliegende Schema im formalen Aufbau gültig ist. Dies bedeutet allerdings nicht zwingend, dass eine solche Spezifikation auch inhaltlich valide ist. Die Inhalte einer Spezifikation sind nach wie vor das Produkt einer kreativen, menschlichen Leistung.

### Maschinenlesbares Spezifikationsformat

Die Validierung von XML-Dokumenten gegen ein XSD-Schema lässt sich mittels frei verfügbarer Software automatisiert, zuverlässig und einfach durchführen.[207] Aus diesem Grund ist es naheliegend, dass die Wahl des maschinenlesbaren Spezifikationsformates für das SIWOB-Projekt auf das XML-Format fällt. Konkret soll eine Spezifikation, die innerhalb von SIWOB verwendet wird, in Form von XML unterstützt werden und entspricht daher in Umfang und Struktur genau dem oben genannten XSD-Dokument. Das aus dem XSD-Schema abgeleitete XML-Dokument wird in Abbildung 49 visualisiert und stellt hierbei einen Auszug des gesamten Dokumentes dar, welches in einem ausführlichen XML-Schema in Anhang 7.2 aufzufinden ist.

```
1   <?xml version="1.0" encoding="UTF-8"?>
2   <ns2:specification xmlns:ns2="http://org.hbrs.siwob.types/">
3       <document-control> [...] </document-control>
19      <introduction> [...] </introduction>
41      <overall-description> [...] </overall-description>
87      <specific-requirements> [...] </specific-requirements>
147     <supporting-information> [...] </supporting-information>
162 </ns2:specification>
```

Abb. 49: Auszug XML-Struktur für eine Software-Spezifikation

Die XML-Struktur stellt hierbei eine abgeleitete Instanz des zuvor vorgestellten XSD-Schemas dar. Die darin spezifizierten Strukturen (komplexe Datentypen, Elemente, Attribute, etc.) werden in ein XML-Dokument überführt. In dieser Dokumentstruktur sind die Elementinformationen, wie beispielsweise die Einlei-

207. vgl. (W3Schools·(Hrsg.),·2014b)

tung oder allgemeine Beschreibung einer Spezifikation, aufzufinden und dienen als Basisgerüst für eine maschinenlesbare Auswertung.

Aus dem konzipierten XSD-Schema wird durch die Unterstützung einer Java-Klasse, ein Java-Objekt für eine Spezifikation abgeleitet, welches für die interne Verarbeitung von Spezifikationsinformationen genutzt werden kann, beispielsweise zur Anreicherung von Daten über das User Interface. Ebenso kann das Java-Objekt für die Visualisierung der Spezifikation im User Interface eingesetzt werden.

### Menschenlesbares Format

Eine Evaluation menschenlesbarer Dokumentformate erfolgt über die jeweils weit verbreiteten und bei Nutzern bekannten Formate DOCX (MS Word), ODF (OpenOffice / LibreOffice), PDF (Adobe Acrobat / PDF-Viewer) und JPEG, wie in Tabelle 11 dargestellt. Die aufgeführten Kriterien spiegeln dabei mögliche Anforderungen an das Dokumentformat wieder und werden im Folgenden genauer vorgestellt.

Spezifikationsdokumente sollen nach ihrer Erzeugung insbesondere für Prüfer unveränderbar sein. Dies gilt vor allem für versehentliche Veränderungen, wie z.B. automatische Aktualisierungen von Datumsfeldern oder versehentliches Verschieben von markierten Textblöcken. Die Integrität eines Dokuments kann unabhängig davon durch geeignete Methoden (Hash-Werte) sichergestellt werden.

| | DOCX | ODF | PDF | JPEG |
|---|---|---|---|---|
| Verbreitungsgrad | sehr hoch | mittel | sehr hoch | sehr hoch |
| Unveränderbare Dokumente möglich? | möglich | möglich | ausschließlich | ausschließlich |
| Bündelung von Seiten zu Dokument? | JA | JA | JA | NEIN |
| Grafiken und Text darstellbar? | JA | JA | JA | eingeschränkt |
| Textsegmente selektierbar? | JA | JA | JA | NEIN |
| Dateianhänge in Dokumenten möglich? | JA | JA | JA | NEIN |
| Druckverbindliche Formatierung? | eingeschränkt | eingeschränkt | JA | JA |

Tabelle 11: Übersicht verbreiteter menschenlesbarer Dateiformate

Spezifikationen sollen außerdem sämtliche Informationen in druckfähiger Form (z.B. Ausgabe auf DIN A4-Papier), also aufgeteilt auf einzelne Seiten und zusammengefasst in einem einzelnen Containerdokument ermöglichen. Zur Vereinfachung einer Prüfung eines solchen Dokuments sollte es möglich sein, einzelne Textstellen selektieren und kopieren zu können. Wesentlich für die Eignung als SIWOB-Format ist darüber hinaus die Möglichkeit, Dateianhänge (u.A. den maschinenlesbaren Anteil) zuzulassen.

Auf Basis dieser Kriterien stellt ein PDF-Dokument die passende Wahl für ein Format zur Repräsentation menschenlesbarer Informationen dar. Deshalb erfolgt die Entwicklung für die menschenlesbare Version eines Spezifikationsformates durch dieses Format.

### Kombination beider Formatanteile

Wie in den vorangegangenen Abschnitten erläutert, wird ein hybrides Spezifikationsformat für die Nutzung in SIWOB entworfen, das einen menschenlesbaren und einen maschinenlesbaren Anteil enthält. Bei diesem Format liegt der maschinenlesbare Teil in Form eines validierbaren XML-Dokuments und der menschenlesbare Teil des Dokuments in Form des standardisierten PDF-Formats vor. Sämtliche Informationen einer Spezifikation werden in beiden Formaten (XML und PDF) getrennt voneinander abgelegt. Abbildung 50 vermittelt einen Eindruck vom Ablauf, welcher zur Generierung des SIWOB Hybrid-Formates durchlaufen wird.

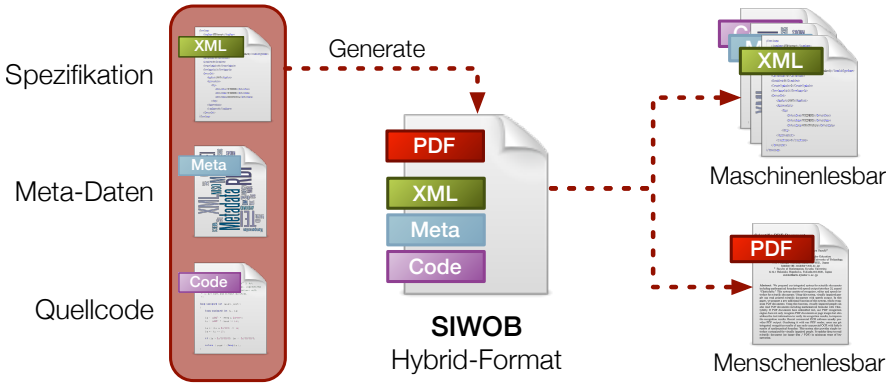



Abb. 50: Hybrid-Format mit Informationen für die Generierung eines SIWOB-PDF

Zunächst stehen Spezifikation, Meta-Daten und Quellcode zur weiteren Verarbeitung bereit. Diese verschiedenartigen Informationen resultieren aus den Eingaben der Benutzer, die über das bereitgestellte User Interface von SIWOB eingeben werden.

Die einzelnen Informationen einer Spezifikation stellen die Basis der Verarbeitung dar und werden durch die beschriebenen XML-Struktur (oder als Java-Objekt) bereitgestellt. Daneben existieren Meta-Daten (in XML-Form), die zusätzliche Informationen anbieten, die im Zusammenhang mit der erstellten Spezifikation nutzbringend erscheinen, wie beispielsweise Kommentierungen einzelner Spezifikationselemente. Der letzte Bereich wird durch Quellcode repräsentiert, der in einer engen Relation zur Spezifikation steht, da dieser mit den Anforderungen der Software verbunden ist.

Diese Dokumente, die auf unterschiedlichen Formaten basieren, stellen die Grundlage des Hybrid-Formates dar, wobei Meta-Daten und Quellcode-Dokumente einen optionalen Bereich bilden und nicht zwingend für die Generierung vorhanden sein müssen. Aus den vorliegenden XML-Informationen der Spezifikation wird anschließend das SIWOB Hybrid-Format gebildet, welches ein PDF-Dokument darstellt und die maschinenlesbaren Inhalte (XML-, Meta-, Quellcode-Dokumente) in einen nicht sichtbaren Bereich integriert.

Nach der Erstellung eines PDF-Dokumentes, gemäß der Spezifikation des SIWOB Hybrid-Formates, besteht die Möglichkeit, den menschenlesbaren Anteil ohne aufwendigen Werkzeugeinsatz einzusehen. Hierzu wird das Dokument mit einem PDF-Viewer (bspw. Adobe Acrobat Reader) geöffnet und gelesen. Weiterhin kann das resultierende PDF-Dokument von einem passenden Interpreter in maschinenlesbarer

Form ausgewertet werden. Die Interpretation dieser Inhalte erfolgt durch passende Werkzeuge, die über das User Interface von SIWOB den Benutzern bereitgestellt werden.

Hinzu kommt die optionale Bereitstellung von Quellcode und Meta-Daten innerhalb des maschinenlesbaren Formates. Hierdurch können neue Informationen mit Unterstützung zusätzlicher Werkzeuge erstellt und wieder in das maschinenlesbare Format eingebunden werden, wie beispielsweise Kommentare zu Teilbereichen der Spezifikation.

## 5.5 System-Ergebnisse

In diesem Kapitel werden die erreichten System-Ergebnisse des Forschungsprojektes vorgestellt, die im Projektverlauf prototypisch realisiert wurden. Abbildung 51 visualisiert in diesem Zusammenhang die konzipierte Gesamtarchitektur der Workbench und wird im folgendem Abschnitt näher erläutert.

Das System umfasst hierbei eine Vielzahl von prototypisch erstellten Komponenten (in grün hervorgehobene Werkzeuge), die eine grundsätzliche Arbeit mit dem System erlauben und damit eine Möglichkeit bereitstellen sollen einen Software-Prüfprozess zu unterstützen.

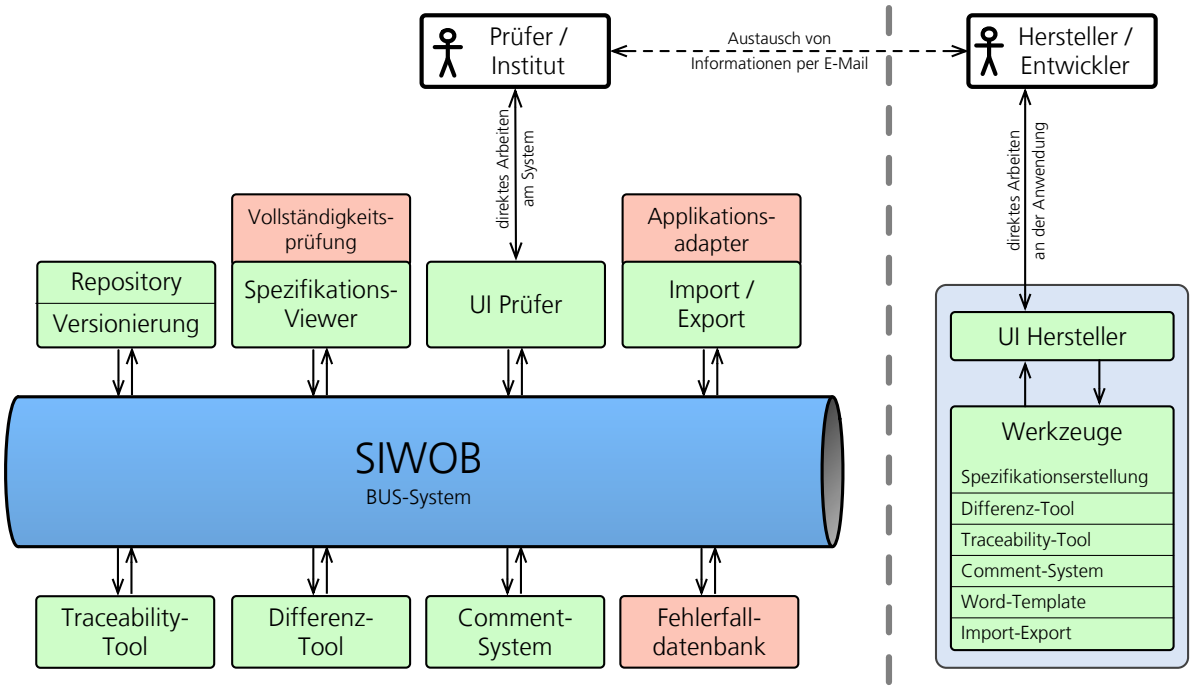



Abb. 51: Übersicht SIWOB BUS-System mit Werkzeugen

Die Kommunikation der prototypischen Komponenten wird über ein zentrales BUS-System realisiert, welches die fundamentale Infrastruktur des Gesamtsystems darstellt. Das BUS-System stellt, vereinfacht ausgedrückt, eine Schnittstelle zwischen den verschiedenen Komponenten dar, sodass eine Kommunikation unter den Komponenten und eine Bedienung durch Benutzer (Prüfer oder Hersteller) möglich wird. Zu den prototypischen Grundkomponenten zählen beispielsweise das Repository mit Zusatz eines Moduls zur Versionierung, welches für die Datenhaltung verantwortlich ist. Oder das User Interface, welches

dem Prüfer eine Schnittstelle zur Interaktion mit dem Gesamtsystem anbietet. Weiterhin wird das Grundsystem durch die Anbindung des Spezifikations-Viewers, welcher das hybride Spezifikationsformat des SIWOB-PDF darstellen kann sowie dem Import/ Export Werkzeug vervollständigt.

Das Grundsystem wurde durch zusätzliche Werkzeuge erweitert, zu denen beispielsweise das Traceability-Tool zählt, das eine Verknüpfung von Quellcode mit den Anforderungen einer Software-Spezifikation, ermöglichen soll. Weiterhin erlaubt das Differenz-Tool eine Visualisierung von Änderungen innerhalb einer Datei zwischen verschiedenen Dateiversionen. Mit Hilfe des Comment-System besteht zusätzlich die Möglichkeit zu einzelnen Dateien fehlerhafte Inhalte, Anmerkungen, oder Fragen über unklare Formulierungen in Kommentarform dokumentieren zu können.

Des Weiteren wird in der Abbildung eine Ansicht für die Seite des Herstellers visualisiert. Dieser kann grundsätzlich auf ein Repertoire von Werkzeugen mit ähnlichen Funktionsumfang zurückgreifen, wie diese auch dem Prüfern bereitgestellt werden. Dementsprechend unterscheidet sich die Anwendungsbereitstellung für Herstellers nur in wenigen Punkten von der Anwendungssicht eines Prüfers. Die grundlegenden Unterschiede bestehen hier aus Werkzeugen, die den Hersteller bei der Erstellung einer Spezifikation unterstützen sollen, wie das Werkzeug zur Spezifikationserstellung oder das Word Spezifikationsvorlage.

Auf Grundlage der Befragungen wurde für die Kommunikation zwischen Prüfer und Hersteller und den Austausch von Informationen, wie beispielsweise Prüfdokumente, ein E-Mail-basierter Verkehr angenommen. Diese Art der Kommunikation impliziert, dass ein paralleles Arbeiten von Prüfer und Hersteller möglich wird. In der Folge greifen Prüfer und Hersteller über zwei unterschiedliche Endpunkte auf verschiedene Systeme zu.

Neben den prototypisch realisierten Komponenten bestehen Ideen zu weiteren Werkzeugen (in rot hervorgehobene Werkzeuge), die sich im Rahmen einer Software-Prüfung als nutzbringend herausstellen könnten. Hierzu zählen insbesondere die formale Vollständigkeitsprüfung von Spezifikationen, die bereits bei der Erstellung einer Spezifikation auf der Herstellerseite eingesetzt werden könnten. Einen weiteren Aspekt stellt eine Fehlerfalldatenbank dar, in der Fehlerfälle von Prüfern oder Herstellern gespeichert und verwaltet werden könnten. Eine derartige Datenbasis könnte dazu verwendet werden, um passende Fehlerfälle für ausstehende Produktprüfungen generieren oder dokumentieren zu können. Eine abschließende Erweiterung betrifft die Import/Export Komponente durch Application-Adapter. Adapter sollen es ermöglichen unbekannte Dokumentformate importieren und auswerten, beziehungsweise in das Zielformat exportieren zu können, wie beispielsweise eine erstellte Spezifikation mit dem Word Spezifikationsvorlage. Dadurch sollen Prüfer Zugriff auf zusätzliche Informationen erhalten, die ohne passenden Adapter nicht zugänglich wären.

Eine kompakte Übersicht der soeben beschriebenen Gesamtarchitektur kann in der nachfolgenden Tabelle 12 eingesehen werden. Diese Tabelle beinhaltet eine kompakte Zusammenfassung der Ergebnisse und der Vorteile für jedes konzipierte Werkzeug der SIWOB Workbench.

**Bus-System (siehe Kapitel 5.5.1)**

Ergebnisse:
- Intranet-Applikation als Zentrale Infrastruktur
- Transparentes Verhalten des Bus-Systems (Bedienung über konzipierte Web-Oberfläche)
- Verwendung etablierter Kommunikationsstandards (SOAP/HTTP)
- Schnittstelle zwischen den konzipierten Werkzeugen, sodass eine Kommunikation unter den diesen möglich wird (als Vermittler-Komponente der Safety Inspection Workbench)
- Bedienung durch Prüfer und Hersteller möglich
- Auswahl eines Enterprise Service Busses durch Evaluation
- Prototypische Anforderungen als nicht-funktionale und funktionale Anforderung erfolgreich umgesetzt
- Skalierbareres System mit redundanten Ausfallsicherheitsmechanismen (Clustering)

Vorteile:
- Hersteller werden bei der Erzeugung von prüfungsgerechten Spezifikationen unterstützt
- Hersteller müssen vertrauliche Informationen nicht auf öffentlich zugänglichen IT-Infrastrukturen speichern
- Prüfvorgänge durch den Zugriff über ein internes BUS-System flexibel abrufen
- Erweiterte Anzeige- und Darstellungsmöglichkeiten von Informationszusammenhängen innerhalb oder zwischen einzelnen Prüfvorgängen
- Verwendung von Open-Source Lizenzen
- Einbindung zahlreicher zusätzlicher Schnittstellen (Kommunikation mit weiteren Werkzeugen)

**Repository (siehe Kapitel 5.5.2)**

Ergebnisse:
- Das Repository kann Dateien auf Dateisystem-Ebene verwalten und darüber hinaus umfangreiche Meta-Daten vorhalten und bereitstellen
- Eine Versionsverwaltung, innerhalb des Repositories, ermöglicht es, die verschiedenen Versionen einer Datei zu speichern, auszulesen und wiederherzustellen
- Entwicklung eines MetaDataContainer, welcher über eine Hash-Map verfügt, um die Korrektheit der Meta-Daten sicherzustellen

Vorteile:
- Eigene Konzeption und Entwicklung des Repositories, als Modul in SIWOB integriert, sodass individuelle Aufgaben eines Software-Prüfers unterstützt werden können
- Create-Read-Update-Delete (CRUD) stellen die Basisfunktionen dar, die eine einfache Verwaltung der Dokumente und ihrer Meta-Daten sicherstellen

**User-Interface (siehe Kapitel 5.5.3)**

Ergebnisse:
- Einheitliche Bedienoberfläche für Benutzer, welche gängige Web-Technologien verwendet
- Zugriff auf gesamte Funktionen von SIWOB, die durch die verschiedenen Werkzeuge angeboten werden
- Verwendung des Open Source Webframework Bootstrap, welches eine einheitliche Visualisierung ermöglicht, wodurch den Benutzern eine standardisierte Oberfläche zur Ausgabe im Web-Browser zur Verfügung gestellt wird
- Ausgabe für unterschiedliche Geräte-Kategorien: Smartphone, Tablet und dem Desktop-PC mit kleiner sowie großer Anzeigefläche

Vorteile:
- Individuelles User-Interface mit Einflüssen des Open-Source Webframeworks Bootstrap wodurch, zum einen die Erwartungskonformität der Benutzer im Umgang mit der Web-Oberfläche erhöht und gleichzeitig der Wiedererkennungswert von gleichen Funktionen auf den einzelnen Web-Seiten verbessert wird
- Eingereichte Unterlagen verschiedener Dokumenttypen eines Hersteller können an einer zentralen Stelle eingesehen werden

**Import/Export (siehe Kapitel 5.5.4)**

Ergebnisse:
- Möglichkeit zum Austauschen von Dokumenten eines Prüfprozesses (Code, Spezifikationen, Abbildungen, etc.)
- Als einzelne Module im User-Interface integriert
- Das Export Tool übermittelt an einen passenden Service die getroffene Elementauswahl des Benutzers. Der angesprochene Service sammelt auf Basis der Selektionen die assoziierten Elemente im Repository, die in ein Container-Archiv überführt werden sollen
- generierte Container-Archive werden, unter der Vorgangsnummer des assoziierten Projektes, in einem passenden Ordner ("transfer/export") abgelegt
- generierte Container-Archive beinhalten eine zum assoziierten Projekt konforme Container-Struktur und können abschließend durch die angebotenen Funktionen heruntergeladen und lokal gespeichert werden
- Der Benutzer hat die Möglichkeit ein exportiertes Container-Archiv an einen anderen Benutzer zu übermitteln, beispielsweise als Email-Anhang
- Import Tool setzt für eine korrekte Verarbeitung ein Container-Archiv voraus, welches über das Export-Tool exportiert wurde
- Neben dem Import von Container-Archiven besteht die Möglichkeit einzelne Dateien zum Datenbestand eines Projektes hinzuzufügen
- Das Import Tool übermittelt an einen Service die getroffene Container-Auswahl und ermittelt die im Container-Archiv enthaltenen Informationen. Informationen, über Ordner und Dateien, berechnet der Service mit den äquivalenten Container-Strukturen im assoziierten Projekt

Vorteile:
- Container-Archive werden grundsätzlich im Zip-Format generiert und gespeichert
- Benutzer können diese Archive mit Hilfe von Betriebsystem-Bordmitteln oder passenden Programmen öffnen. Dadurch kann ein Nutzer ohne passendes Import Werkzeug an die Inhalte eines exportierten Container-Archivs gelangen
- Ein Projekt existiert mit Vorgangsnummer, damit eine Zuordnung für das Container-Archiv möglich ist. Dadurch entsteht weniger Aufwand bei der Integration einzelner Dateien, wie beispielsweise Dokumentationen oder Quellcode
- Integration in das Repository erfolgt durch die Betätigung eines Knopfes auf der Web-Oberfläche "In das Repository einpflegen"
- Sollte ein geöffnetes Container-Archiv bereits in das Projekt aufgenommen worden sein, ist eine erneute Integration nicht möglich und wird dem Benutzer entsprechenden visualisiert. Duplikat-Archive werden somit vermieden
- Änderungen innerhalb der Archive können durch Versionierung im Repository nachgehalten werden

**Differenzvisualisierung (siehe Kapitel 5.5.5)**

Ergebnisse:
- Individuelle Entwicklung eines Differenz-Tools in Form eines Prototypen
- Vergleich von stark strukturieren Dokumenten, wie beispielsweise Code-Strukturen
- Zeilenbasierter Vergleich bildet in dieser Entwicklungsphase des Prototypen den besten Kompromiss aus Effizienz und Realisierbarkeit
- Jedes Plain-Text Dokument kann für einen Vergleich verwendet werden, wodurch Dokumente mit Programm-Strukturen (C-Dateien etc.) und einfachen Textinformationen abgedeckt werden
- Als Ergebnis wird eine tabellarische Ansicht mit mehreren Spalten generiert, bei der die Zeilenzahlen der beiden Dokumente angezeigt werden
- Die Bildung einer Differenz zwischen zwei ausgewählten Dokumente erfolgt automatisiert

Vorteile
- Mit Unterstützung der Versionsverwaltung durch das Repository wird eine schnelle Differenzvisualisierung von geänderten Dateien ermöglicht
- Gängige Dateiformate miteinander vergleichbar, wie beispielsweise Text-oder Programmier-Dateien, sodass der Nutzer ohne aufwändige Vorbereitung eine Differenzbildung ausführen kann
- Das Ergebnis der Differenzberechnung kann innerhalb des User Interfaces für den Benutzer visualisiert werden

**Comment System (siehe Kapitel 5.5.6)**

Ergebnisse:
- Gliederung in zwei Teile:
- Teil 1: Web Service (Comment-System), der Kommentare entgegennimmt bzw. zurückgibt sowie die gesamte Modullogik des Kommentarsystems beinhaltet. Folgende Funktionen wurden dazu integriert:
  1. Eintragen von neuen Kommentaren
  2. Ausgeben der Kommentare zu einer Datei/Ordner
  3. Benutzer kann seine Kommentare verstecken/anzeigen
  4. Überprüfung, ob eine Datei neue Kommentare hat
  5. Import/Export Funktionen
  6. Änderungsfunktionen
- Teil 2: Ausgabe und Präsentation der Informationen im User Interface
- Mit Hilfe eines Buttons auf der Web-Oberfläche, für jede Datei und jeden Ordner, erhält ein Benutzer die Möglichkeit die Kommentaransicht zum entsprechenden Objekt zu öffnen.
- Eine weitere Funktion überprüft, ob neue Kommentare vorliegen und passt diese farblich zur Ausgabe an

Vorteile
- Möglichkeit zu jeder einzelnen Datei einer Baumusterprüfung Kommentare notieren zu können: Quellcode-Dateien oder die Systemspezifikation, aber auch Grafiken und Bauzeichnungen
- Verwendung eines rudimentären Datenbankmanagementsystem für die interne Verwaltung der Kommentare
- Es wird gewährleistet, dass im Fall eines Exportes von Dateien auch die Kommentare mit exportiert werden und genauso beim Import

**Spezifikationsgenerierung (siehe Kapitel 5.5.7)**

Ergebnisse:
- Erstellung eines hybriden Spezifikationsformats, dass menschenlesbar ist und in Form eines PDF-Dokumentes zur Verfügung gestellt werden kann sowie maschinenlesbar, indem es komplexe Informationen in einer XML-Struktur abbildet
- Für die Spezifikationserstellung wurde eine Schnittstelle für das User Interface konzipiert, sodass ein Hersteller eine intuitive Eingabe-Maske erhält
- Generierung einer Spezifikation im SIWOB-PDF Format
- Entwicklung auf Basis des Open-Source Werkzeugs iText 4 für die Erstellung, Verwaltung und Änderungen von PDF-Dokumenten
- Spezifikationsgenerierung über User-Interface möglich
- Spezifikationsgenerierung über Word Spezifikationsvorlage möglich

| | |
|---|---|
| Vorteile: | • Erstellung einer Spezifikation durch eine Vollständigkeitsprüfung unterstützt<br>• Ermöglicht dem Hersteller ein Dokument zu erstellen, welches formal vollständig ist und nimmt dem Prüfer diese Untersuchung der Vollständigkeit ab<br>• Für die Bearbeitung des SIWOB-PDF bestehen zwei Möglichkeiten:<br>1. Das SIWOB-PDF wird im User Interface geöffnet, wodurch eine vorhandene Spezifikation fertiggestellt oder überarbeitet werden kann, wobei die maschinenlesbaren Informationen der enthaltenen XML-Datei verwendet werden.<br>2. Das SIWOB-PDF-Dokument wird mit einem PDF-Viewer (wie z.B. Adobe Acrobat) geöffnet, wobei lediglich der menschenlesbare, formatierte Teil sichtbar wird. Des Weiteren kann ein Prüfer, ebenfalls via SIWOB User Interface, auf den maschinenlesbaren XML-Anteil eines SIWOB PDFs zugreifen |
| **Traceability (siehe Kapitel 5.5.8)** | |
| Ergebnisse: | • Unterstützung eines Prüfers bei der Auswertung von Quelltexten und der dazu passenden Software-Spezifikation<br>• Separate User-Interfaces für Prüfer und Hersteller<br>• Anforderungen können mit korrespondierendem Quellcode-Methoden im Sinne der IEC 61508-3 in Verbindung gebracht werden |
| Vorteile: | • Durch eine Zuordnung des Herstellers von Quellcode-Methoden und den Anforderungen aus der Spezifikation, wird dem Prüfer die eigentliche Analyse der Quelltexte und der Spezifikation erleichtert, da für diesen die notwendige Verknüpfungsarbeit entfällt.<br>• Ein Hersteller kann verknüpfte Dokumente bei einem Prüfer einreichen, worauf dieser ohne Zusatzaufwand die einzelnen Anforderungen der Software-Spezifikation hinsichtlich ihrer Konformität zum Quellcode prüfen und Fehler in Form von Anmerkungen notieren kann |
| **Word Spezifikationsvorlage (siehe Kapitel 5.5.9)** | |
| Ergebnisse: | • Prototypische Dokumentvorlage zu Erstellung von Software-Spezifikationen auf Basis der empfohlenen IEEE/ANSI 830-1998 Struktur entwickelt<br>• Der Benutzer kann mit Hilfe dieser Strukturvorlage alle Inhalte für eine Spezifikation passend eingliedern<br>• Optional können Inhalte in ein XML-Dokument als Ausgabeformat überführt und dieses für weitere Verarbeitungsschritte in der SIWOB-Workbench bereitgestellt werden |
| Vorteile: | • Durch die Formalisierung und Struktureingrenzung der Spezifikation können die Inhalte einer Spezifikation durch geeignete Funktionen überprüft werden<br>• Überprüfung der Benutzereingaben auf Vollständigkeit<br>• Überprüfung auf nicht vorhandene oder inkorrekte Inhalte<br>• Inhalte und Strukturen der Dokumentvorlage können in ein unveränderliches Ausgabeformat (PDF-Dokument) konvertiert und zur Prüfung eingereicht werden<br>• Erstellung einer Spezifikation über Microsoft Word Textverarbeitungsprogramm, als auch lokal über Browser möglich |

Tabelle 12: Kompakte Zusammenfassung der System-Ergebnisse

Die nachfolgenden Unterkapitel liefern eine ausführliche Beschreibung der erzielten Ergebnisse. Der Fokus der Entwicklung bestand in der Erstellung einer geeigneten Infrastruktur (BUS-System) und die Entwicklung, beziehungsweise Anbindung, von prototypischen Komponenten zur Erweiterung der Workbench.

## 5.5.1 BUS-System

Entwickelte Tools werden innerhalb eines erweiterbaren BUS-Systems in Form einer Intranet-Applikation zur Verfügung gestellt. Das BUS-System stellt dabei, als Technik einer Werkbank (als Vermittler-Komponente der Safety Inspection Workbench), die zentrale Infrastruktur dar, an die Komponenten gekoppelt und mit deren Hilfe Informationen zwischen den Komponenten ausgetauscht werden können. Auf diese Weise werden Hersteller bei der Erzeugung von prüfungsgerechten Spezifikationen unterstützt, müssen vertrauliche Informationen aber nicht auf öffentlich zugänglichen IT-Infrastrukturen speichern. Prüfinstitute können ihre Prüfvorgänge durch den Zugriff über ein internes BUS-System flexibel abrufen und profitieren von den erweiterten Anzeige- und Darstellungsmöglichkeiten von Informationszusammenhängen innerhalb oder zwischen einzelnen Prüfvorgängen.

### Anforderungen an das Bus-System

In diesem Kapitel werden die Anforderungen an das BUS-System vorgestellt und ihre Notwendigkeiten erläutert. Diese Anforderungen werden gemäß den Software Requirements in funktionale und nichtfunktionale Anforderungen klassifiziert.[208]

#### Funktionale Anforderungen

Die funktionalen Anforderungen legen nach standardisierten Ansätzen der Software-Entwicklung die einzelnen Funktionen des BUS-Systems fest und werden in diesem Kapitel für eine bessere Übersicht als Tabelle dargestellt.[209]

Die Funktionsnummer beinhaltet die eigentliche Nummer der Funktion sowie am Ende der Notation einen Buchstaben, welcher für die gegebene Eigenschaft der Funktion steht. So steht R für Routing, A für Ausfallsicherheit, V für Verarbeitung und U für Umgebung.

Die Funktionen werden in die vier oben genannten Eigenschaftsgruppen gegliedert, sodass diese funktionalen Anforderungen in die wichtigsten Attribute, welche für die Verknüpfung von Werkzeugen zur Unterstützung von Software-Prüfverfahren notwendig sind, unterteilt werden.

| Funktions-nummer: | Funktionale Anforderung: |
|---|---|
| F1R | Das BUS-System soll SOAP-Nachrichten der unterschiedlichen Prüfdienste empfangen können. |
| F2R | Das BUS-System soll SOAP-Nachrichten der unterschiedlichen Prüfdienste weiterleiten können. |
| F3R | Das BUS-System soll die einzelnen Prüfdienste identifizieren können. |
| F4R | Das BUS-System soll die Prüfdienste als Endpunkte innerhalb des Busses bereitstellen. |

208. vgl. (Schmidt, 2013, S. 43)

209. vgl. (Robertson & Robertson, 2006, S. 9f.)

| | |
|---|---|
| F5R | Das BUS-System soll bei Bedarf mehrere Nachrichten kombinieren und diese als neue Nachricht ausgeben. |
| F6R | Das BUS-System soll eine Kommunikation zwischen den Prüfdiensten ermöglichen. |
| F7A | Das BUS-System soll zusammen mit den Prüfdiensten als Cluster aufgestellt werden. |
| F8A | Das BUS-System soll bei Ausfall eines vorhandenen Moduls auf ein äquivalentes Modul wechseln. |
| F9A | Das BUS-System soll einen Fehlercode ausgeben, wenn der aktive Cluster-Knoten nicht mehr verfügbar ist. |
| F10A | Das BUS-System soll mehrere Nachrichten parallel verarbeiten können. |
| F11A | Das BUS-System soll bei Ausfall eines vorhandenes Systems auf ein äquivalentes Ziel wechseln. |
| F12A | Das BUS-System soll den Nutzer auf der Weboberfläche darauf hinweisen, wenn ein System ausfällt. |
| F13V | Das BUS-System soll entgegengenommene Daten zwischen Prüfdienst und Prüfer in das benötigte Format transformieren können. |
| F14V | Das BUS-System soll auf Exceptions mit dem vordefinierten Verhalten der Prüfdienste reagieren. |
| F15V | Das BUS-System soll die unterschiedlichen Funktionen der Prüfdienste in parallelen Threads ausführen. |
| F16V | Das BUS-System soll das vorhandene Repository der Prüfdienste adressieren. |
| F17V | Das BUS-System soll die vorhandenen XSD-Dateien der Prüfdienste nutzen können. |
| F18V | Das BUS-System soll entgegengenommene Nachrichten anreichern können. |
| F19V | Das BUS-System soll die Nachrichten der Prüfdienste auf Basis von vordefinierten Prozessketten abarbeiten. |
| F20U | Das BUS-System soll transparent für den Benutzer der Weboberfläche des Frontends sein. |
| F21U | Das BUS-System soll die Schnittstellen der Prüfdienste durch WSDL-Dateien integrieren können. |
| F22U | Das BUS-System soll analog zu den Prüfdiensten in das Frontend eingebunden werden. |

Tabelle 13: Funktionale Anforderungen an das BUS-System

### Nichtfunktionale Anforderungen

Im Zuge der prototypischen Konzeption und Entwicklung bestimmen eine Reihe von nichtfunktionalen Anforderungen die Eigenschaften des BUS-Systems.[210] In diesem Teil werden nur Anforderungen (zehn) formuliert, die für dieses Projekt von entscheidender Bedeutung sind.

210. vgl. (Robertson & Robertson, 2006, S. 9f)

### 1. Open Source

Das BUS-System soll eigenständig entwickelt werden oder mit der Unterstützung vorhandener Open Source Software. Open Source Software kann kostenlos genutzt werden und darüber hinaus weitergeteilt oder verändert werden.[211] Diese Eigenschaft ist zwingend erforderlich, denn im Rahmen des vorhandenen Forschungsprojektes soll keine Software zum Einsatz kommen, welche einer kommerziellen Lizenz unterliegt.

### 2. Hochverfügbarkeit

Das BUS-System und auch die Werkzeuge zur Unterstützung der Software-Prüfverfahren müssen jederzeit für den Benutzer verfügbar sein.

Insbesondere Anwendungen, die mit einem Enterprise Service Bus kommunizieren, bestehen oft aus vielen unterschiedlichen Komponenten, welche eine Vielzahl an Fehlern erzeugen können. Auch der ESB kann auf viele Komponenten oder Module zurückgreifen, die Fehler enthalten oder erzeugen können, wodurch die hohe Verfügbarkeit des Systems beeinträchtigt werden kann. Es muss also jeder einzelne Fehler berücksichtigt werden, der die Verfügbarkeit des Systems beeinträchtigen kann. Hochverfügbar sind Systeme, die einen hohen Grad an Betriebskontinuität innerhalb eines bestimmten Zeitrahmens gewährleisten müssen.[212]

### 3. Skalierbarkeit

Im Rahmen eines aktuellen Forschungsprojektes werden stetig weitere Werkzeuge entwickelt oder vorhandene Werkzeuge weiterentwickelt. Dies kann dazu führen, dass kurzfristig deutlich mehr Prüfer auf vorhandene oder neue Prüfdienste zu greifen und somit die Last unter der Verarbeitung von Nachrichten im Enterprise Service Bus steigt. Des Weiteren kann das Hinzufügen neuer Prüfwerkzeuge zusätzliche Systeme erfordern, welche das BUS-System erweitern können. Daher muss auch die Skalierbarkeit des Enterprise Service Busses als nichtfunktionale Anforderung berücksichtigt werden, um Prüfer und Prüfdiensten weiterhin eine zuverlässige Kommunikation zu garantieren.[213]

### 4. Flexibilität

Ein weitere wichtige Anforderung an das BUS-System ist die Eigenschaft Flexibilität. Im Absatz Skalierbarkeit wurde bereits beschrieben, dass Prüfdienste weiterentwickelt werden oder neue Dienste hinzu kommen. Nicht nur die Skalierbarkeit des BUS-System spielt dabei eine große Rolle, sondern auch die Flexibilität des ESBs in Form von Unterstützung vorhandener Protokoll-Standards. Dazu gehören unter anderem Protokolle für die Kommunikationskomponenten innerhalb des ESB, wie zum Beispiel Protokolle für das Routing, die Transformation oder die Fehlerbehandlung.[214]

---

211. vgl. (Opensource.org (Hrsg.), 2014)
212. vgl. (Mike Somekh, Mark Foster, Rastislav Kanocz, 2009, S. 2)
213. vgl. (Bundesamt für Sicherheit in der Informationstechnik (Hrsg.), 2009, S. 23)
214. vgl. (Frick et al., 2009, S. 313ff)

### 5. Interoperabilität

Im Rahmen des vorhandenen Forschungsprojekts wird das Hauptmerkmal auf die Entwicklung von serviceorientierten Architekturen gesetzt. Deshalb muss die Problematik der Verwendung unterschiedlicher Datenformate zwischen heterogenen Systemen durch eine dialogfähige Infrastruktur adressiert werden. Somit wird ein BUS-System benötigt, welches die Interoperabilität zwischen Anwendungen und anderen Komponenten durch standardisierte Adapter und Schnittstellen anbietet.[215]

### 6. Konsistenz

Durch Mehrbenutzerbetrieb an den verschiedenen Software-Prüfdiensten ist zu berücksichtigen, dass Daten durch lesenden und schreibenden Zugriff nicht beschädigt werden.[216] In Tabelle 13 wurde festgelegt, dass das BUS-System als Cluster aufgesetzt wird. Infolgedessen muss das BUS-System die Konsistenz der Daten durch bidirektionale Datensynchronisation erhalten, damit generierte Daten, wie beispielsweise im Service Repository, auf allen Cluster Knoten vorhanden sind.

### 7. Kompatibilität

Eine weitere wichtige Anforderungen ist die Kompatibilität zu den vorhandenen System-Containern, damit die zukünftige Weiterentwicklung des BUS-System für die Entwickler schneller umgesetzt werden kann. Ein System-Container bezeichnet hierbei den Anwendungsserver selbst. Diese Kompatibilität soll erreicht werden, indem der Bus auf den vorhandenen Anwendungsserver portiert wird. Folglich muss das BUS-System bereits kompatibel zu dem existierenden Anwendungsservern sein oder durch spätere Anpassung in den System-Container integriert werden können.

### 8. Orchestrierung

Es ist notwendig, die Menge der lose gekoppelten Prüfdienste, welche auf unterschiedlichen Domänen verteilt sind, innerhalb des BUS-System zu aggregieren.[217]

Die Orchestrierung bezeichnet in Analogie zu einem organisatorischen Arbeitsablauf, die Spezifikation und Koordination mehrerer Dienste als einzelner Aggregierungs-Service. Für den Entwickler bildet die Orchestrierung somit eine Unterstützung von automatisierten Geschäftsprozessen unter den lose gekoppelten Prüfdiensten. Zusammenfassend erzeugt die Orchestrierung durch Interaktion der einzelnen Dienste neue geordnete Geschäftsprozesse.[218]

### 9. Dokumentation

Eine grundlegende Anforderung an Software bezeichnet die umfassende Dokumentation eines Entwicklers. Auch das BUS-System muss diese Anforderung erfüllen, sodass in keiner Stufe der prototypischen Konzeption und Entwicklung unlösbare Probleme entstehen können. Hilfreich für eine zielorientierte Ent-

---

215. vgl. (Papazoglou, M., 2008, S. 270)
216. vgl. (Rahm, 1994, S. 133)
217. vgl. (MuleSoft Inc. (Hrsg.), 2014b)
218. vgl. (MuleSoft Inc. (Hrsg.), 2014a)

wicklung können angebotene Spezifikationen, Konstruktionsunterlagen, Diagramme oder Abbildungen des Produktes sein. Ebenfalls von Bedeutung sind mögliche Anleitungen für die Entwicklung bestimmter Funktionen. Weiterhin werden auch technische Unterlagen benötigt sowie dokumentierte Möglichkeiten der Software-Prüfung.[219]

#### 10. Benutzerfreundlichkeit

Die Benutzerfreundlichkeit bestimmt ebenfalls eine wichtige Anforderung an das BUS-System. Ob eine komplett eigenständige BUS-System-Lösung entwickelt oder eine bestehende Lösung adaptiert und angepasst wird; das BUS-System muss für den Benutzer transparent arbeiten, sodass die bereits vorhandene Benutzerqualität der Werkzeuge nicht beeinträchtigt oder verändert wird. Weiterhin bezieht sich die nichtfunktionale Anforderung Benutzerfreundlichkeit nicht ausschließlich auf die für den Nutzer unsichtbare Tätigkeit des BUS-System, sondern ebenfalls auf die Weiterentwicklung des BUS-Systems mit neuen Werkzeugen zur Unterstützung von Prüfverfahren. Demnach muss eine stabile und funktionierende Plattform, beispielsweise in Form einer Entwicklungsumgebung, gegeben sein.

### Ergebnis der Bus-Evaluation

In folgendem Kapitel werden die Teilergebnisse der drei unterschiedlichen Enterprise Service Busse zusammengetragen und aufgezeigt, welches System sich am besten für die Integration der vorhandenen Werkzeuge für die Unterstützung von Software-Prüfverfahren im aktuellen Forschungsprojekt eignet. Zunächst werden die einzelnen Kriterien der Nutzwertanalyse in Tabellenform dargestellt und die gewichteten Teilnutzen, welche die jeweiligen ESBs in diesem Kriterium erreicht haben, notiert. Im nächsten Schritt wird das System mit dem höchsten Gesamtergebnis aufgezeigt sowie die entscheidenden Punkte für den Einsatz innerhalb des Forschungsprojekts verdeutlicht.

Tabelle 14 stellt die einzelnen Teil-Nutzwertanalysen der drei ESBs in einer gesamten Nutzwertanalyse dar, sodass der bestplatzierte ESB aufgezeigt werden kann. Damit ergibt sich folgende Platzierung: Mule ESB erreicht mit einer Gesamtpunktzahl von 2.71 den letzten Platz. Als Zweitplatziertes folgt der Apache ServiceMix und erreicht 3.46 Punkte. Der Gewinner dieser Nutzwertanalyse und damit der Enterprise Service Bus, welcher im aktuellen Forschungsprojekt für die Verknüpfung von Werkzeugen zur Unterstützung von Software-Prüfverfahren verwendet wird, ist der Open ESB mit einer Gesamtpunktzahl von 4.017.

Weiterhin wird deutlich, weshalb Mule ESB kein geeignetes BUS-System für den Einsatz im aktuellen Forschungsprojekt darstellt. Das Vorkriterium Open Source besteht dieses System zwar, dennoch erreicht es im Hauptkriterium Open Source nur einen gewichteten Teilnutzen von 0.2 und somit den geringsten Wert. Dies ist auf die vorhandene CPAL Lizenz des Mule ESB zurückzuführen, welche für das vorhandene Forschungsprojekt ungeeignet ist. In diesem äußerst wichtigen Kriterium erreicht der Apache ServiceMix

219. vgl. (Schmidt, 2013, S. 17)

die höchste Punktzahl. Diese Punktzahl ist mit der Apache 2.0 Lizenz zu begründen, welche deutlich mehr Freiheiten für den Entwickler zulässt, als die Lizenzen der anderen BUS-Systeme.[220]

| | | Mule ESB | Apache ServiceMix | Open ESB |
|---|---|---|---|---|
| **Kriterium:** | **Gewichtung:** | **gewichteter Teilnutzen:** | **gewichteter Teilnutzen:** | **gewichteter Teilnutzen:** |
| Routing | 7.75 % | 0.31 | 0.31 | 0.31 |
| Ausfallsicherheit | 7.5 % | 0.15 | 0.3 | 0.3 |
| Verarbeitung | 7.3 % | 0.292 | 0.292 | 0.292 |
| Umgebung | 5 % | 0.15 | 0.2 | 0.2 |
| Open Source | 10 % | 0.2 | 0.4 | 0.3 |
| Hochverfügbarkeit | 7.5 % | 0.075 | 0.3 | 0.375 |
| Skalierbarkeit | 6.65 % | 0.133 | 0.1995 | 0.266 |
| Flexibilität | 7.5 % | 0.3 | 0.3 | 0.3 |
| Interoperabilität | 7.25 % | 0.29 | 0.29 | 0.29 |
| Konsistenz | 5.25 % | 0.105 | 0.1575 | 0.1575 |
| Kompatibilität | 9 % | 0.0 | 0.18 | 0.45 |
| Orchestrierung | 7.25 % | 0.29 | 0.29 | 0.29 |
| Dokumentation | 5.8 % | 0.29 | 0.116 | 0.174 |
| Benutzerfreundlichkeit | 6.25 % | 0.125 | 0.125 | 0.3125 |
| **Gesamt:** | **100 %** | **2.71 (3.)** | **3.46 (2.)** | **4.017 (1.)** |

Tabelle 14: Gesamtergebnis der Nutzwertanalyse

Alle ESBs verfügen über hohes Potential und bieten eine außerordentliche Auswahl an Funktionalität für die Integrierung der vorhandenen Werkzeuge. Obwohl der Mule ESB sowie der Apache ServiceMix bei den Kriterien Routing, Verarbeitung und Umgebung auf gleichem Niveau mit dem Open ESB liegen, generieren Sie dennoch in den hochgewichteten Schlüsselkriterien bedeutend weniger Punkte als der favorisierte Open ESB. Besonders sichtbar wird dies für das Attribut Kompatibilität. Hier erreicht der Mule ESB keinerlei Punkte, aufgrund der Inkompatibilität zu dem vorhandenen System-Container. Der Apache ServiceMix besticht ebenfalls nicht mit einer hohen Kompatibilität zu dem vorhandenen Forschungsprojekt, denn eine Realisierung dieses ESBs innerhalb des vorhandenen System-Containers ist nur bedingt möglich. Der Open ESB dagegen verfügt über einen hohen Grad an Kompatibilität zum vorhandenen Forschungsprojekt. Die nichtfunktionalen Anforderungen haben bereits die vollständige Kompatibilität

220. vgl. (Free Software Foundation Inc. (Hrsg.), 2014)

durch die Verwendung des identischen System-Containers hinsichtlich des Sun Glassfish Anwendungsserver verdeutlicht.

Weitere Unterschiede sind in den Kriterien Ausfallsicherheit, Hochverfügbarkeit und Skalierbarkeit zu erkennen. Diese Kriterien stehen alle in einer Beziehung zueinander, aufgrund ihrer gewissen Similarität. In diesen Kriterien kommt der Mule ESB seiner Konkurrenz nicht nach. Aufgrund der verwendeten Open Source Lizenz wird der Mule ESB immens eingeschränkt und kann diese Kriterien nur in der kommerziellen Version ordnungsgemäß adressieren. Deutliche Parallelen zeigen die beiden anderen ESBs in diesem Bereich auf. So verfügt der Apache ServiceMix in dem Kriterium Ausfallsicherheit über die gleiche Punktzahl wie das erstplatzierte System und unterscheidet sich nur in einer sehr geringen Punktzahl in den Bereichen Hochverfügbarkeit und Skalierbarkeit.

Weitere Parallelen werden für die Flexibilität, Interoperabilität und Orchestrierung sichtbar. Alle Systeme verfügen über die gleiche Wertung in diesen Bereichen, womit eine positive Steuerung der Anwendung und Services durch alle BUS-Systeme bereitgestellt wird.

Lediglich in der Dokumentation erreicht der Mule ESB die höchste Punktzahl der drei BUS-Systeme. Diese hebt sich deutlich von denen der Konkurrenz ab. Nicht nur eine bessere Ordnung der Dokumente, sondern auch eine wahrnehmbar umfangreichere Dokumentation, als die der anderen Systeme, bietet das Unternehmen des Mule ESB seinen Kunden an.

Im letzten Punkt zeigt das erstplatzierte BUS-System ebenfalls seinen deutlichen Gesamtabstand zu den übrigen Systemen mit einer hohen Punktzahl in dem Kriterium Benutzerfreundlichkeit. Dabei wurde dieses Kriterium in allen drei BUS-Systemen in zwei Aufgabenbereiche unterteilt. Zum einen in die nichtfunktionale Anforderung Benutzerfreundlichkeit in Form von Transparenz des ESBs für den Software-Prüfer sowie zum anderen auf Basis der Benutzerfreundlichkeit während der Entwicklung. Alle Enterprise Service Busse können zumindest den Punkt Transparenz erfüllen, nur das System des Open ESB sticht mit seiner eigenen Entwicklungsumgebung auf Basis von Netbeans, sowie den unzähligen grafischen Editoren für die Entwicklung hervor.

Der Open ESB stellt sich als bester ESB der vorhandenen Systeme heraus, sodass dieser im Rahmen des aktuellen Forschungsprojekts eingesetzt werden kann und die vorhandenen Werkzeuge für die Unterstützung von Software-Prüfverfahren in einer losen serviceorientierten Architektur verknüpfen kann.

### 5.5.2 Repository

Innerhalb von SIWOB müssen eine stetig wachsende Anzahl an Daten verwaltet und diese in einer Java EE Umgebung zur Verfügung gestellt werden. Diese Daten stellen Dateien und Ordner dar, die im Verlauf eines Prüfvorgangs generiert werden. Dieses Kapitel beschreibt die neue Struktur und Implementierung des Repository Web Services mit seinen Eigenschaften, Vorteilen und Anwendungsfällen.

Ziel des Forschungsprojekts SIWOB ist es, eine Dokumentationsinfrastruktur zu schaffen, die Software-Prüfer bei Ihrer täglichen Arbeit unterstützt. Eine notwendige Aufgabe ist dabei die konsistente Verwaltung von Dokumenten (in Form von Dateien) bei einer Vielzahl von Projekten. Zu den Aufgaben des Re-

positories zählen außerdem die Anreicherung von Dateien mit Meta-Daten und die Versionsverwaltung. Zusätzlich wird die API des Services zur Nutzung innerhalb einer Java EE Anwendung dokumentiert.

Das Repository soll die Verwaltung von Dateien auf Dateisystem-Ebene und darüber hinaus umfangreiche Meta-Daten vorhalten und bereitstellen. Insbesondere sollen Funktionen implementiert werden, wie beispielsweise Create-Read-Update-Delete (CRUD), Move, Meta-Daten CRUD und weitere Funktionen, die für eine einfache Verwaltung der Dokumente und ihrer Metadaten benötigt werden. Außerdem soll das Repository in der Lage sein alle verwalteten Dateien zu versionieren.

Für eine reibungslose Integration und Verwendung innerhalb von SIWOB, wird das Repository nach außen eindeutig durch eine Schnittstelle (Menge von Interfaces) abgegrenzt, sodass andere Services ausschließlich über diese Interfaces mit diesem Web Service kommunizieren. Zu diesem Zweck erfolgt eine Entwicklung in drei Stufen, wobei die dritte Stufe als optional behandelt wird.

1. Konzeption und Dokumentation einer geeignete Schnittstelle.
2. Implementierung notwendiger Basisfunktionen, wie CRUD, die Verwaltung der Metadaten und die Versionierung.
3. Umsetzung von Zusatzfunktionen, wie Statistiken, Performance Verbesserungen, evtl. Replikation der Daten, ein Token-basierter Download Dienst, um beispielsweise die Last der End-Points zu verringern.

Die Vorgehensweise bei der Projektdurchführung und Realisierung des Repository Web Service erfolgt gemäß dem V-Modell.

## Anforderungen und Funktionsumfang

An die Implementierung des Repository Web Service bestehen eine Vielzahl von abhängigen und unabhängigen Bedingungen. In der ersten Phase des V-Models, der sogenannten Systemanforderungsanalyse, werden deshalb Anforderungen für die Konzeption und Umsetzung erhoben. Im Folgenden werden die grundlegenden Anforderungen näher beschrieben.

### Virtuelles Dateisystem

Das Repository soll einem Dateisystem ähneln und in der Lage sein eine beliebige Datei- und Ordneranzahl in einer beliebigen Pfadtiefe zu verwalten, sofern dies durch das verwendete Speichermedium möglich ist (z.B. Einschränkungen bei der Abbildung auf Dateien für NTFS Dateisysteme). Weiterhin soll der CRUD-Mechanismus umgesetzt werden, sodass elementare Operationen durchgeführt werden können.

Ein beispielhaftes Virtuelles Dateisystem ist das Commons Virtual File System (VFS) von Apache, das eine einheitliche API für den Zugriff unterschiedlicher Dateitypen bereitstellt. Weiterhin bietet dieses System eine Zwischenspeicherung von Dateiinformationen an sowie eine optionale Speicherung auf dem lokalen Speichersystem, beispielsweise für Backup-Funktionen. Diese genannten Techniken und Funktionen von

Apache VFS geben einen kleinen Überblick über eine mögliche Funktionsweise des zu implementierenden Repositories.[221]

### Meta-Daten

Das Repository soll eine Unterstützung für eine transparente Verwaltung von Meta-Daten in beliebiger Form für Ordner und Dateien bieten. Hierbei ist eine variable Struktur von besonderer Wichtigkeit, das sogenannte Entity-Attribute-Value (EAV)-Model, da Meta-Daten von Benutzern flexibel angelegt werden sollen.

Das EAV-Model bedient sich einer Meta-Daten-Infrastruktur mit Verwendung eines Objektverzeichnisses in tabellarischer Form. Die Vorteile dieser Struktur bestehen in der Modellierung von internen Klassenbeziehungen, wie auch der Unterstützung von Klassen als Teilnehmer anderer Klassen. Weiterhin können Klassen Attribute zwischen anderen Klassen vererben sowie die Visualisierung der Daten in zweiter Normalform bereitstellen, sodass jedes nicht-primäre Attribut von allen Schlüsseln abhängig ist und nicht nur von einem Teil eines Schlüssels.[222]

### Versionsverwaltung

Das Repository soll in der Lage sein eine beliebige Anzahl von Versionen einer Datei, inklusive der zugeordneten Meta-Daten, zu speichern und diese bei Bedarf auszulesen und wiederherzustellen. Es soll nur die Funktion für die Versionsverwaltung zur Verfügung gestellt werden. Des Weiteren soll die Versionsverwaltung anhand der objektorientierten Public API realisiert werden. Ebenso soll die Versionsverwaltung nur für Dateien umgesetzt werden, da eine Versionsverwaltung von Ordnern und den darin enthaltenen Datenstrukturen (Dateien und Unterordner) einen erheblich größer dimensionierten Speicherplatz und Aufwand bedarf.

### Web Service mit Client Side API

Das Repository soll über eine Web Service API aufrufbar sein. Hierbei soll eine klare Abstraktion zwischen Repository und Web Service API bestehen. Der Web Service benutzt ausschließlich Methoden der öffentlichen Schnittstelle des Repository.

Damit die Verwendung des Web Services innerhalb von SIWOB vereinfacht wird, soll eine Client Side API entwickelt werden, die eine Verwaltung bzw. Steuerung des Repository auf einem entfernten System (Server) erlaubt. Die Nutzung des Web Services erfolgt anhand von objektorientierten Verfahren.

Es werden Methoden auf sogenannten Proxy Objekten aufgerufen, welche Aufrufe über den Web Service ausführen und die Ergebnisse anhand von konkreten Daten oder weiteren Proxy Objekten liefern. Dies kann beispielsweise durch die Traversierung des Dateisystems realisiert werden, sodass alle Elemente des Dateisystems durchlaufen werden können.

---

221. vgl. (The·Apache·Software·Foundation·(Hrsg.),·2014)

222. vgl. (Nadkarni,·Prakash,·1999)

Ein Aufruf von Proxy Objekten kapselt die Konvertierung von Daten, Fehlermeldungen und Aufrufen in einer einfachen und zugleich vollständigen API, mit der gleichzeitig die Produktivität bei der Entwicklung von Komponenten verbessert wird.

#### Performance und Speicherverwaltung

Aufgrund einer möglichen parallelen Verwendung des Repository Web Service durch mehrere Benutzer, müssen unter Umständen größere Datenmengen verwalten werden, sodass eine effiziente Umsetzung unerlässlich ist. Weiterhin soll die Anwendung mit einer minimalen Speicher und Central Processing Unit (CPU) Auslastung erfolgen.

Zu diesem Zweck werden Soft-Referenzen auf Objekte verwendet, sodass ein Objekt nicht zwingend im Arbeitsspeicher vorgehalten werden muss. Sollte der Garbage Collector zusätzlichen Speicher benötigen, so muss der vorgehaltene Dateibaum von der untersten Ebene bereinigt werden. Referenzen auf zwischengespeicherte Daten und Meta-Daten können jederzeit aus dem Speicher entfernt und bei Bedarf wieder geladen werden.

#### Portabilität

Eine weitere Anforderung ist die Austauschbarkeit des Speichermediums, welche einen hohen Grad an Abstraktion und die Anwendung eines Adapter Patterns erfordert. Unter Umständen ist ein einzelnes Dateisystem für die Verwaltung der Datenmengen nicht ausreichend und eine alternative Lösung ist erforderlich, wie beispielsweise eine Verteilte Datenbank. Das System soll folglich so konzipiert werden, dass die Speicherlösung austauschbar ist.

### 5.5.3 User Interface

Dieses Kapitel beschreibt die Visualisierung der notwendigen Bedienoberfläche (User Interface) für Prüfer und Hersteller. Zunächst werden hierzu forschungsrelevante Grundlagen für die Umsetzung einer möglichen Visualisierung zusammengetragen, woraufhin die Konzeption und Entwicklung der Bedienoberfläche folgt.

Das Ziel der Bedienoberfläche ist es Herstellern und Prüfern eine Schnittstelle zur Interaktion zu bieten über die beispielsweise anfallende Dokumente verwaltet oder Kommentare und Anfragen übermittelt bzw. eingesehen werden können. Das User Interface entspricht dabei den aktuellen Anforderungen der Usability-Forschung.

Das entwickelte System soll dabei allen Benutzern ein einheitliches User Interface für die Benutzung zur Verfügung stellen, basierend auf gängigen Web-Technologien. Das User Interface (UI) soll dafür geeignet sein, sämtliche notwendigen Funktionen abzubilden, indem Möglichkeiten zur Ein- und Ausgabe von Dokumenten, zum Durchlaufen von Workflows, zur Unterstützung sämtlicher Kommunikation sowie zur Abfrage von Statusinformationen bereitgestellt werden.

Das User Interface dient dem Prüfer zur direkten Interaktion mit den Services, die an das BUS-System der SIWOB Workbench gekoppelt sind. Dadurch erhält ein Prüfer die Möglichkeit, eingereichte Unterlagen

verschiedener Dokumenttypen eines Hersteller an einer zentralen Stelle einsehen zu können. Weiterhin kann er Dokumente über passende Services beispielsweise kommentieren, Rückfragen formulieren und viele weitere Funktionen ausführen sowie Arbeitspakete an den Hersteller per E-Mail senden. Auf der anderen Seite wird ein Hersteller, über das ihm bereitgestellte User Interface, bei der Erstellung einer Spezifikation unterstützt. Hierzu dient das User Interface des Herstellers als Schnittstelle zu einer Eingabemaske, die durch einen passenden Web Service bereitgestellt wird, über den er eine Spezifikation per Eingabemaske erzeugen oder öffnen kann. Aus den Eingaben kann der Hersteller über das User Interface ein lesbares unveränderliches Dokument erzeugen, welches konform zu den aufgestellten Anforderungen ist und kann dieses Dokument, inkl. benötigter Quellcode-Dateien, an einen Prüfer übermitteln.

#### Responsive Technologien auf Basis eines Open Source Frameworks

Für die Realisierung einer Web-Oberfläche sollen ausschließlich Komponenten mit Open-Source Lizenzen in Betracht gezogen werden. Diese Eigenschaft ist zwingend erforderlich, denn im Rahmen des aktuellen Forschungsprojektes soll keine Software zum Einsatz kommen, welche einer kommerziellen Lizenz unterliegt.

Als Vorlage bietet sich das Open-Source Framework Bootstrap in Version 3.3.X an. Dieses Framework verfügt über die Open-Source Lizenz des MIT und kann somit innerhalb des Forschungsprojekts SIWOB verwendet werden.[223] Zudem hat Bootstrap eine weite Verbreitung gefunden, da seit der Veröffentlichung des Frameworks mit dem Social-Media Dienst Twitter im Jahr 2010 die Bedeutung sowie die Popularität des Frameworks gestiegen sind.[224]

Weiterhin verfügt Bootstrap über Techniken, welche eine einfache und effiziente Portierung auf alle gängigen Geräten mit Internet-Browser-Unterstützung ermöglichen. Somit können einzelne Web-Seiten sowie Anwendungen, welche auf dem gleichen Quellcode basieren parallel auf Smartphone, Tablet oder PC dargestellt werden. Weiterhin enthält Bootstrap eine Reihe von Erweiterungen, wie beispielweise die ausführliche Dokumentation allgemein bekannter HTML-Elemente sowie die Unterstützung spezieller HTML und CSS Komponenten. Zusätzlich bietet Bootstrap die Verwendung nützlicher Javascript Werkzeuge an, welche dazu beitragen die Entwicklung einer Web-Oberfläche in hohem Maße zu vereinfachen und beschleunigen.[225]

### Anforderungen an das User Interface

Im Rahmen der Konzeption und prototypischen Entwicklung bestimmen eine Reihe von nichtfunktionalen Anforderungen die Eigenschaften des User Interface. In diesem Kapitel werden diese Anforderungen an das User Interface vorgestellt und ihre Notwendigkeit erläutert. Für die Anforderungsentwicklung stellt die Analyse der vorhandenen Web Services, die an das BUS-System gekoppelt sind, einen zentralen Arbeitsschritt dar.

---

223. vgl. (Bootstrap (Hrsg.), 2014b)

224. vgl. (Bootstrap (Hrsg.), 2014a)

225. vgl. (Bootstrap (Hrsg.), 2014b)

### Open Source

Die Web-Oberfläche soll eigenständig oder mit der Unterstützung vorhandener Open Source Software entwickelt werden. Der Vorteil von Open Source Software besteht darin diese kostenlos verwenden und insbesondere verändern zu können. Darüber hinaus ist es häufig möglich Entwicklungen auf Basis von Open Source an Benutzer zu verteilen, beispielsweise in Form einer fertig entwickelten Anwendung.[226] Diese Eigenschaft ist von hoher Bedeutung innerhalb des Forschungsprojekts, denn keines der aktuell entwickelten Werkzeuge verwendet eine kommerzielle Lizenz.

### Benutzerfreundlichkeit

Die Benutzerfreundlichkeit ist ebenfalls eine wichtige Anforderung an das User interface. Die Steuerung der Web-Oberfläche muss für einen Prüfer, wie auch für einen Hersteller, intuitiv funktionieren. Dabei soll auf selbsterklärende Funktionen sowie Beschriftungen geachtet werden. Ebenfalls muss die Perspektive des Entwicklers betrachtet werden. Demnach sollte das Open Source Framework eine stabile und leistungsfähige Plattform bereitstellen.

### Kompatibilität

Das vorhandene Frontend muss zu den bereitgestellten Werkzeugen des SIWOB Forschungsprojekts kompatibel sein. Aufgrund der Vielzahl an etablierten Werkzeugen, welche zur Software-Prüfung beitragen, muss die Entwicklung der Web-Oberfläche parallel realisierbar sein, sodass bei kurzfristigen Änderungen der Werkzeuge ebenfalls eine schnelle Anpassung oder Erweiterung im Frontend stattfinden kann. Insbesondere muss das User Interface zu dem bereitgestellten Anwendungsserver kompatibel sein, sodass bei kurzfristiger Änderung der Web-Oberfläche eine zeitsparende Aktualisierung durchgeführt werden kann.

### Dokumentation

Eine grundlegende Anforderung an das User Interface bezeichnet die umfassende Dokumentation der Software. Somit soll das User Interface in keiner Stufe der prototypischen Konzeption und Entwicklung undokumentierte Probleme auslösen. Hilfreich für eine zielorientierte Entwicklung können angebotene Spezifikationen, Konstruktionsunterlagen, Diagramme oder Abbildungen des Open Source Frameworks sein. Ebenfalls von Bedeutung sind mögliche Anleitungen für die Entwicklung bestimmter Funktionen. Weiterhin werden auch technische Unterlagen benötigt.[227]

### Responsive Web Design

Responsive Web Design (RWD) bietet die Möglichkeit Web-Seiten in variablen Größenverhältnissen darzustellen. Es ist ein wichtiges Merkmal für die Visualisierung von Web-Seiten auf Tablets oder anderen mobilen Endgeräten. Die Realisierung einer Responsive Web-Oberfläche kann sowohl durch eine eigene Entwicklung mit Hilfe von CSS-Erweiterungen erstellt werden, als auch über ein etabliertes Web Fra-

226. vgl. (Opensource.org (Hrsg.), 2014)

227. vgl. (Schmidt, 2013, S. 43)

mework wie Bootstrap. Die Verwendung eines derartigen Frameworks bietet den Vorteil, dass standardisiere CSS- und JavaScript-Elemente für die Umsetzung eines Responsive Web-Designs bereitgestellt werden und eine Entwicklung beschleunigen.[228]

Die Weboberfläche soll nach dem sogenannten Mobile First Paradigma entwickelt werden. Dies bedeutet für die Bedienung der Weboberfläche, dass die einzelnen Style-Elemente für mobile Geräte nicht optional bereitgestellt werden, sondern direkt in dem Kern der Web-Oberfläche enthalten sind. Somit sind auch die Mobile First Styles in der gesamten Entwicklungsbibliothek vorhanden und nicht nur in Teilbereichen.[229]

## Prototypische Entwicklung des User Interface

Wie bereits im Stand der Forschung erläutert, entsteht das User Interface unter Verwendung des Open Source Web Framework Bootstrap. Hierdurch wird es möglich die Oberfläche des User Interface durch standardisierte Design Elemente zu realisieren. Die Verwendung von standardisierten Komponenten ermöglicht eine vereinheitlichte Visualisierung der einzelnen Web-Seiten. Dadurch wird zum einen die Erwartungskonformität der Benutzer im Umgang mit der Web-Oberfläche erhöht und gleichzeitig der Wiedererkennungswert von gleichen Funktionen auf den einzelnen Web-Seiten verbessert.

Im folgenden werden wichtige Teilbereiche aus dem gesamten User Interface selektiert und einzeln verdeutlicht, wie diese prototypisch umgesetzt wurden. Dabei wurde soweit dies möglich ist bewusst darauf verzichtet Quelltext anzugeben.

### Grafische Oberfläche des User Interface

Für die Möglichkeit der Autorisierung verschiedener Benutzergruppen wird eine Zutrittskontrolle in Form einer Login-Seite realisiert, wie in Abbildung 52 zu erkennen ist. Unterschiedlichen Benutzern sollen dadurch verschiedene Ansichts- und Bearbeitungsmöglichkeiten im User Interface zur Verfügung gestellt werden.

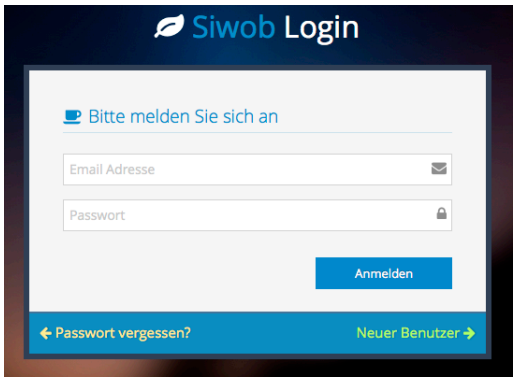



Abb. 52: Benutzer Anmeldung an der SIWOB Workbench

228. vgl. (W3Schools (Hrsg.), 2014a)

229. vgl. (Bootstrap (Hrsg.), 2014c)

Insbesondere Prüfern darf nur eine eingeschränkte Bearbeitungsmöglichkeit von Spezifikationen zugewiesen werden. Wohingegen Hersteller die Funktionalität zur Erstellung einer Spezifikation mit Hilfe einer Maske gewährt werden muss, welche im User Interface visualisiert wird. Demnach können Hersteller Spezifikationen über eine externe Anwendung anlegen und anschließend bearbeiten, wohingegen Prüfer über das entwickelte User Interface die bereits vorhandene Spezifikationen nur lesen können.

Abbildung 53 verdeutlicht die prototypische Umsetzung des User Interface. Die vorhandene Abbildung präsentiert dabei die Willkommensseite, auf welche der Benutzer nach Eingabe von Benutzername und Passwort gelangt. Oberhalb der Web-Seite befinden sich Funktionen, welche dem angemeldeten Nutzer einen Überblick ungelesener Nachrichten oder Erinnerungen zur Verfügung stellen. Im linken Bereich der Web-Seite wird dem Nutzer die Möglichkeit geboten aus einer Auswahl verschiedener (Haupt-)Menüpunkte zu selektieren. Der Menü-Reiter "Startseite" wird nach dem Login bereitgestellt und beinhaltet die Übersichtsseite. Der folgende Menü-Reiter "SIWOB Status" enthält Informationen zu den verschiedenen Werkzeugen, wie beispielsweise Verfügbarkeit oder Anbindungsgeschwindigkeit eines Werkzeugs.

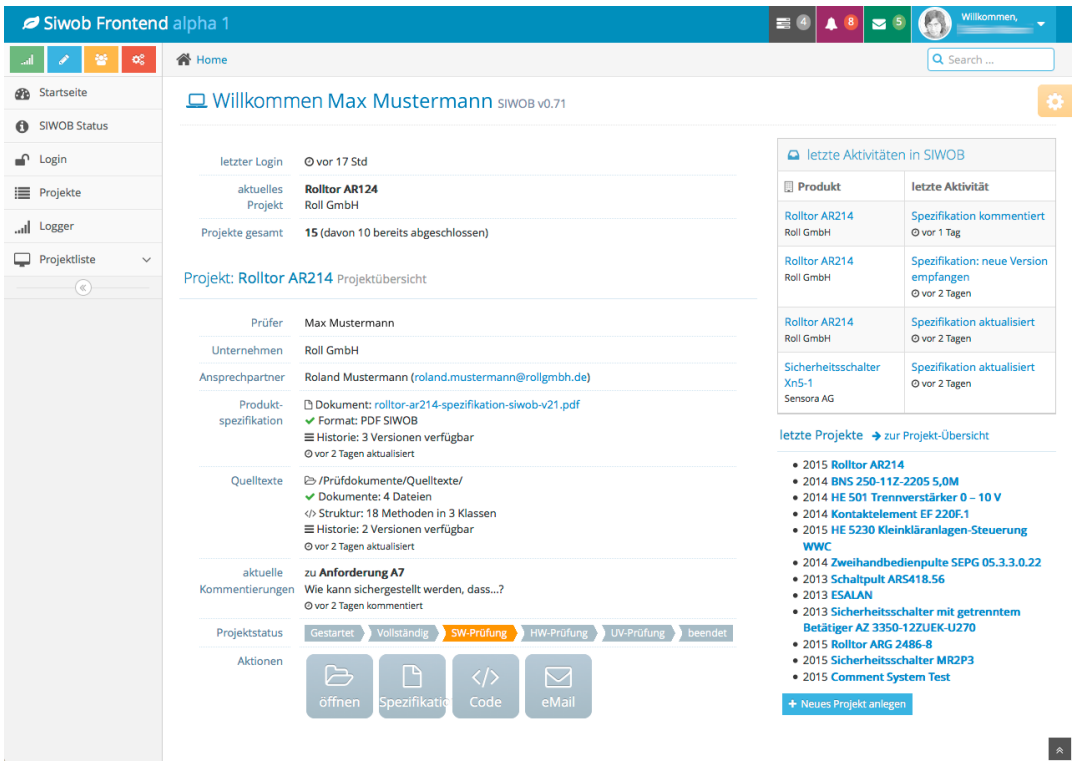



Abb. 53: Visualisierung des User Interface

Für die prototypische Testung verschiedenster Sicherheitsfunktionen wurde als nächster Reiter der Punkt "Login" eingebunden. Der darauf folgende Reiter "Projekte", welcher ebenfalls der linken Menu-Liste zugeordnet ist, beinhaltet eine detaillierte Übersicht der bereits vorhandenen Projekte. Der vorletzte Navigationspunkt "Logger" enthält die Weiterleitung zu einer Logging-Übersicht der etablierten Werkzeuge. Somit kann jederzeit die Kommunikationen der unterschiedlichen Werkzeuge überprüft werden. Im letzten Navigationspunkt wird eine ausklappbare "Projektliste" bereitgestellt, welche sich im vorhandenen Menu unterordnet. Auf der gegenüberliegenden Seite des Hauptmenüs befindet sich zunächst eine

Übersicht der letzten vier Aktivitäten innerhalb der bestehenden Projektstruktur. (Unmittelbar) Unterhalb der Aktivitätenübersicht wird ein Überblick der letzten hinzugefügten Projekte dargestellt. Weiterhin kann in diesem Bereich der Web-Seite über einen Button ein Projekt angelegt werden. Innerhalb des Content-Bereiches werden Meta-Informationen über den angemeldeten Benutzer ausgegeben. In diesem Bereich werden sowohl Informationen über den letzten Login, als auch über das letzte bearbeitete Projekt bereitgestellt. Weiterer Inhalt befindet sich unmittelbar unter den Meta-Daten des angemeldeten Benutzers. Dieser Teilbereich enthält eine detaillierte Übersicht mit Informationen zu dem aktuell bearbeiteten Projekt des angemeldeten Benutzers. Zum einen werden Meta-Daten, wie beispielsweise Prüfername, Unternehmen sowie Ansprechpartner angezeigt. Zum anderen können Projektstatus und Informationen zu vorhandenen Spezifikationen sowie aktuellen Kommentierungen dieser Dokumente innerhalb des Projekts ausgegeben werden. Abschließend können in diesem Teilbereich grundlegende Funktionen für die Unterstützung einer Software-Prüfung mit vorkonfigurierten Buttons ausgeführt werden.

### Implementierung von Responsive Technologien im User Interface

Die prototypische Umsetzung des Responsive Designs kann mit den Responsive Utilities von Bootstrap realisiert werden. Diese Werkzeuge werden für eine effiziente Entwicklung mobiler Endgeräte bereitgestellt und verfügen über sogenannte Werkzeug-Klassen für die Möglichkeit bestimmten Seiten-Inhalt zu verstecken oder anzuzeigen. Zudem können diese Werkzeuge vordefiniert, Inhalte für eine Druckversion anordnen. Die vorhandenen Responsive Werkzeug Klassen unterstützen Anzeigeformate für unterschiedliche Geräte-Kategorien. Die vorhandenen Kategorien teilen sich in Smartphone (Darstellungsfläche kleiner als 768 Pixel), Tablet (Darstellungsfläche größer gleich 768 Pixel), Desktop-PC mit kleiner Anzeige (Darstellungsfläche mindestens 992 Pixel) sowie Desktop-PC mit großer Anzeige (Darstellungsfläche mindestens 1200 Pixel) auf.[230]

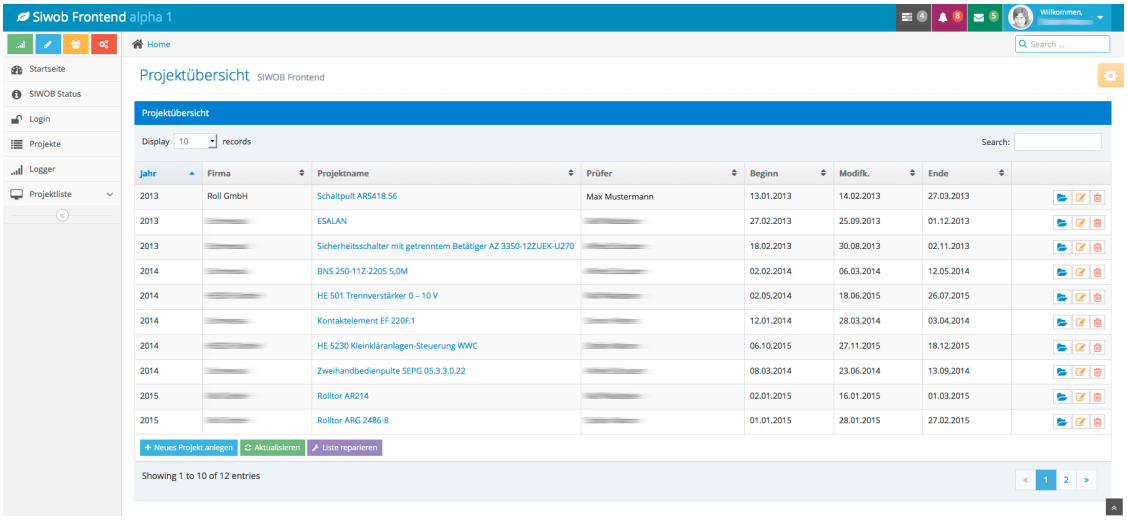



Abb. 54: Visualisierung Unterseite Projekt groß

230. vgl. (Bootstrap (Hrsg.), 2014d)

Nach erfolgreicher Design-Anpassung unter den Richtlinien der verschiedenen Geräte-Kategorien können Testfälle erstellt werden, welche eine Prüfung der Responsive Komponenten zulassen. Es können Testfälle durch die Größenanpassung des Browser-Fensters emuliert werden oder die Anzeige der verschiedenen Elemente durch grüne Häkchen bestätigt werden. Ebenso kann versteckter Inhalt durch grüne Hinweishäkchen bestätigt werden.[231]

Die Abbildungen 54 - 56 vermitteln den Unterschied der verschiedenen Browser-Ansichten bezüglich des Responsive Designs für das Forschungsprojekt SIWOB. Auf allen Abbildungen wird die identische Unterseite "Projekte" des User Interfaces aufgerufen und in unterschiedlichen Anzeigegrößen anzeigt.

Zunächst wird mit Abbildung 54 ein Desktop-PC Format in großer Ansicht dargestellt. Die Visualisierung der Unterseite "Projekt" besitzt das Layout der Hauptseite, sodass oberhalb der Web-Seite ebenfalls Funktionen enthalten sind, welche dem angemeldeten Nutzer einen Überblick ungelesener Nachrichten oder Erinnerungen zur Verfügung stellt. Im linken Bereich der Unterseite wird dem Nutzer die Möglichkeit geboten aus einer Auswahl verschiedener (Haupt-)Menüpunkte zu selektieren. Der inhaltliche Content-Bereich verfügt über eine Tabellenstruktur, welche Informationen über die existierenden Projekte darstellt. Zu diesen Informationen gehören unter anderem das Erstellungsjahr, Firma, Projektname, Prüfer, Erstellungsdatum, Modifikationsdatum und Enddatum. An diese Informationen angehängt, befindet sich eine Darstellungsfläche mit parallel angeordneten Funktions-Buttons, welche für jedes einzelne Projekt angezeigt werden und die Möglichkeit bieten das Projekt zu öffnen, zu editieren oder zu löschen.

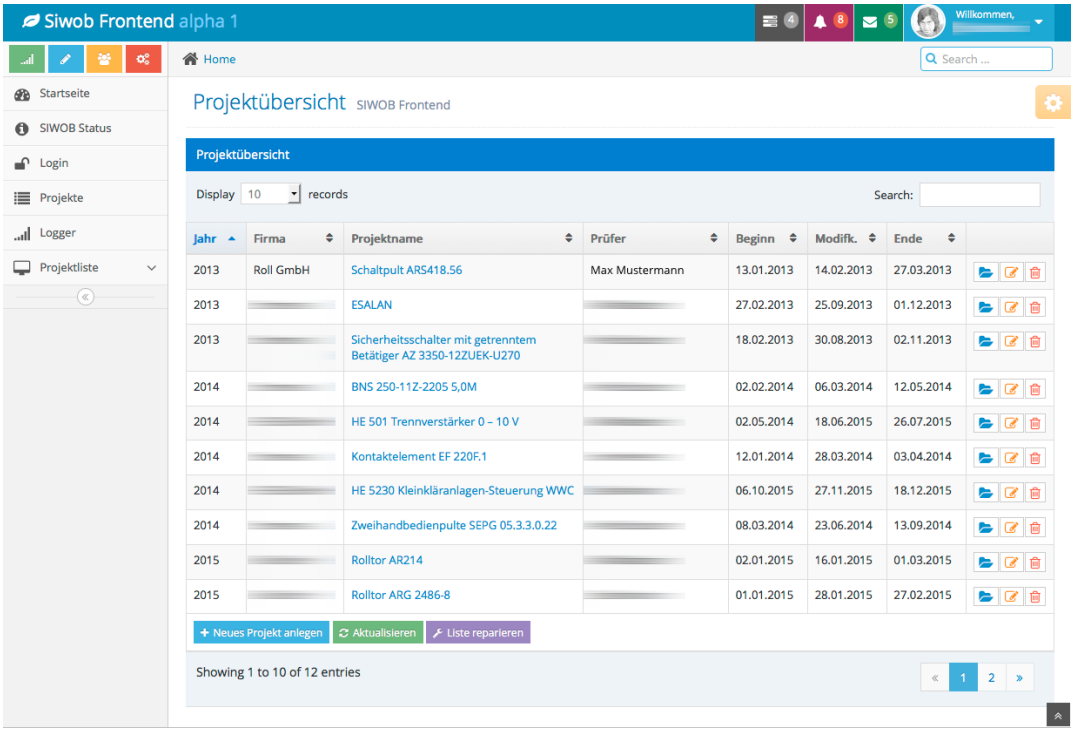



Abb. 55: Visualisierung Unterseite Projekt mittlerer Fensterbreite

231.vgl. (Bootstrap·(Hrsg.),·2014d)

Die identische Web-Seite wird in Abbildung 55 in dem Format Desktop-PC mit kleiner Anzeige (Darstellungsfläche mindestens 992 Pixel) dargestellt. Das Layout der Web-Seite erscheint identisch zu dem Desktop-PC Format mit großer Anzeige. Ein bedeutsamer Aspekt für die Einsatzfähigkeit von Responsive Design ist im Content-Bereich zu erkennen. Sowohl das Format der Tabelle, als auch die Anordnung der Funktions-Buttons passen sich mit der Größe des Web Browser-Fensters an. Dabei wird der existierende Inhalt der Web-Seite weiterhin anzeigt und durch eine engere Anordnung der Elemente in ihrer Anzeigegröße komprimiert.

In der letzten Abbildung 56 wird das Anzeigeformat für Tablets aufgezeigt. Hierbei werden bereits unübersehbare Optimierungen angewendet.

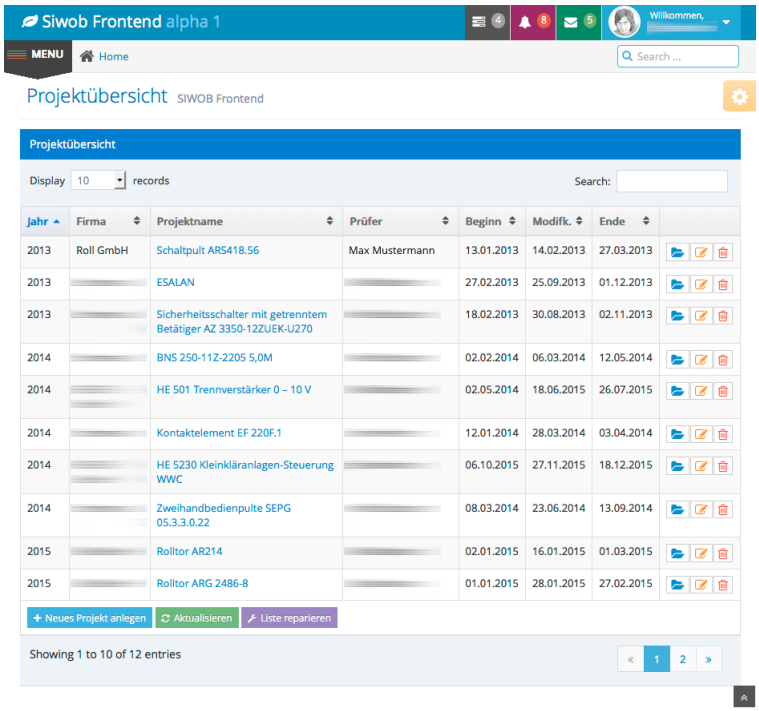



Abb. 56: Visualisierung Unterseite Projekt in Tablet-Größe

Die vorhandenen Menüpunkte aus der linken Navigationsleiste, als auch die Tabellenstruktur aus dem Content-Bereich erscheinen in einem anderen Anzeigeformat. Die Tabellenstruktur wird aus Gründen der Lesbarkeit weiter gestaucht. Das Navigationsmenü wird automatisch zum linken Seitenrand eingezogen, sodass für den Content-Bereich Platz geschaffen werden kann.

### 5.5.4 Import/Export

Die verschiedenen Benutzer der SIWOB Workbench (Hersteller oder Prüfer) sollen eine einfache Möglichkeit erhalten Dokumente eines Prüfprozesses (Code, Spezifikationen, Abbildungen, etc.) untereinander austauschen zu können. Ein Austausch von Dokumenten erfolgt auf Basis des konzipierten Container-Formates (siehe Kapitel 5.4.1) und bietet vor allem den Vorteil, dass alle Dokumente eines Exportes oder Importes konform zum konzipierten Container-Format bleiben und dadurch ein Austausch vereinfacht werden kann.

## Export Tool

Die Unterstützung des Export erfolgt über ein prototypisch konzipiertes Werkzeug, welches die Generierung und Auswertung von Container-Archiven übernimmt. Abbildung 57 zeigt in diesem Zusammenhang die Export Oberfläche des User Interface, auf das ein Benutzer (Hersteller und Prüfer) zugreifen kann, um ein Container-Archiv für den Austausch von Daten generieren zu können. Die Ansicht des Export Werkzeuges gliedert sich hierbei in drei Bereiche auf, die jeweils unterschiedliche Aufgaben übernehmen und Informationen für den Benutzer bereitstellen.

Innerhalb eines Projektes kann ein Benutzer die Export Oberfläche aufrufen. Damit für Benutzer weniger Aufwand bei der Suche einzelner Dateien entsteht, wie beispielsweise Dokumentationen, Quellcode und weitere Dokumente, erfolgt die Selektion der Zieldokumente über eine passende Auswahl. Diese Auswahl wird im unteren Bereich abgebildet und zeigt die verfügbaren Objekte (Ordner oder Dateien), die in ein Container-Archiv exportiert werden können. In dieser beispielhaften Abbildung wird der Inhalt des Ordners "Prüfdokumente" ausgegeben. Dieser beinhaltet eine Datei (Spezifikation.pdf) und einen Ordner (Quellcode), der verschiedene Quellcode-Dateien beinhaltet. Aus diesen Elementen kann ein Benutzer eine partielle Auswahl treffen oder alle Elemente selektieren, die für einen Export berücksichtigt werden sollen. Die Auswahl der Elemente wird über den Knopf "Generate" bestätigt und damit der Export Prozess gestartet.

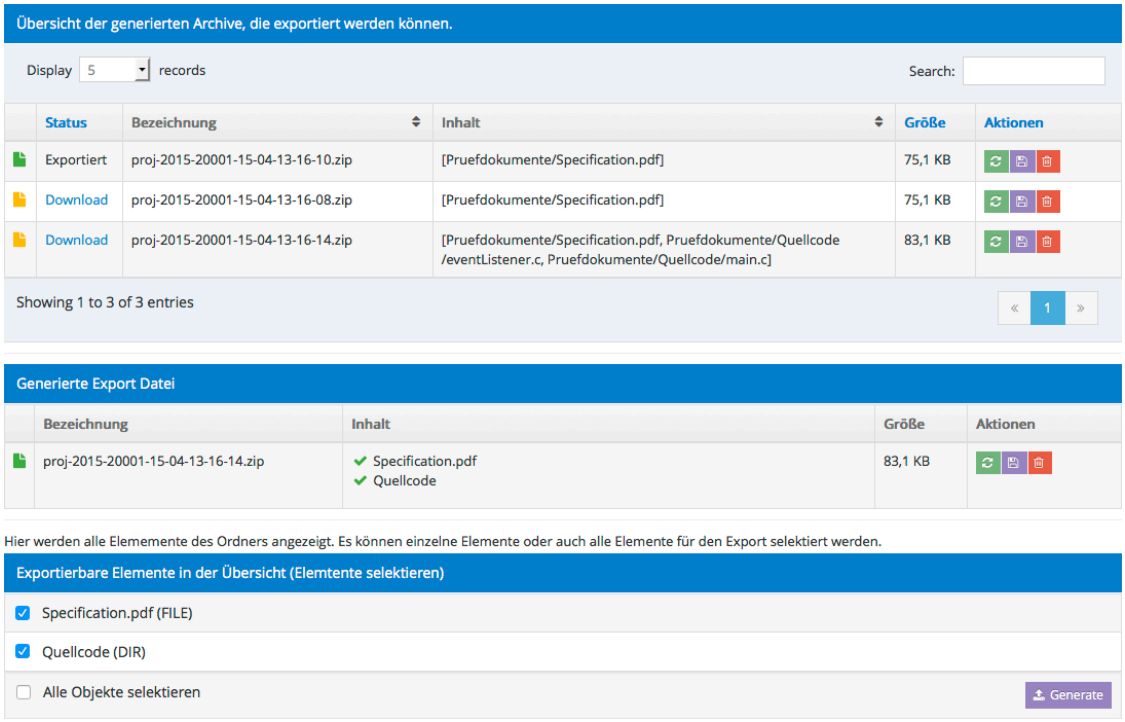



Abb. 57: Export Oberfläche des User Interface

Der Export Prozess übermittelt an einen passenden Service die getroffene Elementauswahl des Benutzers. Dieser Service sammelt auf Basis der Selektionen die assoziierten Elemente im Repository, die in ein Container-Archiv überführt werden sollen. Sollte in der Selektion ein Ordner enthalten sein, werden die darin

enthaltenen Strukturen (Dateien und Ordner), mit Hilfe von rekursiven Algorithmen, für den Export mit berücksichtigt. Sobald der Service alle Zielelemente ermittelt hat, werden diese mit den Hierarchie-Informationen des Containers in ein Container-Archiv überführt. Das Container-Archiv stellt eine Zip-Datei dar und erhält eine spezifische Bezeichnung ("proj-2015-20001-15-04-13-16-14.zip"). Diese setzt sich aus der Vorgangsnummer und dem Erstellungsdatum zusammen, sodass eine eindeutige Unterscheidung zwischen verschiedenen Container-Archiven eines Projektes und unterschiedlichen Projekten möglich wird. Für den Export generierte Container-Archive werden, unter der Vorgangsnummer des assoziierten Projektes, in einem passenden Ordner ("transfer/export") abgelegt. Das Zip-Format bietet in diesem Zusammenhang den Vorteil, dass alle selektierten Elemente in einer einzelnen Datei zusammengeführt werden.

Das Ergebnis der Generierung wird im mittleren Bereich ausgegeben und zeigt dem Benutzer, neben der Dateigröße, welche selektierten Elemente erfolgreich in das Container-Archiv überführt werden konnten. Die erfolgreiche Verarbeitung eines selektierten Elementes wird durch einen grünen Haken und eine fehlerhafte Verarbeitung durch ein rotes Kreuz symbolisiert.

Gleichzeitig wird der obere Bereich um das aktuell generierte Container-Archiv aktualisiert. Dieser Bereich liefert eine Übersicht der Container-Archive, inkl. der Dateigrößen,, die vom Benutzer generiert wurden. Hierbei findet eine Unterscheidung der Container-Archive anhand des Status statt, an dem ein Benutzer erkennen kann, ob das Container-Archiv bereits heruntergeladen oder bisher nur generiert wurde. Weiterhin wird durch die Bezeichnung erkennbar, zu welchem Projekt das Container-Archiv gehört und zu welchem Zeitpunkt es generiert wurde. In einer separaten Spalte wird zusätzlich eine Ausgabe aller im Container-Archiv enthaltenen Elemente aufgelistet, inkl. der hierarchischen Struktur, sodass die Korrektheit der Inhalte verifiziert werden kann.

Alle generierten Container-Archive beinhalten eine zum assoziierten Projekt konforme Container-Struktur. Diese können abschließend durch die angebotenen Funktionen in der Aktionsspalte heruntergeladen und lokal gespeichert werden. Ein Benutzer hat dadurch die Möglichkeit ein exportiertes Container-Archiv an einen anderen Benutzer zu übermitteln. Dieser kann beispielsweise ein Container-Archiv einer Email als Anhang hinzufügen. Im Grunde können auch andere Versandvarianten für den Datenaustausch berücksichtigt werden, wie etwa eine CD oder ein USB-Stick. Zusätzlich besteht über die Aktionsspalte die Möglichkeit ein generiertes Container-Archiv zu löschen, beispielsweise wenn es unvollständig generiert wurde.

Alle Container-Archive werden über den Export Service grundsätzlich im Zip-Format generiert und gespeichert. Benutzer können diese Archive mit Hilfe von Betriebsystem-Bordmitteln oder passenden Programmen öffnen, wie beispielsweise WinZip. Dadurch kann ein Benutzer auch ohne passendes Import Werkzeug an die Inhalte eines exportierten Container-Archivs gelangen. Die extrahierte Struktur entspricht hierbei der Container-Struktur des jeweiligen Projektes.

## Import Tool

Wie auch beim Export erfolgt die Unterstützung des Import über ein prototypisch konzipiertes Werkzeug, welches die Auswertung und Integration von Container-Archiven übernimmt. Abbildung 58 zeigt in diesem Zusammenhang die Import Oberfläche des User Interface, auf das ein Benutzer (Hersteller und Prüfer) zugreifen kann, um ein Container-Archiv einzulesen. Die Ansicht des Import Werkzeuges gliedert sich in vier Bereiche, die jeweils unterschiedliche Aufgaben übernehmen und Informationen für den Benutzer bereitstellen.

Das Import Tool setzt für eine korrekte Verarbeitung ein Container-Archiv voraus, welches zuvor über das Export-Tool von einem beliebigen Benutzer exportiert wurde. Zusätzlich muss ein Projekt mit Vorgangsnummer existieren, damit eine Zuordnung für das Container-Archiv möglich ist. Dadurch entsteht für Benutzer weniger Aufwand bei der Integration einzelner Dateien, wie beispielsweise Dokumentationen, Quellcode und weitere Dokumente, da die Eingliederung der Daten über ein konformes Container-Format erfolgt.

Neben dem Import von Container-Archiven besteht die Möglichkeit einzelne Dateien zum Datenbestand eines Projektes hinzuzufügen. Diese Integration erfolgt nicht über das Import Tool, sondern beispielsweise über eine bereitgestellte Funktion im User Interface zum Hochladen einzelner Dateien auf der jeweiligen Hierarchieebene eines Projektes.

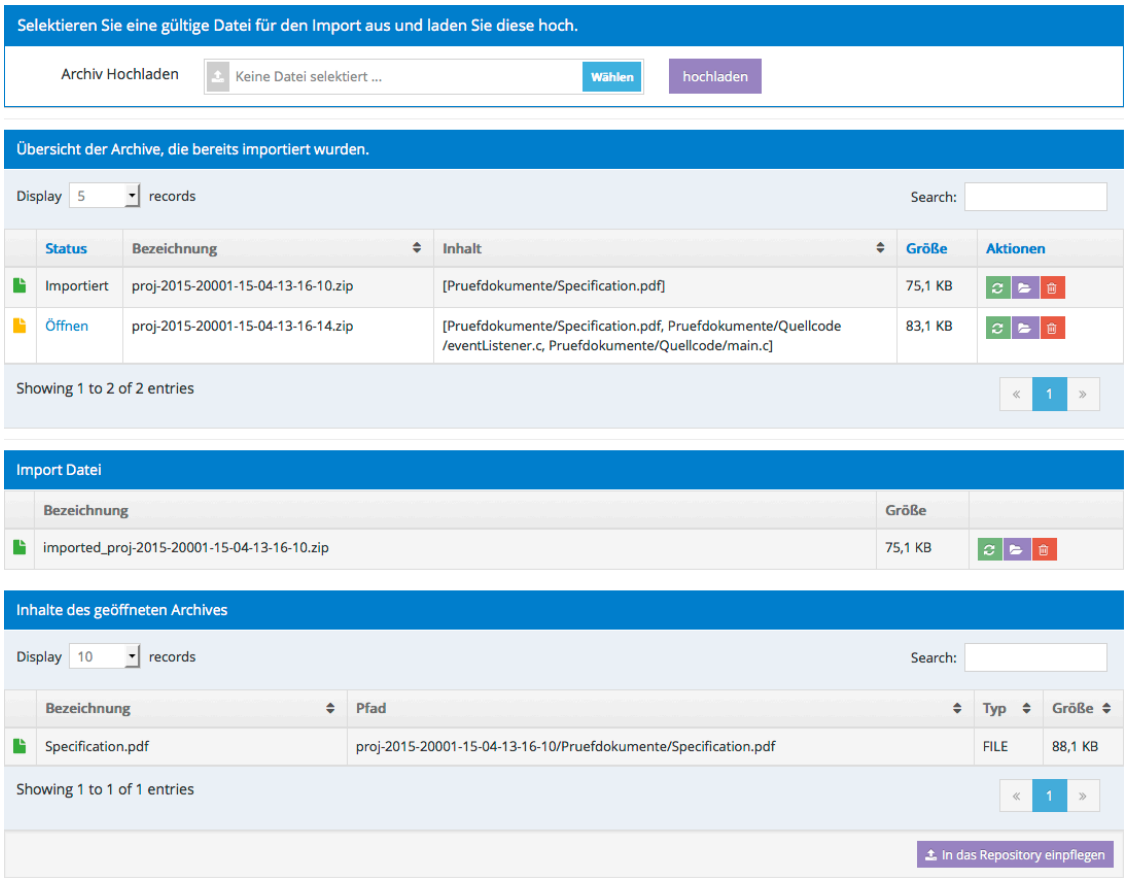



Abb. 58: Import Oberfläche des User Interface

Innerhalb eines Projektes kann ein Benutzer die Import Oberfläche aufrufen. Die erste Aufgabe besteht darin ein gültiges Container-Archiv für den Import auszuwählen. Hierzu wird im oberen Bereich, über den Knopf "Wählen", ein Auswahlfenster geöffnet, sodass der Benutzer die Möglichkeit erhält in seinem lokalen Datenbestand ein Container-Archiv zu selektieren. Dieser Container kann beispielsweise im Form eines Email-Anhanges beim Benutzer eingegangen sein. Die Selektion wird mit dem Knopf "hochladen" bestätigt, woraufhin eine Gültigkeitsprüfung des Container-Archivs erfolgt. Sollte die selektierte Datei kein Container-Archiv sein oder nicht zum aktuellen Projekt passen, wird der Benutzer mit einer passenden Informationen informiert. Bei einer erfolgreichen Untersuchung wird das Container-Archiv, unter der Vorgangsnummer des assoziierten Projektes, in einem passenden Ordner ("transfer/import") integriert, in dem sich alle importierten Container-Archive befinden.

Nach erfolgreicher Integration des Container-Archivs in den Datenbestand des Projektes, wird die Übersicht der Archive um einen neuen Eintrag aktualisiert. Dieser Bereich befindet sich direkt unterhalb der Archiv-Selektion und visualisiert dem Benutzer in einer Übersicht alle Container-Archive, inkl. der Dateigröße, die bereits importiert wurden. Wie auch beim Export findet eine Unterscheidung der Container-Archive anhand des Status statt, an dem ein Benutzer erkennen kann, ob ein Container-Archiv bereits in das Repository integriert oder bisher nur hochgeladen wurde. Weiterhin wird durch die Bezeichnung erkennbar, zu welchem Projekt das Container-Archiv gehört und zu welchem Zeitpunkt es generiert wurde. In einer separaten Spalte wird zusätzlich eine Ausgabe aller im Container-Archiv enthaltenen Elemente und ihrer hierarchischen Pfadstruktur im Projekt abgebildet.

Mit Hilfe der angebotenen Funktionen, in der Aktionsspalte der Übersicht, kann ein beliebiges Container-Archiv vom Benutzer geöffnet werden. Der Import Prozess übermittelt an einen Service die getroffene Container-Auswahl und ermittelt die im Container-Archiv enthaltenen Informationen. Auf Basis dieser Informationen, über Ordner und Dateien, berechnet der Service die dazu äquivalenten Container-Strukturen im assoziierten Projekt, anhand der eine Integration in den Datenbestand des Projektes erfolgen kann.

Als Ergebnis werden im unteren Bereich der Import Oberfläche die Inhalte des geöffneten Archivs ausgegeben. Dieses beinhaltet eine Listenansicht mit der jeweiligen Elementbezeichnung, dem Datentyp und der Dateigröße. Hervorzuheben ist in dieser Ansicht der berechnete Pfad, in den das jeweilige Element in das Repository integriert werden soll. Abbildung 58 zeigt in diesem Zusammenhang den Inhalt eines geöffneten Container-Archivs mit dem einzigen Element der "Spezifikation.pdf" und dem dazu passenden Container-Pfad im Projekt.

Eine Integration in das Repository erfolgt durch die Betätigung des Knopfes "In das Repository einpflegen". Sollte das geöffnete Container-Archiv bereits in das Projekt aufgenommen worden sein, ist eine erneute Integration nicht möglich und wird dem Benutzer mit einer entsprechenden Meldung visualisiert. Die wiederholte Integration dieser Daten würde zu unnötigen Duplikaten führen, beispielsweise von Dateien. Ist das Container-Archiv allerdings noch nicht integriert, wird die Liste der zu importieren Elemente an einen Service zur Bearbeitung übergeben.

Der Service verarbeitet jedes einzelne Element (Dateien und Ordner) der transferierten Liste und integriert diese in die Container-Struktur des Projektes. Bei der Integration des Container-Archivs werden in einem ersten Schritt die Ordnerstrukturen verarbeitet. Diese werden im Container des Projektes auf Existenz überprüft und bei Bedarf angelegt, damit dort die Dateien fehlerfrei abgelegt werden können. Anschließend erfolgt mit Hilfe der Versionierung eine Integration der Dateien des Container-Archivs, sodass eine Prüfung auf bestehende Versionen gewährleistet wird. Sollte noch keine Version einer Datei vorhanden sein, wird diese als initiale Version aufgenommen. Existieren jedoch bereits Versionen zu einer Datei, wird die Zieldatei zuvor versioniert, bevor die neue Datei integriert wird und die Position der aktuellsten Version einnimmt.

Nach der Integration aller Dateien in das Projekt, wird dem Benutzer im oberen Bereich das Container-Archiv als "Importiert" markiert. Alle importierten Daten können daraufhin in der Container-Struktur des jeweiligen Projektes aufgerufen und eingesehen werden.

### 5.5.5 Differenzvisualisierung

Eine Software-Prüfung besteht im Regelfall aus mehreren iterativen Schritten, in denen unterschiedliche Software-Versionen vom Hersteller beim Prüfer eingereicht werden müssen. Ein Grund für eine neue Software-Version könnten Mängel in der Programmierung darstellen, die zu einer Gefährdung beteiligter Personen bei der Bedienung einer Maschine führen könnten. Aber auch Abweichungen von der formulierten Spezifikation könnten Gründe für Nachbesserungen durch den Hersteller und somit für neue Versionen sein. Alle diese Beispiele und Weitere führen schlussendlich zu einer Überarbeitung der Software oder auch der Spezifikation des zu prüfenden Objektes durch den Hersteller. Dieser muss anschließend eine neue Version der Dokumente und des Programm-Codes an den zuständigen Prüfer übergeben, der diese Informationen (Dokumente und Software) erneut auf Vollständigkeit und Korrektheit untersucht. Dieser Vorgang wird in so vielen Schritten wiederholt, bis das eingereichte Ergebnis des Herstellers den Anforderungen der Prüfer entspricht.

Ein großes Problem bei der Begutachtung aller eingereichten Dokumente durch des Herstellers besteht in der Identifikation der Änderungen zwischen den jeweiligen Versionen. Dies betrifft jegliche Änderungen, von der Konzeptbeschreibung, die unter anderem Anforderungsdokumente beinhaltet, bis hin zum Programm-Code, der auf den Maschinen eingesetzt werden soll. Weiterhin besteht dieses Problem in jedem zusätzlichen Iterationsschritt des Prüfprozesses, außer dem ersten initialen Schritt, bei dem eine erste Gesamtbegutachtung erfolgt.

Grundsätzlich muss ein Prüfer, zwischen den einzelnen Iterationen, die Änderungen in den betroffenen Dateien identifizieren und diese darauf prüfen, ob die beanstandeten Mängel der vorherigen Version durch den Hersteller behoben wurden. Erschwert wird dieser Prozess wenn ein Hersteller Änderungen, beispielsweise Modifikationen an der Spezifikation oder dem Programm-Code, durchgeführt und diese nicht dokumentiert oder dem Prüfer kommuniziert. Vor allem im Programm-Code können kleine Änderungen zu ganz anderen Ergebnissen führen, ggf. auch zu Gefährdungen von Menschen während des Betriebs einer Maschine.

Bei der Identifikation von Unterschieden, zwischen zwei Iterationen einer Software-Prüfung, ist aus diesem Grund eine maschinelle Überprüfung auf Veränderungen eine sinnvolle Möglichkeit Prüfer bei Ihrer Tätigkeit zu unterstützen. Durch eine automatisierte Änderungsüberprüfung entfällt der manuelle Abgleich den ein Prüfer durchführen müsste und dieser kann sich somit seiner eigentlichen Aufgabe, dem Prüfen der Hersteller-Software, intensiver widmen. Zum Einen fällt der Aufwand für die Identifikation von Änderungen geringer aus und zum Anderen könnten auch versteckte Änderungen, die vom Hersteller nicht dokumentiert wurden, identifiziert und dem Prüfer visualisiert werden.

Das automatisierte Bilden von Differenzen zwischen unterschiedlichen Versionen eines Dokumentes muss mit Hilfe geeigneter Werkzeuge erfolgen. Hierbei sollte es nebensächlich sein, um welche Dokumentart (Word, PDF, Code, ...) es sich bei der Differenzbildung handelt. Im Ergebnis muss dem Prüfer visualisiert werden, welche Dokumente Änderungen beinhalten und an welcher Stelle sich diese befinden.

Bei dem Vorgang der Differenzbildung von Dokumenten in unterschiedlichen Versionen unterstützt die Methode der Prüfsummenbildung mit Hilfe von berechneten Hash-Werten. Unter Einsatz von Hash-Werten kann effizient und mit annähernd hundertprozentiger Wahrscheinlichkeit vorhergesagt werden, ob zwei Versionen eines Dokumentes einen anderen Inhalt aufweisen, ohne hierfür aufwendige Vergleichsanalysen der Inhalte berechnen zu lassen. Erst wenn zwischen zwei Versionen auch unterschiedliche Prüfsummen berechnet werden können, liegt eine Änderung des Inhaltes vor, so dass eine Vergleichsanalyse angestoßen werden kann.

In diesem Zusammenhang wird die Vorgehensweise und der Nutzen von Hash-Algorithmen zur Bildung von Prüfsummen (siehe Kapitel 5.1.2.2) und der Einsatz geeigneter Werkzeuge zur Bildung von Differenzen (siehe Kapitel 5.1.2.3) und den möglichen Methoden zur Differenzberechnung (siehe Kapitel 5.1.2.4) herangezogen. Auf Basis dieser Informationen erfolgt eine Bewertung der Werkzeuge, die in einer Entwicklungsentscheidung resultiert, anhand derer eine prototypische Implementierung als SIWOB-Werkzeug erfolgt.

### Entwicklungsentscheidung

Die bisherigen Ansätze zur Differenzbildung werden nicht allen Anforderungen gerecht, da diese ausschließlich für den Differenzvergleich von Plain-Text Dokumenten, wie beispielsweise Programm-Code, entwickelt wurden und auch nur dort zum Einsatz kommen. Allerdings soll es auch möglich sein Dokumente, die komplexer als Plain-Text ausfallen, auf Differenzen analysieren zu können.

Basierend auf den Analysen der Ansätze zur Differenzbildung von Dokumenten in verschiedenen Versionen und der Auswertung möglicher Verfahren fiel deshalb die Entscheidung für deshalb eine individuelle Entwicklung eines Differenz-Tools in Form eines Prototypen.

Diese berücksichtigt die grundlegenden Ansätze der Differenzbildung zwischen zwei Dokumenten aus den zuvor analysierten Werkzeugen und Methoden. Insbesondere die Algorithmen zum Vergleich von Zeichenketten spielen bei der Entwicklung des Prototypen eine entscheidende Rolle. Aber auch die Darstellung der Differenzen für den Anwender, in einer geeignet formatierten Ausgabe, darf nicht vernach-

lässigt werden, damit dieser nicht unnötig viel Zeit aufwenden muss, um Differenzen in Dokumenten ermitteln zu können.

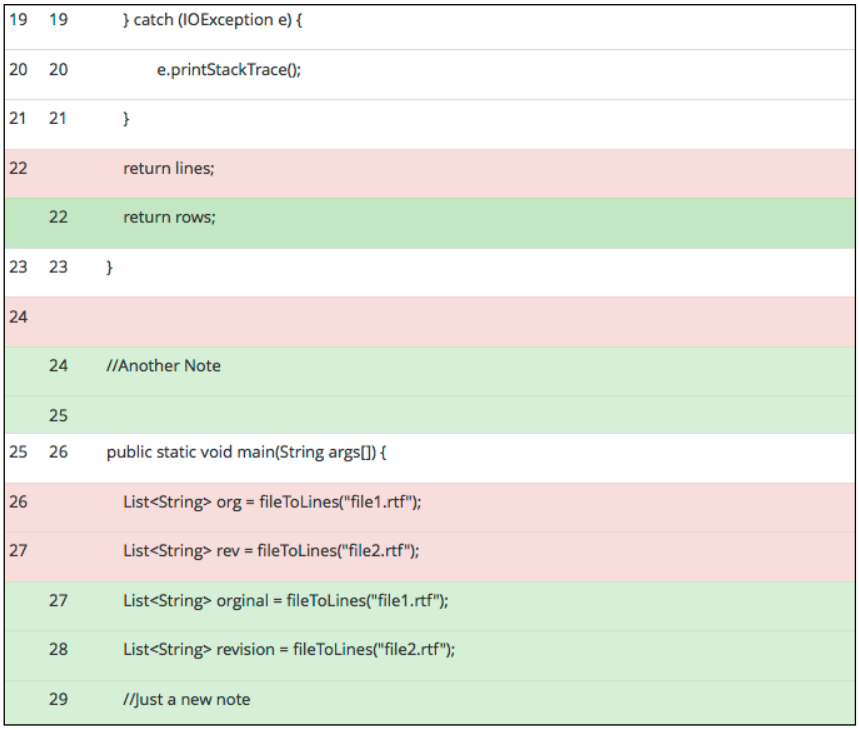
Die Auswertung der Werkzeuge im vorherigen Kapitel zeigt dazu eine signifikante Gemeinsamkeit in der Berechnung der Differenzen auf. Alle Werkzeuge Implementieren in ihrer Kernfunktion den Myer's Diff Algorithmus zur Berechnung von Differenzen zwischen zwei Plain-Text Dokumenten. Eine logische Schlussfolgerung hieraus ist es, diesen Algorithmus als Basis für die eigene Individuelle Entwicklung zu adaptieren, indem dieser entweder in einer bestehenden Library aufgegriffen oder anhand der Dokumentation von Myer selbst implementiert wird.

Aus diesem Grund sieht eine Neuentwicklung im Kern die Implementierung der Differenzberechnung vor, mit der jegliche Zeichenketten verglichen werden können. Die weiteren Entwicklungsaufgaben bestehen darin die Zeichenketten aus verschiednen Quellen zu extrahieren und die berechneten Differenzen für eine visuelle Ausgabe aufzuarbeiten. Hierzu können gegebenenfalls wiederum Best Practice Ansätze aus den evaluierten Werkzeugen als Grundlage dienen.

Das Ziel der Entwicklung stellt den Vergleich von stark strukturieren Dokumenten dar, die beispielsweise Code-Strukturen beinhalten. Weiterhin soll es möglich sein Inhalte einer Spezifikation zu vergleichen, die auf dem in SIWOB entwickelten Format basieren. Hierbei ist zu berücksichtigen, dass neben den menschenlesbaren Inhalten, die mittels PDF-Viewer angezeigt und gelesen werden können, auch maschinell lesbare Strukturen in das Format integriert sind. Diese Strukturen stellen die Grundlage für die Generierung des menschenlesbaren Inhaltes dar und sind für den Leser nicht sichtbar, können jedoch für einen effizienten Vergleich der Inhalte verwendet werden. Deshalb wird zur Vereinfachung der prototypischen Entwicklung im ersten Schritt eine zeilenbasierte Lösung erarbeitet, um derartige Dokumente miteinander vergleichen zu können.

Der Ansatz eines zeilenbasierten Vergleichs bildet in dieser Entwicklungsphase den besten Kompromiss aus Effizienz und Realisierbarkeit, da strukturierte Informationen relativ leicht eingelesen und ausgewertet werden können. Insbesondere für Code-Dokumente stellt die Entwicklung dieser Methode das Mittel der Wahl dar, um einzelne Code-Dokumente in verschiedenen Versionen miteinander vergleichen und die berechneten Ergebnisse angemessen visualisieren zu können. Dies gilt insbesondere auch für die Inhalte der SIWOB-Formate, die mithilfe passender Werkzeuge automatisiert generiert werden.

Grundsätzlich kann mit dem zeilenbasierten Ansatz jedes einfache Plain-Text Dokument für einen Vergleich verwendet werden, wodurch Dokumente mit Programm-Strukturen (C-Dateien etc.) und einfache Textinformationen abgedeckt werden. Damit ein Vergleich möglich ist, dürfen diese Dokumente keine unbekannten Zeichenzusatzinformationen (Header-Informationen) enthalten, sondern ausschließlich Textinformationen aufweisen. Sofern diese Bedingungen erfüllt sind, kann ein Vergleich auf Zeilenebene erfolgen und eine Ausgabe im SIWOB-Frontend generiert werden (siehe Abbildung 59).



| | | |
|---|---|---|
| 19 | 19 | } catch (IOException e) { |
| 20 | 20 | e.printStackTrace(); |
| 21 | 21 | } |
| 22 | | return lines; |
| | 22 | return rows; |
| 23 | 23 | } |
| 24 | | |
| | 24 | //Another Note |
| | 25 | |
| 25 | 26 | public static void main(String args[]) { |
| 26 | | List<String> org = fileToLines("file1.rtf"); |
| 27 | | List<String> rev = fileToLines("file2.rtf"); |
| | 27 | List<String> orginal = fileToLines("file1.rtf"); |
| | 28 | List<String> revision = fileToLines("file2.rtf"); |
| | 29 | //Just a new note |

Abb. 59: Zeilenbasierte Differenzberechnung und Visualisierung im SIWOB-Frontend

Für die Berechnung einer Ausgabe müssen zuvor zwei Dokumente ausgewählt werden, die auf Veränderungen überprüft werden sollen. Als Ergebnis wird eine tabellarische Ansicht mit mehreren Spalten generiert, bei der in den ersten beiden Spalten die Zeilenzahlen der beiden Dokumente angezeigt werden. In der linken Spalte sind die Zeilenzahlen des ersten Dokumentes und in der zweiten Spalte die Zeilennummern des zweiten Dokumenttest zu erkennen. Sollten im Zieldokument Zeilen entfernt oder hinzugefügt worden sein, kann anhand dieser Zeilennummern eine Abweichung leicht lokalisiert werden. Unveränderte Inhalte werden durch keinerlei Markierungen hervorgehoben (weiße Zeilen). Bei Änderungen hingegen, die durch Hinzufügen (grüne Zeilen) oder Entfernen entstehen (rote Zeilen) wird dies durch eine entsprechende Farbe hervorgehoben. Zeilen die in einer Änderung zusammengehören werden zusammengefasst (siehe Zeile 26-27 in rot oder Zeile 27-29 in grün) und in farblichen Blöcken gebündelt, sodass eine Zuordnung erleichtert wird.

Ein Vergleich von komplexen Text-Dokumenten, die von den entwickelten Formaten (SIWOB-Format) oder einfachen Plain-Text Dokumenten (C-Dokumente) abweichen, wie beispielsweise Word-Dokumente, lassen sich nicht oder nur mit deutlichen Einschränkungen abbilden. Dies begründet sich insbesondere darin, dass unbekannte Formate mit vielen Header-Informationen oder in unterschiedlichen Zeichensatzformaten (Zeichensatzkodierung, Formatierungsinformationen etc.) vorliegen können, die nur von entsprechende Zielprogrammen interpretierbar sind. Eine Berücksichtigung solcher Dokumente würde in einem erheblichen Entwicklungsaufwand münden und ist deshalb nicht Bestandteil der prototypischen Entwicklung.

## Weitere Entwicklungsschritte

An erster Stelle für weitere Entwicklungsschritte stehen umfangreiche Tests, mit Hilfe derer die Robustheit und Fehlerfreiheit der programmierten Algorithmen, zu Berechnung und Visualisierung der Differenzen, nachgewiesen werden muss. Im Rahmen des Projektes wurden kleine Test durchgeführt, die jedoch kein Nachweis auf die gewünschte Fehlerfreiheit darstellen. Ein Testverfahren muss deshalb durchgeführt und kann in drei Schritte aufgeteilt werden, sodass am Ende der Testprozesse ein konsistentes Werkzeug entsteht.

Die erste Testphase zielt auf den Nachweis der Robustheit des entwickelten Algorithmus ab. Es muss nachgewiesen werden, dass auch zuverlässig alle Änderungen zwischen zwei Dokumentversionen identifiziert werden. Hierzu ist es notwendig gezielt manipulierte Dateien miteinander zu vergleichen, sodass mögliche Fehlberechnungen eindeutig zugeordnet werden können. Eine Überprüfung auf die korrekte Berechnung der Zeilennummern und der Abbildung hinzugefügter, gelöschter oder unveränderter Zeilen stehen im Fokus. Die Bündelung von zusammenhängenden Zeilen in Blöcke muss in dieser Testphase gesondert berücksichtigt werden. Erst wenn zuverlässig garantiert werden kann, dass Änderungen erkannt und dargestellt werden, kann mit der zweiten Testphase begonnen werden.

Die zweite Testphase zielt auf einen Vergleich von großen Dokumentinhalten ab. Dies können beispielsweise Code-Dokumente sein, die aus mehreren tausend Zeilen Code bestehen, oder umfangreichen Spezifikationsdokumente, die mit dem passenden SIWOB-Werkzeug erstellt wurden. Die Ergebnisse müssen dahingehend überprüft werden, ob Dokumente mit umfangreichen Inhalten vollständig verarbeitet werden. Hierbei könnten Probleme in der Speicherverwaltung dazu führen, dass Dokumente nicht vollständig verarbeitet werden. Hierbei könnte es zu einer Ausgabe von Teilergebnissen oder einer leeren Ausgabe kommen. Sofern die Robustheit nachweisbar ist, kann mit der dritten Testphase begonnen werden.

Die dritte Testphase zielt auf verschiedene Dokumentarten oder -formate ab. Hierbei müssen verschiedenartige einfach strukturierte Dokumente ohne Header-Informationen verglichen werden, wie beispielsweise C-Dokumente, C++-Dokumente, Plain-Text Dokumente etc., um nachzuweisen, dass das Werkzeug die verschiedenen Dokumente fehlerfrei verarbeiten kann. Dies gilt auch für das speziell entwickelte Dokumentformat zur Abbildung von Spezifikationen.

Für alle identifizierten Probleme muss eine Lösung erarbeitet werden, die die prototypische Entwicklung verbessert und zu einer Einsatzbereitschaft führt. Im Anschluss an die Verbesserung erfolgt wiederum eine Testphase, die belegen muss, dass die realisierten Lösungen alle gefundenen Probleme beseitigt haben. Sofern alle Probleme behoben sind, könnte das fertiggestellte Werkzeug in einer ersten Version produktiv verwendet werden.

In weiteren Schritten kann das Werkzeug zur Differenzberechnung um zusätzliche Funktionen (nachfolgend beschrieben) erweitert werden, mit der die Funktionalität verbessert werden kann, wobei anschließend jeweils eine Testphase angeknüpft werden muss.

Eine nützliche Verbesserung stellt eine Differenzberechnung auf Wortebene dar, wodurch eine detaillierte Visualisierung von Änderungen zwischen zwei Dokumentversionen möglich wird. In Dokumenten mit

Inhalten im Fließtextformat, wie beispielsweise der Spezifikation, kann durch eine Differenzbildung auf Wortebene eine detailliertere visuelle Ausgabe berechnet werden, sodass Änderungen leichter identifiziert werden können. Diese Verbesserung ist in erster Linie bei Dokumenten mit Fließtext sinnvoll, da sich hier ein Absatz über viele Zeilen erstrecken kann. Wohingegen der Nutzen in Code-Dokumenten nicht so signifikant wäre.

Neben dem wortbasierten Vergleich stellt die Reduzierung der Ausgabe auf die reinen Änderungen zwischen den Dokumenten eine mögliche Verbesserung dar. Hierbei könnte nach der Differenzberechnung nur der geänderte Inhalt im SIWOB-Frontend visualisiert werden, wodurch der Leser eine kompakte Ansicht erhält. Das aufwändige Suchen nach Änderungen, insbesondere bei sehr umfangreichen Dokumenten mit vielen Seiten und Zeilen, würde durch eine reduzierte Visualisierung erheblich vereinfacht werden.

Weiterhin würden rein statistische Informationen über die Differenzberechnung eine Verbesserung darstellen, mit deren Hilfe zusätzlich eine kompakte Übersicht erstellt werden könnte, wie beispielsweise die Anzahl der Seiten oder Zeilen, die Anzahl von entfernten oder hinzugefügten Zeilen/Worten und weitere nutzbringende Informationen. Diese statistischen Informationen stellen ein schnelles Kontrollmittel dar, anhand dessen ein Leser ohne große Umwege Auskunft darüber erhalten könnte, ob und wie stark sich ein Dokument zwischen den gewählten Versionen verändert hat.

Zuletzt wäre eine Erweiterung des Vergleichs auf weitere Dokumentformate denkbar, wie etwa PDF-, Word-Dokumente oder auch gänzlich andere Formate. Bei komplexen Formaten besteht die Schwierigkeit darin die nutzbringenden Inhalte zu separieren, um diese anschließend analysieren zu können. Zumeist bestehen komplexe Dokumente (Word etc.) neben dem Textinhalt aus vielen Meta-Informationen, wie allgemeinen Header-Informationen, Formatanweisungen, Textkodierungen etc, die bei einer Differenzberechnung zu fehlerhaften Ausgaben führen würden. Hierzu müsste, mit Hilfe einer (ausführlichen) Dokumentation seitens des Dokument-Entwicklers, sofern diese verfügbar ist, das Format geeignet fragmentiert und der notwendige Textinhalt für einen Vergleich extrahiert werden.

Eine weitere Herausforderung stellen darüber hinaus Dokumente dar, die mit Passworten geschützt oder sogar verschlüsselt sein könnten. Diese Problematik könnte mit Hilfe geeigneter Verfahren zu dem Schlüsselaustausch gelöst werden, auf die sich Hersteller und Prüfer zuvor einigen müssten.

### 5.5.6 Comment System

Während einer Software-Prüfung entsteht in der Regel für den Prüfer die Notwendigkeit eingereichte Dokumente kommentieren zu müssen, um beispielsweise fehlerhafte Inhalte zu markieren und zu dokumentieren. Aber auch klärende Anmerkungen oder Fragen über unklare Formulierungen werden in Kommentarform festgehalten.

Grundsätzlich besteht die Notwendigkeit zu jeder einzelnen Datei einer Baumusterprüfung Kommentare notieren zu können, wie etwa Quellcode-Dateien oder die Systemspezifikation, aber auch Grafiken und Bauzeichnungen. Diese Kommentierungen des Prüfers dienen insbesondere dazu Fehler, Anmerkungen

oder Fragen schriftlich festzuhalten, damit ein Hersteller auf dieser Basis Verbesserungen an seinem Produkt und der Dokumentation durchführen kann.

Es muss folglich die Möglichkeit bestehen mit Hilfe von passenden Werkzeugen Kommentare zu Dateien verfassen und diesen eindeutig zuordnen zu können. Im Ergebnis müssen dem Prüfer und Hersteller diese Kommentare visualisiert werden, damit eine einheitliche Diskussionsgrundlage besteht, sollten die Kommentierungen zu Fragen führen, die einer persönlichen Aufklärung bedürfen.

### Konzept- und Anforderungsentwicklung

Der erste Schritt zur Konzeptentwicklung für das Kommentarsystem ist die Analyse der vorhandenen Systemstruktur. Dabei soll insbesondere erarbeitet werden, wie das Kommentarsystem in die vorhandene Systementwicklung eingebettet werden kann. Zu diesem Zweck wird die Systemstruktur von SIWOB im Groben betrachtet. Dadurch soll ein allgemeiner Überblick über die wichtigsten Komponenten und das Zusammenwirken dieser geschaffen werden.

Abbildung 60 zeigt den schematischen Aufbau von SIWOB zum Zeitpunkt der Entwicklung des Kommentarsystems. Das zentrale Element stellt das BUS-System dar, über das alle Kommunikation zwischen den Services erfolgt. An den BUS sind verschiedene Services angebunden, wie beispielsweise das Repository oder das Traceability-Tool, die den internen Bereich darstellen und für den Benutzer nicht direkt erreichbar sind. Darüber hinaus ist das User Interface für den Prüfer hervorzuheben, das die Schnittstelle zum Benutzer darstellt und ihm eine grafische Oberfläche präsentiert, mit der er arbeiten kann.

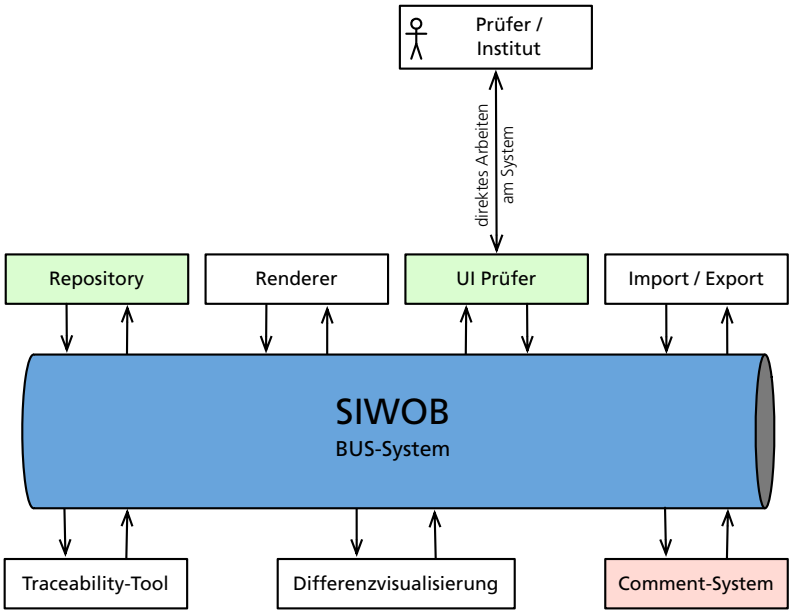



Abb. 60: SIWOB Aufbau

Die Entwicklung eines Kommentarsystem (Comment-System) soll einen neuen Web Service bereitstellen (rot hervorgehoben), der an den BUS angeknüpft wird. Damit das Kommentarsystem realisiert werden kann, ist es außerdem notwendig Informationen zu speichern und dem Benutzer eine Interaktion zur

Ein- und Ausgabe von Kommentaren anzubieten. Deshalb müssen für die Entwicklung eines neuen Plugin zur Kommentareingabe das Repository sowie das User Interface des Prüfer (grün hervorgehoben) in die Entwicklung einbezogen werden. Im Web Service des Comment-System wird die gesamte Programmlogik des Kommentarsystems realisiert. Hier werden die verfassten Kommentare entgegengenommen und verarbeitet, die daraufhin im Repository abgespeichert werden. Das User Interface liefert eine grafische Oberfläche mit der beispielweise ein Prüfer seine Kommentare zu einer Datei eingeben kann. Das BUS-System dient als zentraler Vermittler zwischen diesen Services und verbindet die einzelnen Teilentwicklungen miteinander, sodass Kommentare formuliert, gesendet, verarbeitet und abgespeichert werden können.

Die Realisierung des Kommentarsystems gliedert sich in zwei Teile auf. Im ersten Teil muss ein Web Service (Comment-System) vollständig neu konzipiert werden, der Kommentare entgegennimmt bzw. zurückgibt sowie die gesamte Modullogik des Kommentarsystems beinhaltet. Der zweite Teil ist für die Ausgabe und Präsentation der Informationen zuständig und dient als Schnittstelle zwischen User und System. Hierzu muss der Web Service für das User Interface (siehe Kapitel 5.5.3) des Prüfers angepasst werden. Das Repository bedarf keiner weiteren Änderungen, da die bestehende Programmlogik des Web Service zum Speichern oder Abrufen von Informationen verwendet werden kann.

1. Web Service (Comment-System)

Der Web Service ist dafür zuständig, Befehle entgegenzunehmen, diese auszuführen und Ergebnisse zurückzugeben. Dabei ist im Vorfeld wichtig zu klären, welche Funktionen von diesem für das Kommentarsystem bereitgestellt werden müssen. Die Hauptfunktion des Systems ist die Verwaltung von Kommentaren. Unter Verwaltung sind folgende Funktionen zusammengefasst:

- Sicherung der Konsistenz
- Import/Export mehrerer Kommentare
- Zuordnung von Kommentaren zu Benutzern

Sicherung der Konsistenz bedeutet, dass Kommentare zuverlässig zu Dateien bzw. Ordnern zugeordnet werden können, auch wenn diese zum Beispiel verschoben werden. Dies ist wichtig, damit keine Kommentare während der Benutzung von SIWOB verloren gehen und damit wichtige Informationen nicht wahrgenommen werden.

Die Import/Export Funktionalität beschreibt Funktionen, die beim Import/Export von Projekten benötigt werden. Mit Import/Export ist gemeint, dass es möglich ist, in SIWOB ganze Projekte zu exportieren oder zu importieren. Dabei sollen die vorhandenen Kommentare mit übertragen und richtig zugeordnet werden.

Weiterhin ist die Zuordnung zu Benutzern sehr wichtig. Es muss möglich sein zu erkennen, welcher Benutzer einen Kommentar hinzugefügt hat. Dabei könnten Kommentare von Entwicklern oder Prüfern eingetragen werden.

### 2. Frontend (User Interface)

Da die Logik für das Kommentarsystem im Web Service eingebettet ist, bestehen für das Frontend eher Anforderungen an die Usability als an die Funktionalität. Das Frontend muss dabei lediglich die Informationen darstellen und die Funktionalität zum Einfügen von neuen Kommentaren bieten.

- Aufgabenangemessenheit

  Im Frontend soll es möglich sein neue Kommentare einfügen zu können. Weiterhin sollen hier auch die bereits vorhandenen Kommentare in geeigneter Form dargestellt werden. Wichtig dabei ist die Art der Präsentation, damit der Nutzer diese einfach einsehen und intuitiv erfassen kann. Neben der Funktionalität Kommentare einfügen und ansehen zu können, soll der Nutzer auch selbst geschriebene Kommentare verstecken können. Ein Löschen wird nicht unterstützt, damit Kommentare nicht verloren gehen können.

- Selbstbeschreibungsfähigkeit

  Der Aufbau soll so gestaltet werden, dass der Nutzer intuitiv das System bedienen kann. Das heißt, dass keine Einarbeitung notwendig ist, um mit dem Kommentarsystem arbeiten zu können. Weiterhin sollen Rückmeldungen den Nutzer bei der Benutzung unterstützen. Dabei soll er auf Fehler oder andere Ereignisse aufmerksam gemacht werden. Zum Beispiel soll er bei einer unkommentierten Datei darauf hingewiesen werden, dass diese keine Kommentare besitzt. Eine leere Seite würde gegebenenfalls den Eindruck eines Fehler vermitteln.

- Steuerbarkeit

  Dieser Teil der Usability ist für das Frontend nicht stark von Bedeutung, da dieses keine dialogbasierten Eingaben enthält.

- Erwartungskonformität

  Der Aufbau soll an andere verbreitete Systeme angepasst sein. Ein Beispiel für den Aufbau einer Kommentarseite kann YouTube sein. Dort gibt es zwei verschiedene Arten die Kommentare zu sortieren, "Top-Kommentare" und "Neuste zuerst". Für das SIWOB Kommentarmodul fällt die Möglichkeit von Top-Kommentaren weg, da es nicht möglich ist Kommentare zu bewerten.

- Fehlertoleranz

  Neben den Rückmeldungen an den Nutzer sollen Fehler und Falscheingaben abgefangen werden. Dabei ist es wichtig, dass selbst bei einem fehlerhaften Speichervorgang eines Kommentars nicht das ganze System abstürzt.

- Individualisierbarkeit

  Durch ein festes Design, welches unter dem Punkt "Erwartungskonformität" bereits kurz beschrieben wurde, entfällt eine Individualisierbarkeit des Frontends.

- Lernförderlichkeit

  Da die Funktionalität des Frontends sehr begrenzt bleibt, wird auf die Lernförderlichkeit nicht weiter

eingegangen. Durch ein bekanntes Design des Kommentarmoduls soll eine Nutzung ohne große Einarbeitung geschaffen werden.

### 5.5.7 Spezifikationsgenerierung

Für die Generierung von Software-Spezifikationen wurde ein prototypisches Werkzeug konzipiert, welches Benutzer bei der Erstellung, Bearbeitung und Betrachtung unterstützt. Insbesondere werden Hersteller bei der Eingabe von spezifikationsrelevanten Daten unterstützt. Weiterhin bietet das Werkzeug die Möglichkeit die maschinenlesbaren Inhalte, der Spezifikationen im Hybrid-Format, auf dem User Interface zu visualisieren. Zusätzlich werden die darin enthaltenen Meta-Daten für den Prüfer in die Ausgabe integriert, sodass dieser eine erweiterte Ansicht zur menschenlesbaren Version erhält.

#### Konzeption- und Anforderungsentwicklung

Folgende Anforderungen an ein Format zur Darstellung von Spezifikationen konnten anhand des entwickelten Qualitätsindex (siehe Kapitel 5.3) identifiziert werden und wurden für die Erstellung eines passenden Konzeptes zu Grunde gelegt:

| | |
|---|---|
| Dokumentbasiert | Das Spezifikationsformat sollte in Form eines digitalen Dokuments vorliegen, das alle Informationen einer Software-Spezifikation aufnehmen kann. |
| Menschenlesbar | Das Spezifikationsformat soll mit Hilfe von verfügbarer Software in einer für einen Menschen lesbaren Form (z.B. in der digitalen Version eines Din A4 Dokuments) zur Verfügung stehen und sich auch auf Papier ausdrucken lassen. |
| Unveränderbar | Das Spezifikationsformat soll nicht versehentlich während einer Begutachtung verändert werden können. Ausserdem soll sichergestellt sein, dass sich variable Felder nicht beim Öffnen eines Dokuments verändern. |
| Maschinenlesbar | Alle strukturellen Elemente der repräsentierten Software-Spezifikation (Gliederungspunkte, Inhalte etc.) sollen einzeln zugreifbar in einer maschinenlesbaren Version vorhanden sein. Damit soll die prinzipielle Fähigkeit zum Export gegeben werden. |
| Hilfestellung | Insbesondere unerfahrene Nutzer sollen bei der Erstellung und Bearbeitung von Spezifikationen durch die Vorgabe von sinnvollen Strukturelementen unterstützt werden. |
| Prüfbarkeit | Eine Prüfung von Dokumenten im vorliegenden Spezifikationsformat soll vereinfacht werden. |
| Sicherheitsrelevanz | Dokumentabschnitte, die besondere Relevanz bei der Sicherheitsprüfung haben, sollen markiert werden können. |

#### Evaluation von PDF-Frameworks

Für die Nutzung der Spezifikationsgenerierung in SIWOB wurde ein hybrides Dokumentformat entworfen, das ein menschenlesbares (PDF-Dokument) und ein maschinenlesbares (XML-Dokument) Format

enthält. Deshalb erfolgt zunächst ein Vergleich etablierter PDF-Frameworks, welche den notwendigen Funktionsumfang zur PDF-Generierung bereitstellen, mit deren Hilfe die Generierung eines menschenlesbaren PDF-Dokumentes realisiert werden kann.

Tabelle 15 enthält eine Auswahl verschiedenster Frameworks sowie wichtige Kriterien, welche für die Entwicklung in SIWOB entscheidend sind. Da die Entwicklung von SIWOB und von dessen Werkzeuge auf Basis von Java EE erfolgt, wurden ausschließlich Frameworks evaluiert, die auf der Programmiersprache Java basieren und nicht veraltet sind.

| | iText 5.0+ | iText 4 | PDF Box | FOP |
|---|---|---|---|---|
| Aktualität | V5.5.1 | V4.2.1 | V1.8.6 | V1.1 |
| Release Datum | 05-2014 | 07-2013 | 06-2014 | 10-2012 |
| Framework Umfang | sehr groß | groß | mittel | gering |
| Programmiersprache | Java | Java | Java | Java |
| Dokumentation? | Web/Buch | Web/Buch | eingeschränkt | kaum |
| API vorhanden? | JA | JA | JA | NEIN |
| PDF Generierung möglich? | JA | JA | JA | JA |
| Weiterentwicklung? | JA | NEIN | JA | n.A. |
| Lizenzmodell | AGPL | LGPL | Apache v2.0 | Apache v2.0 |

Tabelle 15: Vergleich verschiedener Frameworks zur Generierung von PDF-Dokumenten

Ein unmittelbares Ausschlusskriterium für die Verwendung innerhalb des Forschungsprojekts ist ein geringer bis mittlerer Framework Umfang. Dieses Kriterium bestimmt maßgeblich das Spektrums bereitgestellter Funktionen zur PDF-Generierung. Hierbei müssen nicht nur einfache Textinhalte oder Abbildungen in ein PDF-Dokument überführt, sondern auch komplexe Strukturen, wie etwa Tabellen, Kopfzeilen, Fußzeilen oder Seitenzahlen abgebildet werden können. Neben diesen menschenlesbaren Elementen, müssen auch maschinenlesbare Elemente in ein PDF-Dokument einfügbar sein, sodass mindestens die Integration von Anhängen möglich sein muss.

Die Produkte von PDF Box und FOP werden deshalb bereits durch den nur geringen bis mittleren Framework Umfang ausgeschlossen. Daraufhin stehen iText in Version 4, sowie ab Version 5.0, für den Einsatz in SIWOB zur Verfügung. Überdies stellt die Verwendung proprietärer Lizenzmodelle ein weiteres Ausschlusskriterium dar. Basierend auf dem Lizenzmodell könnten grundsätzlich beide PDF-Frameworks genutzt werden, da die Verwendung der Code-Bibliotheken kostenfrei ist. Jedoch sieht die AGPL eine Einschränkung im Vertrieb von Software vor, die diese Bibliotheken verwenden, durch die es zwingend erforderlich wird eine kommerzielle Lizenz von iText 5 zu erwerben, sofern das Produkt im kommerziellen Bereich vertrieben werden soll.

Aufgrund der strengeren Restriktion in dem Lizenzmodel AGPL wurde entschieden, iText in Version 4.2.1 zu verwenden. Da iText 5 eine Weiterentwicklung von iText 4 darstellt, ist eine Funktionsdeckung zwi-

schen diesen beiden Versionen weitestgehend vorhanden, sodass die prototypische Entwicklung in iText 4 erfolgen kann.

## Prototypische Entwicklung

Nachdem etablierte PDF-Frameworks zur Generierung eines PDF-Dokumentes in menschenlesbarer Form evaluiert wurden und iText 4 für den Einsatz in SIWOB als Ergebnis resultiert, erfolgt die prototypische Entwicklung des Werkzeuges zur Spezifikationsgenerierung. Dieses Werkzeug soll gewährleisten, dass sämtliche Informationen einer Spezifikation im menschenlesbaren (PDF-Dokument) und maschinenlesbaren (XML-Dokument) Format gespeichert werden. Anschließend wird das XML-Dokument in das generierte PDF-Dokument integriert, wodurch das spezifizierte Hybrid-Format entsteht

Damit sichergestellt werden kann, dass beide Dokumente zu jeder Zeit identische Inhalte aufweisen, wurde ein passendes prototypisches Werkzeug konzipiert. Dieses Werkzeug soll gewährleisten, dass eine inhaltliche Änderung in der Spezifikation immer zu einer Änderung in beiden Dokumenten (PDF und XML) führt.

Die Entwicklung dieses Werkzeuges setzt als Grundlage die konzipierten Datenstrukturen des Dokumentformates voraus. Hierzu zählt insbesondere die XML-Struktur einer Spezifikation, auf deren Basis geeignete Java-Klassen generiert wurden, um diese Strukturen mit Inhalten zu versehen.

### User Interface zur Spezifikationsgenerierung

Grundsätzlich bestehen die Inhalte der XML-Struktur aus Eingaben der Benutzer. Sämtliche Informationen, aus denen eine Spezifikation besteht, werden hierzu über eine GUI innerhalb des SIWOB User Interfaces eingegeben, wie in Abbildung 61 zu erkennen ist. Diese spezielle GUI bietet Eingabemöglichkeiten für alle Elemente einer Spezifikation, gemäß des in Kapitel 5.4.2.2 vorgestellten Schemas.

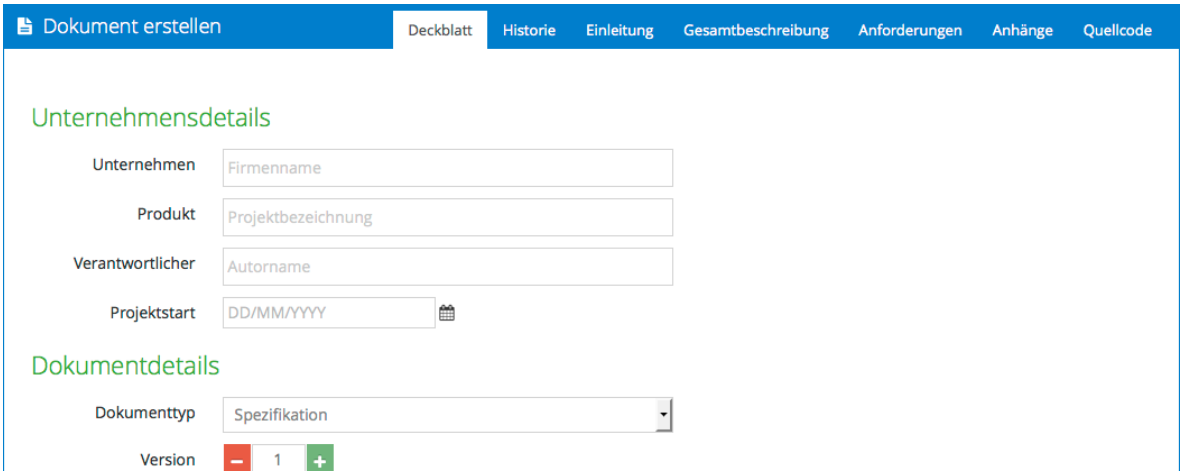



Abb. 61: GUI im User Interface für die Generierung einer Spezifikation

Für die Informationserhebung wird ein Prozess initiiert, der aus mehreren Arbeitsschritten (Kategorien) zur Informationseingabe (Deckblatt, Historie, Einleitung usw.) besteht und in jedem Fall in einem SIWOB-Spezifikations-PDF endet. Allgemein beschrieben, wird ein maschinenlesbares XML-Dokument generiert,

das im Anschluss in ein Java-Objekt mit identischem Aufbau überführt wird. Dieses Java-Objekt enthält alle eingegebenen Informationen und kann durch eine Java-Klasse zur Generierung eines formatierten PDF-Dokuments im DIN-A4 Format genutzt werden. Nachdem dieses menschenlesbare Dokument vorliegt, wird in dieses das generierte (maschinenlesbare) XML-Dokument als Anhang beigefügt, wodurch das Hybrid Format entsteht.

Auf diese Weise entsteht bei jedem Abspeichern einer Spezifikation ein SIWOB-Spezifikations-PDF, das beide Anteile synchron enthält. Soll eine vorhandene Spezifikation fertiggestellt oder überarbeitet werden, so kann für einen Hersteller das SIWOB-PDF in der GUI geöffnet werden, wobei lediglich die Informationen der enthaltenen XML-Datei angezeigt werden. Wird das PDF-Dokument jedoch mit einem PDF-Viewer (wie beispielsweise Adobe Acrobat) geöffnet wird ausschließlich der menschenlesbare und formatierte Teil sichtbar. Darüber hinaus erhalten Prüfer, ebenfalls über die GUI, Zugriff auf den XML-Anteil eines SIWOB-PDFs, wobei keinerlei Möglichkeiten bestehen, inhaltliche Änderungen vorzunehmen. Er kann lediglich Kommentare zu beliebigen Elementen des Dokuments hinzufügen.

Weiterhin bietet das konzipierte User Interface (GUI) zur Spezifikationsgenerierung zahlreiche Funktionen, die den Benutzer bei der Eingabe von Informationen und der Erstellung einer Spezifikation unterstützen. Abbildung 62 zeigt in diesem Zusammenhang das User Interface für den Arbeitsschritt zur Informationseingabe des Deckblattes.

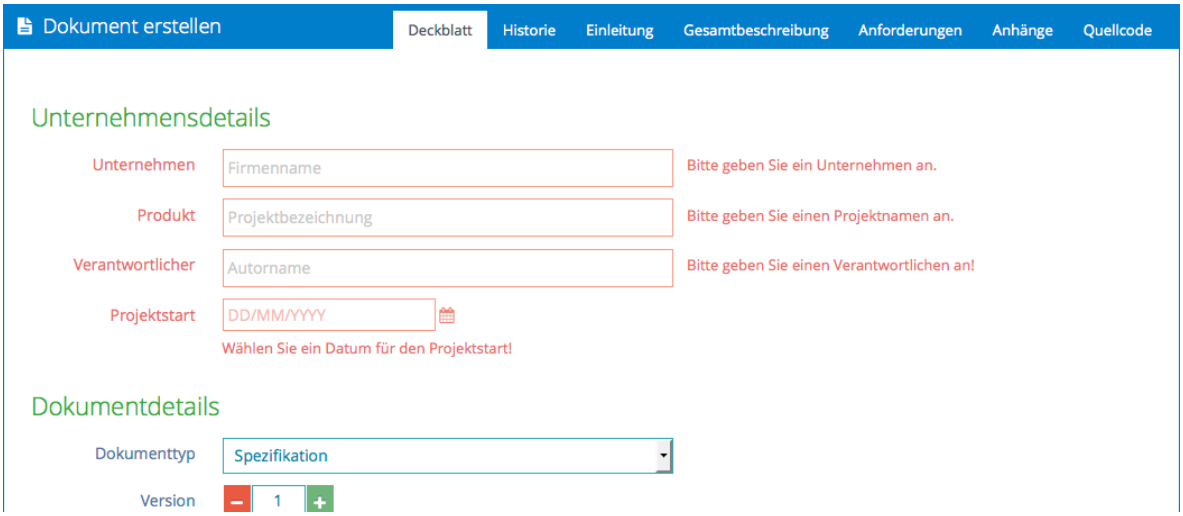



Abb. 62: User Interface zur Eingabe von Spezifikationselementen mit Vollständigkeitsprüfung

Grundsätzlich sind die verschiedenen Eingabeelemente ohne eine Hervorhebung in das User Interface integriert (siehe Abbildung 61), in das der Benutzer seine Informationen eintragen kann. Bei fehlenden Eingaben wird der Benutzer auf diese Probleme hingewiesen, beispielsweise wenn das Dokument gespeichert oder zwischen den verschiedenen Kategorien (Deckblatt, Historie, etc.) gewechselt werden soll.

Jedes Eingabeelement mit fehlenden Inhalten wird dem Benutzer in roter Farbe hervorgehoben und schließt die Bezeichnung des Elementes auf der linken Seite mit ein. Zusätzlich zur Hervorhebung wird ein passender Hinweisdialog zu jedem fehlerhaften Eingabeelement ausgewiesen. Diese Ausgabe erfolgt

meist auf der rechten oder unteren Seite des Elementes. Dadurch entsteht eine örtliche Nähe von Fehler und Eingabe, sodass eine Zuordnung für den Benutzer erleichtert wird.

Ein Benutzer erhält dadurch nicht nur ein visuelles farbliches Feedback, sondern gleichzeitig eine beschreibende Anweisung, wie das identifizierte Problem gelöst werden kann. Diesbezüglich können die beschreibenden Anweisungen zu jedem Eingabeelement angepasst werden, sodass im Fehlerfall für jedes einzelne Element eine individuelle Anweisung visualisiert werden kann.

Mit dieser Technik kann eine formale Vollständigkeitsprüfung realisiert werden, die es erlaubt den Benutzer auf fehlende Eingaben hinzuweisen. Eine inhaltliche Analyse der Benutzereingaben ist nicht möglich, da diese in Textform erfolgt und nur mit entsprechendem Fachwissen eines Prüfers verifiziert werden kann. Die Vollständigkeitsprüfung kann von Prüfern und Herstellern dazu verwendet werden eine Spezifikation automatisiert auf formale Vollständigkeit zu überprüfen. In der prototypischen Fassung beschränkt sich die Vollständigkeitsprüfung mit Fehlerausgaben auf die Eingaben für das Deckblatt einer Spezifikation und könnte ganzheitlich auf eine Spezifikation angewendet werden.

Zusätzlich zur Vollständigkeitsprüfung wird ein Benutzer bei der Eingabe von umfangreichen Textpassagen unterstützt. In diesem Zusammenhang zeigt Abbildung 63 den Arbeitsschritt zur Informationseingabe des User Interfaces für die Einleitung einer Spezifikation. Diese zeigt verschiedene Elemente für die Eingabe von formatierten Textinhalten, wie beispielsweise unter der Überschrift "1. Beschreibung" oder "1.1 Zweck" gezeigt wird.

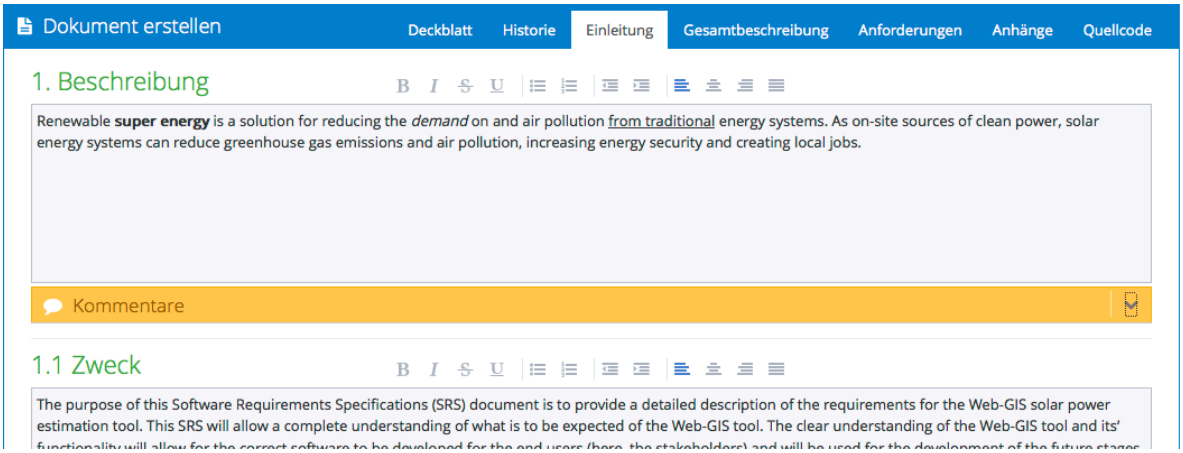



Abb. 63: Unterstützung von Texteingaben mit Hilfe eines WYSIWYG-Editor

Eingabeelemente für die umfangreiche Textinhalte zu erwarten sind, wurden um die Technologie eines WYSIWYG-Editors erweitert. Der Benutzer erhält dadurch die Möglichkeit Textinhalte durch Formatierungen anreichern zu können. Durch diese Formatanweisungen kann ein Benutzer Inhalte hervorheben, wie es beispielsweise in einem einfachen Programm zur Textverarbeitung möglich ist. Hierzu steht dem Benutzer eine Palette mit bekannten Optionen zur Formatierung, oberhalb der Texteingaben, zur Verfügung. Über diese kann er selektierte Textinhalte beispielsweise durch Fettschrift (siehe Hervorhebung "superenergy") oder Unterstreichungen (siehe Hervorhebung "from traditonal") akzentuieren. Diese For-

matierungen werden bei der Generierung des SIWOB-PDF ausgewertet und in das menschenlesbare Format überführt, sodass diese dort sichtbar werden.

In der aktuellen prototypischen Fassung werden in der Oberfläche einige ausgewählte Formatanweisungen berücksichtigt. Diese werden in den maschinenlesbaren Anteil überführt und bleiben dort erhalten, sodass diese ebenfalls in der Oberfläche angezeigt werden können. Die Überführung in ein menschenlesbares Dokument berücksichtigt bisher nur Basiselemente (fett, kursiv, unterstrichen und durchgestrichen). Dies begründet sich in der Umsetzungskomplexität, da jedes Formatelement bei der Generierung identifiziert und für eine PDF-Ausgabe passend übersetzt werden muss. Es wäre deshalb empfehlenswert zuerst die benötigten Formatelemente für die Erstellung einer Spezifikation zu ermitteln, die Benutzeroberfläche dahingehend anzupassen und anschließend diese Elemente vollständig in die Generierung aufzunehmen, sodass ein konsistenter Ablauf gewährleistet werden kann.

Neben dieser Erweiterung muss zusätzlich eine Integration von Bilddaten bedacht werden, die in der prototypischen Fassung ausgespart wurde, jedoch grundsätzlich implementiert werden kann. Hierbei könnten verschiedene Ansätze realisiert werden, wie beispielsweise die Integration von Bilddaten mit Hilfe des WYSIWYG-Editors. Eine weitere Möglichkeit könnte darin bestehen, Bildinformationen als Anhang zum jeweiligen Eingabeelement hochzuladen. In allen Varianten muss die Verknüpfung zu einem Eingabeelement berücksichtigt werden, damit eine eindeutige Zuordnung von Bild- und Textinformation sichergestellt werden kann. Die Vor- und Nachteile dieser und eventuell weiterer Möglichkeiten sollten näher evaluiert und in eine abschließenden Entwicklung berücksichtig werden.

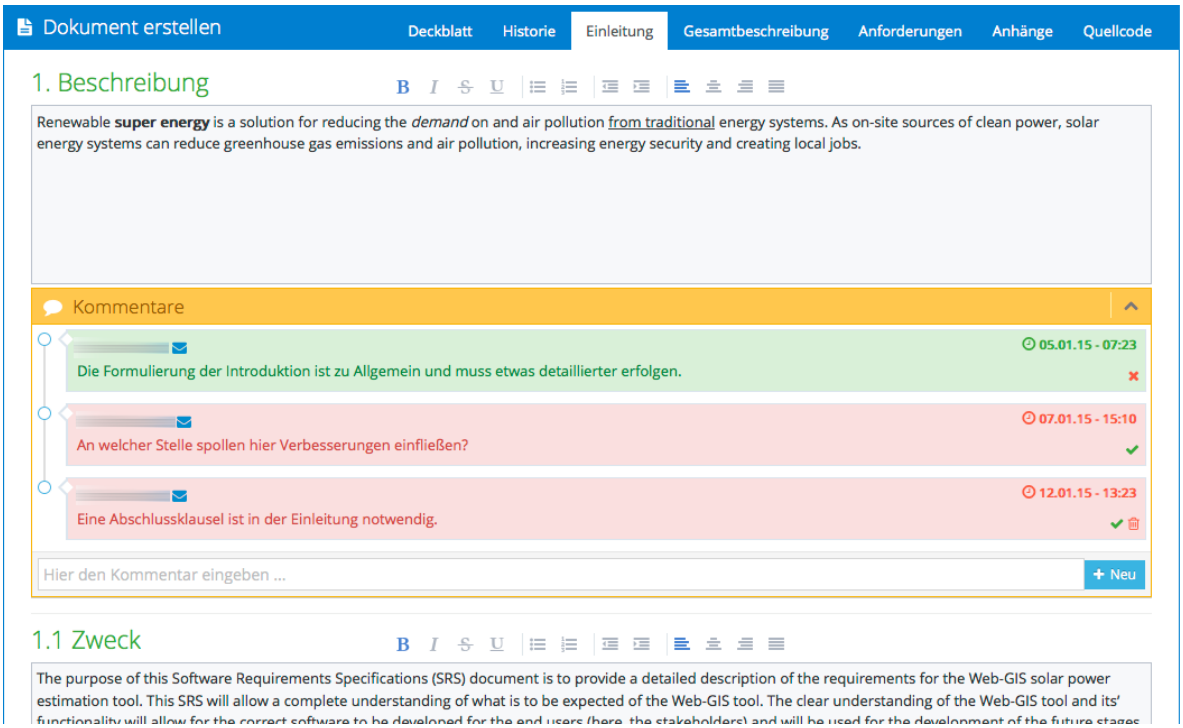



Abb. 64: Visualisierung von Meta-Daten im User Interface

Darüber hinaus enthält das maschinenlesbare Dokument, neben sichtbaren Elementen, zusätzliche Informationen. Ein Beispiel hierfür stellen Kommentare dar, die von Benutzern (Herstellern oder Prüfern) zu einzelnen Elementen einer Spezifikation formuliert werden können, wie in Abbildung 64 dargestellt wird. Alle Kommentarinformationen werden in den Meta-Informationen einer Spezifikation gespeichert und sind im menschenlesbaren Ausgabeformat nicht sichtbar.

In diesem Beispiel werden die Kommentare gezeigt, die dem Eingabefeld des ersten Elementes ("1. Beschreibung") innerhalb der Einleitung einer Spezifikation zugeordnet sind. Unterhalb dieses Feldes haben Benutzer die Möglichkeit Informationen in natürlicher Sprache als Kommentartext zu formulieren. Ein formulierter Kommentar kann beispielsweise eine Anmerkung, bzw. einen Verbesserungsvorschlag beinhalten oder eine formulierte Frage darstellen. Ein Kommentar wird über den Knopf "NEU" zum jeweiligen Element zugewiesen und gespeichert. Anschließend wird die Ansicht um den neu hinzugefügten Kommentar aktualisiert.

Die zugewiesenen Kommentare zu einem Element der Spezifikation werden in chronologischer Reihenfolge aufgelistet, sodass einfach ersichtlich wird, in welcher Reihenfolge ein Dialog entstanden ist. Diese Sortierung wird zusätzlich durch einen vertikalen Zeitstrahl auf der linken Seite unterstützt, anhand dessen eine visuelle Unterscheidung einzelner Kommentare erfolgen kann. Der Textinhalt eines Kommentars kann von den einzelnen Benutzern ausschließlich gelesen werden und ist nach dem Hinzufügen nicht veränderbar.

Jeder einzelne Kommentar wird zudem durch verschiedene personenbezogene Elemente erweitert, die auf den jeweiligen Benutzerprofilen basieren. Die erste Zeile beinhaltet auf der linken Seite den Verfasser des Kommentars, wodurch eine Zuordnung zu einer Person möglich wird. Eine Email-Funktion (Brief-Symbol) ermöglicht es den Verfasser direkt zu kontaktieren, sofern ein Kommentar umfangreicherer Informationen zur Klärung bedarf, die beispielsweise auf Basis von Unklarheiten in der Interpretation von Normdokumenten resultieren. Auf der rechten Seite der ersten Zeile werden Datum und Uhrzeit notiert und erlauben so eine zeitliche Zuordnung der Kommentarformulierung. In der zweiten Zeile erfolgt die Ausgabe des formulierten Kommentars, die auf der rechten Seite durch zusätzliche Funktionen ergänzt wird. Diese bieten den Benutzern (Prüfern und Herstellern), die Möglichkeit einen Kommentar in seinem Arbeitsstatus zu verändern. Neue Kommentare werden grundsätzlich mit roter Farbe hinterlegt und sollen dem Benutzer visualisieren welche Einträge noch nicht bearbeitet wurden. Wenn ein Benutzer einen Kommentar gelesen und die Anmerkungen berücksichtigt hat, kann er diesen als bearbeitet markieren, indem er auf den grünen Haken klickt, sodass die rote Hintergrundfarbe in grün umgefärbt wird. Durch dieses visuelle Hilfsmittel können Benutzer neue von bereits bearbeiteten Kommentaren unterscheiden. Zusätzlich bietet der letzte Kommentar die Option entfernt zu werden, wenn beispielsweise ein Eintrag unvollständig oder irrtümlich formuliert wurde, sodass eine Korrektur möglich wird.

Weiterhin muss für eine abschließende Entwicklung ermittelt werden, wie feingranular Kommentare in einer Spezifikation integriert werden können. In der aktuellen prototypischen Entwicklung ist die Kommentarfunktion für wenige Elemente realisiert, wie der Beschreibung in der Einleitung, womit die Funktionalität dieses Ansatzes belegt wird. Die feinste Granularität würde die Kommentarfunktion für jedes

einzelne Element einer Spezifikation realisieren, beispielsweise je Eintrag einer funktionalen Anforderung, wohingegen die gröbste Granularität einer einzelnen Kommentarfunktion je Spezifikationsdokument entsprechen würde. Zwischen der feinsten und gröbsten Granularität existieren jedoch eine Vielzahl von Abstufungen, wie beispielsweise eine Kommentarfunktion je Spezifikationskapitel, die dem Nutzen und dem Entwicklungsaufwand gegenübergestellt werden müsste, sodass eine optimale Funktionsabdeckung für Prüfer und Hersteller realisiert werden kann.

Der letzte Arbeitsschritt im Workflow für die Erstellung einer Spezifikation besteht aus der Selektion von Quellcode, welcher in den maschinenlesbaren Dokumentbereich integriert wird. In Abbildung 65 wird die Bedienoberfläche zur Auswahl von Quellcode gezeigt, die dem Benutzer visualisiert wird.

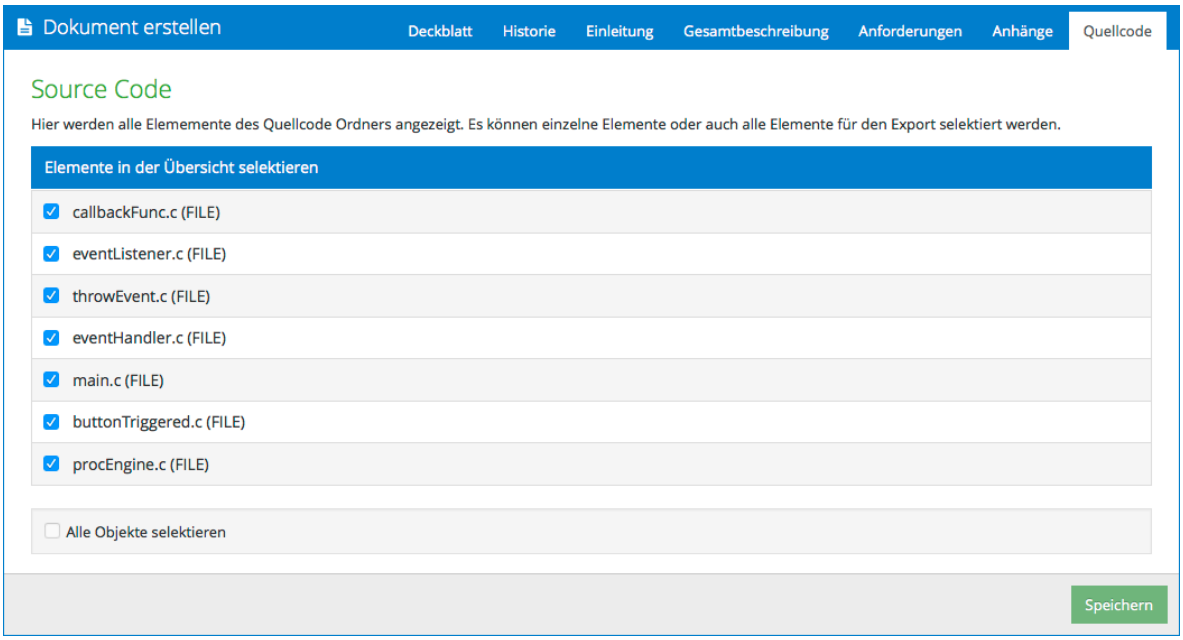



Abb. 65: Auswahl von Quellcode zur Integration in das Hybrid Format

Das User Interface generiert eine Liste von Quellcode-Elementen, die auf dem Inhalt des "Quellcode"-Ordners basieren, der im Containerformat spezifiziert wurde. Auf Basis dieser Ausgabe kann ein Benutzer die mit der Spezifikation assoziierten Quellcode-Dokumente selektieren, woraufhin die Software-Spezifikation im SIWOB Hybrid-Dokument integriert wird. Hierbei hat der Benutzer die Möglichkeit einzelne Quellcode-Elemente oder auch alle Elemente auf einmal für den Export auszuwählen.

In der aktuellen prototypischen Entwicklung ist eine Selektion von Ordnerstrukturen, die einzelne Quellcode-Elemente oder weitere Ordnerstrukturen enthalten können, nicht implementiert und muss in einer abschließenden Entwicklung berücksichtigt werden. Weiterhin wird die Versionierung von Dokumenten nicht berücksichtigt, sodass gegenwärtig ausschließlich die aktuellen Quellcode-Versionen in das Hybrid-Dokument integriert werden können. An dieser Stelle sollte hinterfragt werden, welche zusätzliche Funktionalität für eine Software-Prüfung vorteilhaft wäre. Beispielsweise könnte der versionierte Quellcode in allen Versionen in den maschinenlesbaren Teil des Hybrid-Dokumentes integriert werden oder auch nur eine bestimmte Version von einzelnen Dokumenten. Auch hier sollte eine Analyse des Nutzens in der ab-

schließenden Entwicklung vorangestellt werden, sodass eine optimale Funktionsabdeckung für Prüfer und Hersteller realisiert werden kann.

Der Workflow zur Erstellung einer Spezifikation wird im letzten Arbeitsschritt, der Auswahl von Quellcode, durch die Betätigung des Knopfes "Speichern" initialisiert (siehe Abbildung 65). Durch diese Aktion werden alle Eingaben und Informationen des Benutzers in einem Spezifikationsobjekt gespeichert. Zusätzlich werden selektierte Optionen in einer Liste gesammelt und alle Informationen an einen Service zur Generierung des Hybrid-Formates übermittelt.

Anschließend berechnet der Hauptprozess dieses Services, auf Basis der übermittelten Informationen aus dem User Interface, ein PDF-Dokument, das ausschließlich die menschenlesbaren Elemente enthält, wie in Abbildung 66 im rechten Bereich zu erkennen ist.

Für die Generierung dieses PDF-Dokumentes werden die übertragenen Objekt-Informationen Schritt für Schritt ausgewertet und mit Hilfe des PDF-Frameworks (iText) jede einzelne Seite des Dokumente erzeugt.

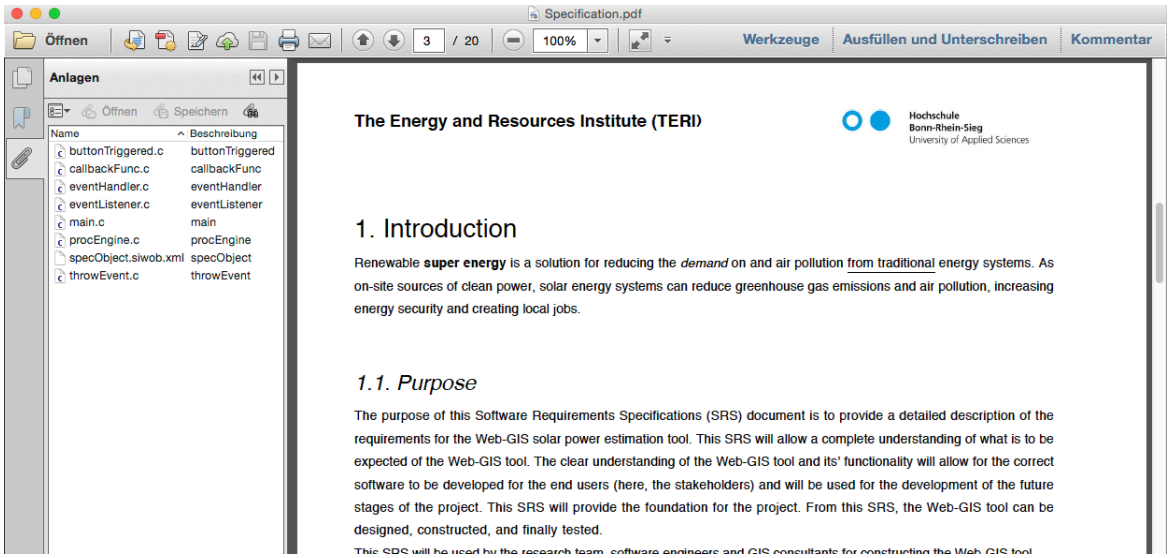



Abb. 66: Hybrid-PDF in menschenlesbarer Form mit integrierten Inhalten

Zuerst werden die Elemente definiert, die auf jeder einzelnen Seite gleiche Inhalte aufweisen, wie beispielsweise die Kopfzeile und die Fußzeile. In der Kopfzeile sind Informationen über den Titel des Dokumentes und das Logo des Unternehmens enthalten, wohingegen in der Fußzeile Informationen zur Dokumentversion, dem Autor, dem Erstellungsdatum, der Seitenzahl und weitere Informationen aufgeführt werden.

Anschließend werden die einzelnen Seiten erzeugt, beginnend mit dem Deckblatt. Abbildung 66 zeigt einen Auszug der Seite 3 des generierten PDF-Dokumentes, die auf Basis der Eingaben aus Abbildung 63 entstanden sind. Diese stellt das 1. Kapitel der Spezifikation dar und belegt, dass die Texteingaben mit dem WYSIWYG-Editor, insbesondere der Anweisungen zur Textformatierung, interpretiert und in das Dokument äquivalent übersetzt wurden. Weiterhin ist zu erkennen, dass die einzelnen Kommentare, die an

das Kapitel 1 gebunden wurden, in der Berechnung isoliert wurden und somit nicht im menschenlesbaren PDF-Dokument ausgegeben werden.

Auf diese Weise erfolgt für jede einzelne Seite des PDF-Dokumentes die Berechnung der menschenlesbaren Inhalte, bis alle Kapitel der Spezifikation verarbeitet sind. Den Abschluss der Berechnung bildet die Generierung eines Inhaltsverzeichnisses (Table of Content), anhand dessen eine Übersicht der Kapitelstrukturen und Seitenzuordnung ersichtlich wird.

Der letzte Verarbeitungsschritt besteht in der Integration der maschinenlesbaren Inhalte in das PDF-Dokument, wie in Abbildung 66 auf der linken Seite zu erkennen ist. Hier werden alle Dokumente aufgelistet, die sich im versteckten Bereich des PDF-Dokumentes als Anhang befinden. Neben den eingebundenen Quellcode-Dateien, die kongruent mit der Selektion aus Abbildung 65 sind, ist auch das generierte XML-Dokument ("specObject.siwob.xml") enthalten. Diese Dokumente stellen die maschinenlesbaren Inhalte des Hybrid-PDFs dar und erlauben eine erweiterte Verarbeitung der Spezifikationsinhalte.

Das fertig generierte Hybrid-PDF wird in den Container des jeweiligen Projektes gespeichert und über das User Interface als neues Dokument angezeigt. Das Hybrid-PDF kann vom Hersteller über das Import/Export Werkzeug (siehe Kapitel 5.5.4) in einen passenden Container exportiert oder über das Dateisystem des Betriebssystems lokalisiert werden, sofern er das Dokument an einen Prüfer senden möchte.

Eine weitere Möglichkeit besteht darin das Hybrid-PDF über das User-Interface zu öffnen. In diesem Fall wird der maschinenlesbare Anteil des Dokumentes gelesen, ausgewertet und an das Werkzeug zur Spezifikationsgenerierung übergeben, sodass ein Hersteller die Inhalte der Spezifikation verändern kann. Dieser Schritt wird immer dann notwendig, wenn ein Prüfer Änderungen verlangt, beispielsweise aufgrund inhaltlicher Schwächen der Spezifikation. Für eine Änderung kann ein Hersteller die Inhalte über das User Interface modifizieren und das Dokument abschließend speichern, wie in den vorherigen Prozessschritten beschrieben wurde, sodass eine neues Hybrid-PDF generiert wird.

Auf der Seite eines Prüfers besteht zudem die Anforderung Inhalte der Spezifikation nicht ändern zu können. Ein Prüfer kann dennoch ein Hybrid-PDF, welches er vom Hersteller erhalten hat, über das User Interface öffnen, mit der Einschränkung, dass alle Inhalte einen Schreibschutz aufweisen. Die Inhalte werden ebenfalls aus dem maschinenlesbaren Bereich des Hybrid-PDFs entnommen und entsprechend visualisiert.

Darüber hinaus erhält ein Prüfer die Möglichkeit Kommentare zu den jeweiligen Elementen zu formulieren, da diese Änderungen nicht den menschenlesbaren Anteil verändern, sondern ausschließlich die XML-Datei im maschinenlesbaren Bereich des Hybrid-PDFs. Dadurch kann er die Inhalte der Spezifikation zielgenau kommentieren und das Dokument an den Hersteller zurücksenden. Der Hersteller kann das Dokument mit schreibenden Rechten über das User Interface öffnen, die Kommentare auswerten und daraufhin das Dokument entsprechend korrigieren.

### 5.5.8 Traceability

Innerhalb eines Prüfprozesses werden unter anderem die Software-Spezifikation und der Quellcode vom Hersteller zur Verfügung gestellt. Da Spezifikation und Quellcode in unterschiedlichen Dokumenten zu finden sind, muss zuerst eine Verknüpfung von Anforderungen zu korrespondierenden Implementierungen erfolgen (siehe Abbildung 67), damit eine Analyse erfolgen kann. Anschließend werden die einzelnen Anforderungen der Software-Spezifikation hinsichtlich ihrer Konformität zur Spezifikation untersucht und Abweichungen sowie Fehler in Form von Anmerkungen manuell notiert.

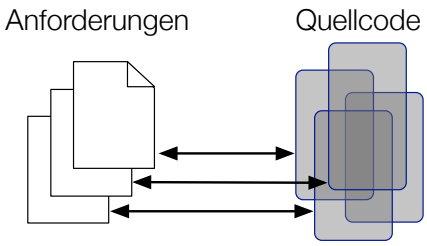



Abb. 67: Zuordnung von Spezifikation zu Quellcode

Um den Prüfer bei der eigentlichen Prüfung der Quelltexte und der Software-Spezifikation zu unterstützen, soll ein Teil dieser Schritte durch den Einsatz geeigneter Tools automatisiert werden. So kann eine Software einen Zeitgewinn ermöglichen, die eine Zuordnung von Elementen der Spezifikation zu Quellcode-Modulen ermöglicht. Die angestrebten automatisierten Vorgänge haben weitere Vorteile, wie die Möglichkeit, Anmerkungen digital zu speichern und den beteiligten Personen zur Verfügung zu stellen.

Übergeordnet wurde ein System entwickelt, welches Prüfer und Hersteller bei den beschriebenen Prozessen unterstützt. Dies umfasst unter anderem die Teilbereiche der Anforderungsanalyse, Designfragen und Software-Entwicklung. Aufbauend wurde ein Prototyp realisiert, welcher die Machbarkeit der erkannten Anforderungen belegt. Ein Vorteil eines derartigen Systems besteht darin, dass ein Hersteller bei der Verknüpfung von Spezifikation und Quellcode unterstützt wird und ein Ausgabeformat erzeugt, welches ein Prüfer nutzten kann.

Die Richtlinie IEC 61508-3 bezieht sich in diesem Zusammenhang speziell auf die Software von sicherheitskritischen Systemen und schreibt vor, welche Anforderungen diese zu erfüllen hat. Dabei betrifft ein großer Teil der Richtlinie die organisatorische Sicherheit, welche Maßnahmen durchgeführt werden müssen, um sichere Software zu entwickeln und welche Verfahren einzuhalten sind. Dazu zählt unter anderem die Verbindung von Software-Spezifikation, Implementierung und den dazugehörigen Tests der Software und Systeme, die sogenannte Traceability.

Bei der anschließenden Verifikation eines Baumusters wird die Software gegen eine Software-Spezifikation geprüft und die Fragestellung geklärt, ob das Programm die Anforderungen erfüllt.[232] Im Regelfall

232. vgl. (Flanagan & Qadeer, 2002)

wird die Verifikation manuell durch einen qualifizierten Software-Prüfer durchgeführt, dem die Arbeit durch eine passende Software-Lösungen vereinfacht werden soll.

Um diese Prozesse möglichst effizient zu gestalten und den Ansprüchen der "Traceability" gerecht zu werden, werden Möglichkeiten evaluiert, wie bestimmte Teilbereiche bereits gelöst wurden. Es wird erörtert, ob eine automatisierte Erkennung und Zuordnung von Spezifikationspunkten, beispielsweise anhand der Beschreibung möglich ist und anhand dieser die passende Methode im Quellcode erkannt werden kann.

Im Bereich des Text-Mining werden Berechnungsmodelle herangezogen, um unterschiedliche Texte zu analysieren und Ähnlichkeiten zwischen diesen zu identifizieren. Durch diese Technik wird es möglich ähnliche Dokumente oder Textpassagen in einer großen Datensammlung zu entdecken. Die Methode aus dem Text-Mining ist für das vorliegende Problem allerdings nur schwierig anzuwenden. So sind die zu untersuchenden Texte vergleichsweise kurz und unter Umständen sehr technisch. Auf der anderen Seite ist Quellcode nicht geeignet, um mit Methoden untersucht zu werden, die auf natürlich sprachlichen Texten aufbauen, sodass eine Berechnung der Ähnlichkeit von Dokumenten und Inhalten mittels Text-Mining ausgeschlossen ist. Eine weitere Möglichkeit wäre theoretisch, die Entwickler von Quellcode anzuleiten, Methoden speziell zu bezeichnen, beispielsweise mit Tags, die in der Spezifikation vorgegeben sind. Dann könnten die einzelnen Anforderungen in der Spezifikation mit den einzelnen Methoden im Quellcode mit Hilfe der Tags verknüpft werden. Dies kann aber ebenfalls problematisch werden. So würden bestehende Konventionen, die Entwicklerteams einhalten, zwanghaft aufgebrochen werden, um eine organisatorische Aufgabe zu erfüllen.[233] Es muss somit eine Lösung gefunden werden, um möglichst komfortabel die Verknüpfung zwischen Spezifikation und Quellcode manuell durchführen zu können.

### Konzept- und Anforderungsentwicklung

Der Aufbau für dieses Konzept der Traceability basiert auf den zu Beginn dieses Kapitels (5.5.8) formulierten Zielen, Zeiteinsparungen zu erzielen und gleichzeitig ein Software-System zu entwickeln, welches den aktuellen Anforderungen an Software wie Benutzbarkeit, Performanz und Wartbarkeit genügt. Im aktuellen Kapitel werden die Anforderungen an dieses System analysiert, um daraus das Konzept für den Aufbau zu entwickeln.

Das Konzept führt Anforderungen auf, die durch Rahmenbedingungen der bestehenden SIWOB Entwicklung ermittelt wurden. In Iterationszyklen wurden anschließend die grundlegenden Vorstellungen immer weiter verfeinert und herausgearbeitet.

Da das System, sowohl auf der Herstellerseite, wie auch auf der Prüferseite, von mehreren Personen gleichzeitig benutzt werden soll, bietet sich hier der Aufbau einer verteilten Anwendung an. Durch die Trennung der Benutzeroberfläche von der Programmlogik und der Datenhaltung, entsteht die Möglichkeit, mehreren Benutzern parallel Zugriff zu ermöglichen.

233. vgl. (Martin, 2003, S. 149ff.)

Eine verteilte Anwendung setzt besteht aus verschiedenen Systemen, bestehend aus heterogener Hardware und Software, die miteinander verknüpft ein Produkt oder Service anbieten. Das größte verteilte System ist das Internet, welches 2012 aus mindestens 450 Millionen Geräten bestand, die durch gemeinsam genutzte Protokolle miteinander kommunizieren.[234] Um eine verteilte Anwendung zu ermöglichen, wird daher ein verteiltes System benötigt.[235] Die Komponenten einer Anwendung können sich auf physikalisch getrennten Systemen befinden und über ein oder verschiedene Protokolle kommunizieren. Gründe für verteilte Anwendungen sind unter anderem

- durch Redundanz der Komponenten die Ausfallsicherheit der gesamten Anwendung zu erhöhen
- die Last auf mehrere Systeme zu verteilen
- Ressourcen effizienter zu nutzen

Ein Ziel des Services ist die Verarbeitung und Erkennung von Quellcode. Um den Quellcode zu verarbeiten und sinnvoll zu strukturieren, muss dieser zuvor analysiert werden. Ein naheliegender Weg ist es, diesen in seine einzelnen Bestandteile wie beispielsweise Variablen und Methoden zu unterteilen. Dazu wird ein Quellcode-Parser benötigt. Im weiteren Schritt ist zu entscheiden, wie die erkannte Struktur des Programms gespeichert werden soll.

Als eine geeignete Möglichkeit wird die Speicherung in einer XML-Datei ausgewählt. Extensible Markup Language (XML) eignet sich sehr gut, um strukturierte Informationen vorzuhalten. Im vorliegenden Fall sind verschiedene Verfahrensschritte zu bedenken, wenn eine Quellcode-Datei ausgewertet werden soll. Es muss eine Prüfung darauf erfolgen, ob die Quellcode-Datei:

1. Ein lesbares Dokument darstellt. (Das Dokument muss in einem Zeichensatz gespeichert sein und vom System unterstützt werden, wie ASCII oder Unicode und kein Binärformat.)
2. Ein gültiges Programm ist.

Punkt 1 kann umgesetzt werden, indem beispielsweise die unterstützten Kodierungen bei der Verarbeitung der Programmdateien vorausgesetzt werden und auf ein bestimmtes Muster eines Programms geprüft wird. Da der Fokus auf Software liegt, die in der Programmiersprache C geschrieben ist, könnte eines der Muster die Signatur der `"main"`-Methode sein oder eine `"include"`-Anweisung.

Punkt 2 kann umgesetzt werden, indem vorab geprüft wird, ob ein Programm ausführbar ist. Insbesondere eine Ausgabe von gültigen Informationen weist auf ein funktionsfähiges Programm hin. In weiteren Prüfschritten muss validiert werden, wie sich das Programm beispielsweise bei fehlerhaften Eingaben verhält. Die Prüfung auf ein fehlerhaftes Verhalten von Programmen ist jedoch nicht Bestandteil dieser Entwicklung.

Im weiteren Verlauf wird das angestrebte Software-System evaluiert, welches der Hersteller verwendet, um eine Spezifikation zu erstellen und Quellcode hochzuladen. Weiterhin ist es erforderlich die Anforde-

234. vgl. (Stöcker, 2013)

235. vgl. (Hammerschall, 2005, S. 16)

rungen der erstellten Spezifikation mit den einzelnen Methoden des Quellcodes zu verknüpfen. Dabei wird eine lokale Benutzung über einen Browser (Firefox, Safari, etc.) vorausgesetzt. Aus dem Konzept und den Analysen ergeben sich die nachfolgenden Anforderungen zwischen Hersteller und Software-System.

| 1. Projekt anlegen |
|---|
| **Beschreibung:** Der Benutzer legt im ersten Schritt ein Projekt an, welches eine eindeutige Bezeichnung erhält. Diese Identifikation wird im weiteren benutzt, um die einzelnen Teile der Spezifikation zuzuordnen.<br>Im vorliegenden Konzept wird dieser Schritt prototypisch in den Anwendungsfall "Spezifikation bearbeiten" integriert, indem die entsprechenden Eingabefelder dem Benutzer angezeigt werden und auszufüllen sind. |
| **Vorbedingung:** Kein aktives Projekt ausgewählt. |
| **Nachbedingung** Erstelltes Projekt als aktives Projekt ausgewählt. |

| 2. Projekt löschen |
|---|
| **Beschreibung:** In einer Produktionsumgebung ist es sinnvoll, bereits erstellte Projekte nach Abschluss zu archivieren und aus der Liste "aktiver Projekte" zu entfernen. Da im vorliegenden Konzept der Fokus nicht auf der Erstellung einer Projektverwaltungs-Software liegt, wird hier nicht weiter auf diesen Punkt eingegangen. |
| **Vorbedingung:** Projekt ausgewählt. |
| **Nachbedingung:** Projekt aus der Liste der aktiven Projekte entfernt. |

| 3. Spezifikation bearbeiten |
|---|
| **Beschreibung**: Vom Benutzer wird eine Spezifikation bearbeitet. Dies umfasst die Schritte:<br>• Erstellen einer neuen Spezifikation<br>• Bearbeiten der Punkte einer existierenden Spezifikation<br>Diese Spezifikation ist das Pflichtenheft des Entwicklers und beschreibt die formalen Anforderungen des einzureichenden Programms. Dabei soll der Benutzer eine Maske erhalten, welche dem vereinbarten Aufbau entspricht. In diese Maske wird jeweils ein Punkt der Spezifikation eingegeben. Durch den Klick auf einen Knopf im Browser-Fenster wird der Punkt gespeichert. Weiter soll der Benutzer die Möglichkeit haben, bereits gespeicherte Punkte auszuwählen, zu bearbeiten und zu löschen. |
| **Vorbedingung:** Es existiert ein Projekt zu dem die Spezifikation zugehörig ist. |
| **Nachbedingung**: Spezifikation inklusive der erstellten Punkte existiert. |

| 4. Einstellen von Quellcode |
|---|
| **Beschreibung:** Vom Benutzer wird der bestehende Quellcode dem System bekannt gemacht. Dieser Vorgang ist nötig, um den Quellcode mit der Spezifikation verknüpfen zu können. Hierzu wählt der Benutzer den Knopf "Datei hochladen" und wählt die Datei mit dem Quellcode aus. |
| **Vorbedingung**: Es wird der Quellcode in Form einer Datei benötigt. |
| **Nachbedingung**: Die Anforderungen innerhalb der Spezifikation, inklusive der Quellcode-Datei, müssen inhaltlich in Beziehung zueinander stehen. |

| 5. Verknüpfung von Anforderungen mit Quellcode |
|---|
| **Beschreibung:** Der Benutzer öffnet die Maske zur Verknüpfung von Spezifikation und Quellcode. In dieser Oberfläche öffnet der Benutzer auf der linken Seite die Spezifikation über den Knopf "Spezifikation öffnen", sodass die Anforderungen einzeln selektiert werden können. Anschließend öffnet der Benutzer über den Knopf "Quellcode öffnen" auf der rechten Seite die einzelnen Quellcode-Dateien. Der Benutzer selektiert eine Anforderungen auf der linken Seite und eine Methode einer Quellcode-Datei auf der rechten Seite und verbindet diese über den Knopf "Verbinden", wodurch in der Logik eine Verbindung erstellt und gespeichert wird. In dieser Methodik verfährt der Benutzer weiter, bis alle Elemente miteinander verknüpft sind. |
| **Vorbedingung**: Es existiert ein Projekt mit Spezifikation und Quellcode |
| **Nachbedingung**: Anforderungen und Quellcode sind miteinander verknüpft. Alle Informationen, inkl. der Verknüpfungsinformationen sind gespeichert. |

Tabelle 16: Anforderungen für die prototypische Entwicklung der Traceability

Alle formulierten Anforderungen wurden mit den zuvor beschriebenen Technologien (Web Services, etc.) prototypisch realisiert. Dabei soll die Programm-Oberfläche für den Benutzer möglichst intuitiv und selbsterklärend sein.

### 5.5.9 Word Spezifikationsvorlage

In der Software-Entwicklung sind Dokumentationen ein wichtiger Bestandteil für eine erfolgreiche und nachvollziehbare Umsetzung von Lösungen. Werden Dokumentationen nicht oder unzureichend angefertigt, können Missverständnisse oder Unklarheiten zu fehlerhaften Implementierungen führen, die Geld und Zeit kosten. Eine nutzbringende Dokumentation im Entwicklungsprozess stellt die Software-Spezifikationen dar, in der Anforderungen an die zu erstellende Software festgehalten werden.

Die bestmögliche Unterstützung bei der Erstellung und Prüfung einer Software-Spezifikation wird erreicht, indem Vorgaben an eine Software-Spezifikation und ihre Struktur untersucht und in einem prototypischen Vorlagendokument umgesetzt werden.

Hierbei bestand die Herausforderung darin eine Methodik zu entwickeln, anhand der die Struktur von Software-Spezifikationen erfasst und auf Basis dieser eine Formularvorlage erstellt werden kann. Dadurch soll eine Basis geschaffen werden, um Daten aus Software-Spezifikationen für Konsistenz- und Inhaltsprüfungen zu erfassen. Die Konsistenz der Software-Spezifikation ist dann gegeben, wenn die ermittelten Strukturvorgaben eingehalten sowie die verlangten Anforderungen definiert wurden. Im Fokus steht deshalb die Konsistenzsicherung, die mit Hilfe automatisierter Prüfungen unterstützt werden kann. Weiterhin sollen die Inhalte der Software-Spezifikation mithilfe geeigneter Auszeichnungssprachen, wie beispielsweise XML, darstellbar sein.

Zuerst muss hierfür eine Analyse über die Vorgaben und Normen erfolgen, anhand derer eine Software-Spezifikationen erstellt wird. Auf Basis dieser Informationen gilt es Rückschlüsse auf eine Struktur der Software-Spezifikation zu ziehen, sodass diese durch eine Auszeichnungssprache abgebildet werden

kann. Die geplante Struktur soll zusätzlich als Grundlage für die Validierung der Nutzereingaben des Vorlagendokumentes dienen.

Weiter galt es zu evaluieren, welche Werkzeuge bei der Erstellung von Software-Spezifikationen zum Einsatz kommen. Mit dem Ergebnis wurden die in der Analyse festgestellten Strukturmerkmale in einem Vorlagendokument umgesetzt. Im weiteren Verlauf gilt es dieses Vorlagendokument mit Formularfeldern auszugestalten. Das Dokument muss so realisiert werden, dass Nutzer neue Inhalte generieren können, diese aber immer innerhalb der definierten Struktur bleiben. Die Struktur und Nutzereingaben sind anschließend für die Validierung und Verifikation auszulesen.

## Anforderungen und Anforderungsmanagement

Innerhalb eines Gesamtentwicklungsprozesses einer Software werden mehrere Phasen durchlaufen, die strukturierte Abläufe zur planvollen Erstellung der Software definieren. Beispielsweise werden im V-Modell die Phasen in Anforderungsanalyse, Systemanalyse, Entwurfserstellung, Implementierung und Verifikation, Integrationstest, Systemtest und Abnahme unterteilt.[236] Der Detailgrad der einzelnen Phasen, wie auch die Anzahl der Phasen, können hierbei je nach verwendetem Modell variieren. Die Anforderungsanalyse stellt die erste Aufgabe im Gesamtentwicklungsprozess dar, aus der im Ergebnis die Software-Spezifikation resultiert. Die allgemeine Aufgabe des Anforderungsanalyse kann charakterisiert werden, als "das Finden eines geeigneten, theoretisch fundierten Beschreibungsmittels, das (in einem Gesamtentwicklungsprozess) methodisch sauber eingebettet ist und durch praktisch brauchbare Werkzeuge unterstützt wird."[237] Um bei der Erstellung eines Produktes zu wissen, aus welchen Teilen und mit welchen Funktionen die Software erstellt werden soll, werden Anforderungen erhoben. Hierzu werden alle Anforderungen zu einem Produkt erhoben und in einer Software-Spezifikation dokumentiert. Eine Anforderung stellt hierbei eine Funktion oder ein Qualitätsmerkmal dar und existiert, weil sie zur Erfüllung einer Funktion oder zur Sicherstellung der Qualität in einem fertiggestellten Produkt vorhanden sein muss.[238]

Eine Spezifikation stellt im Grunde die Beschreibung eines Produktes, eines Systems oder einer Dienstleistung dar. Ziel der Spezifikation ist es, Anforderungen zu definieren und zu quantifizieren.[239] Die Spezifikation dient zusätzlich bei der Übergabe des fertigen Produktes an den Auftraggeber als Prüfungsgrundlage für die Erreichung der definierten Ziele. Diese Dokumentation soll bei allen Projektbeteiligten und Interessengruppen ein gleiches Verständnis der Anforderungen schaffen.[240] In der Praxis kommen deswegen verschiedenste Methoden und Werkzeuge zum Einsatz, um Anforderungen besser definieren zu können.

Grundsätzlich kann eine Software-Spezifikationen in formale und natürlichsprachliche Varianten unterschieden werden. Formale Software-Spezifikationen enthalten Anforderungen, die wenig bis keine Aus-

236. vgl. (Ebert, 2012, S. 303)
237. vgl. (Ebert, 2012, S. 102)
238. vgl. (Ebert, 2012, S. 23f.)
239. vgl. (Ebert, 2012, S. 88)
240. vgl. (Robertson & Robertson, 2006, S. 26)

drücke in natürlicher Sprache enthalten und mithilfe von formalen Spezifikationssprachen formuliert sind. Diese Art der Spezifikationen findet Einsatz in Gebieten die eine geringe Fehlertoleranz fordern, wie beispielsweise sicherheitskritische Einsatzbereiche. Zusätzlich dient die formale Software-Spezifikation dazu, ausführbaren Programmcode aus den Formulierungen direkt zu erzeugen.[241] In natürlichsprachlichen Software-Spezifikationen hingegen werden die Anforderungen detailliert in natürlicher Sprache beschrieben und mit Diagrammen und technischen Definitionen erweitert.[242] Aus derartigen Formulierungen lassen sich deshalb keine ausführbaren Programme generieren, da die Beschreibungen meist zu wenig Formalismen aufweisen, aus denen Quellcode generiert werden könnte. Im weiteren Verlauf wird der Begriff der Software-Spezifikationen im Kontext der natürlichsprachlichen Variante betrachtet.

Wird auf eine Spezifikation verzichtet oder diese nur unvollständig angefertigt, kann dies im Entwicklungsprozess die anknüpfenden Phasen behindern oder verzögern. So lassen sich beispielsweise Testfälle ohne klare Testziele nur schwer realisieren. Entwicklern kann es an detaillierten Vorgaben fehlen oder im späteren Entwicklungsprozess fällt auf, dass einzelne Anforderungen ungeklärt sind.[243] Solche Fehlannahmen oder fehlerhafte Lösungsansätze, die während der Erstellung einer Software-Spezifikationen auftreten, erfordern im Nachhinein weiteren Personen- und Zeiteinsatz.[244] Dadurch ist es von hohem Wert Inkonsistenzen in der Software-Spezifikationen bereits in der Phase der Erstellung zu erfassen und zu beseitigen.

Damit eine Software-Spezifikation für die weiteren Phasen tauglich ist, muss diese korrekt, eindeutig, vollständig, konsistent, priorisierbar, testbar, veränderbar und nachvollziehbar sein.[245] Eine Software-Spezifikation ist korrekt, wenn jede Anforderung, die zur Erfüllung der Funktion der Software nötig ist, beschrieben wurde. Die Eindeutigkeit einer Spezifikation ist dann gegeben, wenn die Beschreibung jeder Anforderung nur eine Interpretation zulässt. Ein Mindestgrad an Eindeutigkeit ist sichergestellt, wenn jede Anforderung eindeutig bezeichnet wurde. Eine Spezifikation muss bezüglich der zu spezifizierenden Anforderungen und des Verhaltens der Software auf Eingangsinformationen vollständig sein. Außerdem müssen alle notwendigen Informationen und Definitionen zur Struktur und Lesbarkeit der Spezifikation vorhanden sein. Die Konsistenz der Spezifikation ist dann gegeben, wenn keine Konflikte von Anforderungsdefinitionen innerhalb der zu erstellenden Spezifikation sowie zu referenzierten Spezifikationen entstehen. Jede Anforderung muss unterscheidbar sein, beispielsweise nötige und optionale Anforderungen. Damit die Testbarkeit sichergestellt ist, müssen die Anforderungen verifizierbar sein, indem diese durch quantifizierbare Angaben spezifiziert sind. Weiterhin muss das Spezifikationsdokument leicht veränderbar sein, dabei aber nicht die Struktur und das Format verletzten. Die Herkunft und weitere Ver-

241. vgl. (Ebert, 2012, S. 103)
242. vgl. (Ebert, 2012, S. 101)
243. vgl. (Ludewig & Lichter, 2010, S. 355f)
244. vgl. (Pohl, 2008, S. 12)
245. vgl. (Rupp, 2009, S. 24)

wendung von beschriebenen Anforderungen müssen abschließend nachvollziehbar dargestellt werden.[246]

## Analyse und Abbildung einer Spezifikationsstruktur

In diesem Kapitel wird die Analyse der Norm IEEE/ANSI 830-1998 "IEEE Recommended Practice for Software Requirements Specifications" vorgenommen. Dazu wird die Methodik vorgestellt, mit der die Struktur untersucht und die identifizieren Struktureigenschaften für die folgenden Prozesse dokumentiert wurde. Mithilfe der vorgestellten Methodik werden die Empfehlungen der Norm einzeln auf ihre Struktureigenschaften untersucht. Die analysierten Eigenschaften werden in einem XML-Dokument festgehalten und die Strukturmerkmale in einem darauf aufbauenden Schema-Dokument spezifiziert, sodass dieses für eine Prüfung der Inhalte verwendet werden kann.

Die IEEE/ANSI 830-1998 beschreibt Vorgaben und Empfehlung, um Anforderungen in einer Spezifikation zu strukturieren und abzubilden. Diese Vorgaben beinhalten eine Kapitelstruktur sowie detaillierte Vorschläge für die Strukturierung der einzelnen Unterkapitel und deren Inhalte. Für die Erstellung der prototypischen Vorlage werden diese Vorgaben auf Strukturmerkmale analysiert und diese festgehalten. Ein Strukturmerkmal beschreibt, wie spezifische Inhalte des Elements angeordnet werden sollen, wobei diese Inhalte wiederum aus weiteren Elementen bestehen können.

Für die Analyse der Struktur einer Spezifikation ist es nicht notwendig die natürlich sprachlichen Inhalte auszuwerten. Es genügt hierzu nur die äußere Struktur auf ihre Merkmale zu untersuchen. Für die Beschreibung der zu analysierenden Elemente werden folgende Strukturmerkmale definiert:

- **"Inhalt"**: Es wird nur beschreibender Inhalt dargestellt, der nicht weiter strukturiert wird (keine weiteren Elemente).
- **"Gruppierung"**: Es wird kein Inhalt dargestellt, sondern eine Anzahl von Elementen gruppiert.
- **"Wiederholung"**: Ein Strukturmerkmal, das eine Wiederholung einer definierten Struktur beschreibt und hierarchisch unterhalb des betrachteten Strukturmerkmals Anwendung findet.

Gemäß der Definition dürfen Inhalte keinen Gruppierungen zugewiesen werden. Gruppierung und Wiederholung sowie Inhalt und Wiederholung, sind Strukturmerkmale, die paarweise verbunden werden können. Kapitelüberschriften sind immer als ein Element zu betrachten und sollten nicht als Inhalt klassifiziert werden. Für den Fall, dass auf ein Element ein weiteres folgt, wird dies dem Strukturmerkmal Gruppierung zugewiesen.

Für die weitere Vorbereitung der prototypischen Dokumentvorlage, ist die analysierte Struktur der Norm und die daraus identifizierten Strukturmerkmale in einer Tabelle (siehe Anhang 7.3) dokumentiert. In der ersten Spalte wird hierzu das Element benannt und in der zweiten Spalten die zugeordneten Struktur-

246. vgl. (IEEE Computer Society, 1998, S. 4f)

merkmale aufgeführt. Die Strukturmerkmale werden mit den Zeichen "i" für Inhalt, "g" für Gruppierung und "w" für Wiederholung gekürzt dargestellt und bei mehrfacher Nennung mit Komma getrennt.

Nach IEEE/ANSI 830-1998 besteht die Kapitelstruktur für eine Spezifikation aus mindestens vier übergreifenden Kapiteln, der "Einleitung", "Allgemeine Beschreibung der Anforderungen", "Detaillierte Beschreibung der Anforderungen" und "Appendix". Nachfolgend werden diese Kapitel und die darin enthalten Strukturen detailliert analysiert und aus den einzelnen Elementen geeignete Merkmale zugeordnet. Zu diesem Zweck erfolgt eine Abbildung der Kapitelstruktur in Form einer abgestuften Nummerierung, damit später eine leichtere Zuordnung zur prototypischen Dokumentvorlage möglich wird.

### Kapitel 1. Einleitung

Die Einleitung gliedert Inhalte, die einen Gesamtüberblick über die Software-Spezifikation und das Produkt geben. Innerhalb der Einleitung sind weitere Unterkapitel enthalten, die die Inhalte innerhalb dieses Kapitel weiter aufgliedern und strukturieren. Da der Einleitung, weitere Unterkapitel untergeordnet sind, weist sie eine gruppierende Eigenschaft auf.[247]

#### Kapitel 1.1 Zweck des Dokuments

Innerhalb des Kapitels Zweck des Dokuments wird beschrieben, welche Zielsetzung und welchen Zweck das Dokument hat. Im Weiterem ist zu erläutern, an welche Adressaten sich das Dokument richtet. Diese Elemente sind dem Strukturmerkmal Inhalt zuzuordnen.

#### Kapitel 1.2 Abgrenzung der Software

Neben dem Zweck des Dokuments muss ein Überblick über das Produkt gegeben werden. Dazu wird innerhalb des Kapitels Abgrenzung der Software der Name des Produkts beschrieben. Um das Produkt weiter abzugrenzen sind Kernfunktionalitäten zu schildern. Für ein Grundverständnis darüber, welchen Mehrwert das Produkt erzeugt, werden die Ziele und der Nutzen in dem Einsatzgebiet der Software erläutert. Dieses Kapitel gruppiert die verschiedenen Elemente Name, Kernfunktionalität, Einsatzgebiet, Ziele und Nutzen. Da diese Elemente nicht wiederholt werden, ist diesem Eintrag eine gruppierende Struktureigenschaft zuzuweisen. Die einzelnen Elemente dieser Gruppe werden in natürlicher Sprache formuliert und weisen keine weiteren Gliederungen auf, sodass diese jeweils das Strukturmerkmal Inhalt erhalten.[248]

#### Kapitel 1.3 Begriffsdefinitionen und Kapitel 1.4 Abkürzungen

Teil der Einleitung sind Begriffsdefinitionen sowie Abkürzungen und die dazu gehörenden Definitionen, die in der Regel in einer tabellarischen Auflistung zusammengefasst werden. Diese Kapitel gruppieren jeweils einzelne Einträge, die aus der Definition oder der Abkürzung bestehen und einfach Inhalte darstellen. Dadurch dass die einzelnen Einträge, innerhalb des jeweiligen Kapitels

247. vgl. (IEEE·Computer·Society,·1998,·S.·11)

248. vgl. (IEEE·Computer·Society,·1998,·S.·11)

Begriffsdefinition und Abkürzungen, in derselben Struktur wiederholt werden, kann den Einträgen eine gruppierende und wiederholende Struktureigenschaft zugeordnet werden.

### Kapitel 1.5 Referenzen

Wenn Referenzen zu anderen Dokumenten vorhanden sind, sind diese in dem Kapitel Referenzen aufzuzeigen. Auch hier wird in der Regel eine tabellarische Darstellung angewendet. Diese Tabelle enthält die Elemente Titel, (wenn zutreffend) Report Nummer, Datum, die veröffentlichende Organisation und die Quelle. Dieses Kapitel ist ebenso wie die Definitionen zu behandeln.

### Kapitel 1.6 Überblick und Struktur des Dokumentes

Innerhalb des Überblicks und Struktur des Dokuments wird eine Zusammenfassung der Inhalte, des Aufbaus und der behandelten Themen gegeben. Das Kapitel enthält einen Eintrag und hat somit das Strukturmerkmal Gruppierung. Der Eintrag selber enthält nur Inhalt und ist somit dem Strukturmerkmal Inhalt zuzuweisen.[249]

## Kapitel 2. Allgemeine Beschreibungen

Das Kapitel der allgemeinen Beschreibungen setzt das zu entwickelnde Produkt in Zusammenhang zu seiner Umwelt und gibt einen Überblick der Interaktionen mit der Software und der Operationen der Software. Diese Beschreibungen enthalten Erläuterungen über die Interaktionen und dadurch entstehende Einschränkungen bzw. Abhängigkeiten bezüglich des Produkts. In den allgemeinen Beschreibungen werden keine ausführlichen Definitionen formuliert, sondern nur eine kurze Einführung in das Produkt gegeben. Da dieses Kapitel weitere Unterpunkte strukturiert, wie nachfolgend beschrieben, ist eine gruppierende Struktureigenschaft zuzuweisen.[250]

### Kapitel 2.1 Produktperspektive

Die Produktperspektive beschreibt die Software in Bezug auf die Interaktionen mit der näheren Umwelt. Dazu sind verschiedene Schnittstellen des Produkts mit der Umwelt sowie aus der Umwelt bedingte Abhängigkeiten zu beschreiben. Auch dieses Kapitel weist nur das Strukturmerkmal der Gruppierung auf, da dieses weitere Unterkapitel zusammenfasst.

#### Kapitel 2.1.1 Systemschnittstellen

In den Systemschnittstellen werden die vom Produkt verwendeten Verbindungen aufgelistet und beschrieben in welcher Art die zu erstellende Software diese Schnittstelle verwendet. Ein einzelner Eintrag beinhaltet den Namen und die Beschreibungen der jeweiligen Schnittstelle. Dieses Kapitel gruppiert einzelne Einträge, die den Strukturmerkmalen Wiederholung und Inhalt entsprechen.[251]

---

249. vgl. (IEEE·Computer·Society,·1998,·S.·12)
250. vgl. (IEEE·Computer·Society,·1998,·S.·12)
251. vgl. (IEEE·Computer·Society,·1998,·S.·12f.)

### Kapitel 2.1.2 Nutzerschnittstellen

In den Nutzerschnittstellen werden notwendige Konfigurationskriterien und Optimierungsaspekte jeder Schnittstelle erläutert. Solche Konfigurationskriterien sind beispielsweise Bildschirm-, Fenstergrößen, Menüpunkte oder Report-Inhalte. Ein Eintrag einer Nutzerschnittstelle enthält die Beschreibungen, Konfigurationskriterien und Optimierungsaspekte. Dieses Kapitel gruppiert einzelne Einträge, die den Strukturmerkmalen Wiederholung und Inhalt entsprechen.

### Kapitel 2.1.3 Hardwareschnittstellen

In den Hardware-Schnittstellen werden die Verbindungen von der Software zu den Hardware-Komponenten zu beschrieben, die unter anderem die Beschaffenheit dieser enthalten. Hierfür sind die Geräte aufzuführen, die von der Schnittstelle unterstützt werden und die Funktionsweise ist kurz zu erläutern. Ein Eintrag der Hardware-Schnittstellen enthält einen Namen und eine Beschreibung, die jeweils einfache Inhalte darstellt. Dieses Kapitel gruppiert einzelne Einträge, die den Strukturmerkmalen Wiederholung und Inhalt entsprechen.[252]

### Kapitel 2.1.4 Softwareschnittstellen

In den Software-Schnittstellen werden benötigte Software und die Interaktion mit dieser definiert. Eine solche Software könnte beispielsweise ein Betriebssystem oder ein Datenbanksystem sein und wird mit grundlegenden Daten beschrieben. Diese Daten beinhalten den Namen, eine Abkürzung, die Versionsnummer sowie die Quelle der Software. Im Weiteren erfolgt eine Beschreibung der genutzten Schnittstellen und deren Einsatzzweck, die unter anderem die Nachrichtenart und das Nachrichtenformat einschließen. Wenn bereits eine Dokumentation der Schnittstelle besteht, ist auf diese zu referenzieren. Alle grundlegenden Daten entsprechen hierbei einfachen Inhalten. Dieses Kapitel gruppiert die einzelnen Einträge, die den Strukturmerkmalen Wiederholung und Inhalt entsprechen.

### Kapitel 2.1.5 Kommunikationsschnittstellen

In den KommunikationSchnittstellen werden Protokolle für den Datenaustausch und daraus resultierende Anforderungen erläutert, wie beispielsweise Netzwerkprotokolle. Ein Eintrag in den Kommunikationsschnittstellen enthält einen Namen und eine Beschreibung, die jeweils einfache Inhalte darstellen. Dieses Kapitel gruppiert einzelne Einträge, die den Strukturmerkmalen Wiederholung und Inhalt entsprechen.[253]

### Kapitel 2.1.6 Speichernutzung

In der Speichernutzung werden mögliche Einschränkungen und Limitierungen durch den Speicher in einzelnen Einträgen durch Inhaltselemente beschrieben. Dieses Kapitel gruppiert einzelne Einträge, die den Strukturmerkmalen Wiederholung und Inhalt entsprechen.

---

252. vgl. (IEEE·Computer·Society,·1998,·S.·13)

253. vgl. (IEEE·Computer·Society,·1998,·S.·13f.)

### Kapitel 2.1.7 Softwareeinsatz

Im Software-Einsatz werden die Einsatzmöglichkeiten für die Software beschrieben, beispielsweise welche Vorgänge die Bedienung durch den Nutzern erfordern sowie Beschreibungen von Funktionen zum Verarbeiten, Sichern und Wiederherstellen von Daten. Ein Eintrag stellt die Beschreibungen einzelner Vorgänge in einfacher inhaltlicher Form dar. Dieses Kapitel gruppiert einzelne Einträge, die den Strukturmerkmalen Wiederholung und Inhalt entsprechen.

### Kapitel 2.1.8 Installation und Initialisierung

In der Installation und Initialisierung werden die Anforderungen an die Software am gewünschten Einsatzort beschrieben. Diese geben Maßnahmen vor, die eingehalten werden müssen oder eine Voraussetzung darstellen, um die Software in Betrieb nehmen zu können. Ein Eintrag stellt die Beschreibungen einer Anforderung in einfacher inhaltlicher Form dar. Dieses Kapitel gruppiert die einzelnen Einträge, die den Strukturmerkmalen Wiederholung und Inhalt entsprechen.[254]

### Kapitel 2.2 Produktfunktionen

Die Beschreibungen der Produktfunktionen geben eine Zusammenfassung der wichtigsten Funktionen und Merkmale des Produkts. Es soll keine Detailbeschreibung der Funktionen erfolgen, es können aber einfache Diagramme benutzt werden, wenn es der Erklärung dient, um logische Zusammenhänge einzelner Funktionen zu veranschaulichen. Dieses Kapitel gruppiert einzelne Einträge, die den Strukturmerkmalen Wiederholung und Inhalt entsprechen.

### Kapitel 2.3 Nutzereigenschaften

Damit einzelne Funktionen einer Spezifikation besser nachvollzogen werden können, sind Nutzereigenschaften zu erläutern, die technische Kenntnisse oder Erfahrungen eines Nutzers mit der Materie beschreiben. Hierbei handelt es sich um eine natürlichsprachliche Beschreibung der Nutzertypen. Dieses Kapitel gruppiert einzelne Einträge, die den Strukturmerkmalen Wiederholung und Inhalt entsprechen.

### Kapitel 2.4 Weitere Beschränkungen

Grundsätzlich sind weitere Beschränkungen für die Entwicklermöglichkeiten mit einer kurzen Erläuterung aufzuführen, wie beispielsweise die Limitierung durch Hardware, Schnittstellen zu anderen Applikationen oder politische Richtlinien. Weitere Aspekte können Sicherheitsbedenken, parallele Arbeitsprozesse, Prüfungs- und Kontrollfunktionen sowie Anforderungen an die Zuverlässigkeit der Software sein. Dieses Kapitel gruppiert einzelne Einträge, die den Strukturmerkmalen Wiederholung und Inhalt entsprechen.

254. vgl. (IEEE Computer Society, 1998, S. 14)

### Kapitel 2.5 Annahmen und Abhängigkeiten

Annahmen und Abhängigkeiten, die Einfluss auf die Funktion der Software haben, sind in einem weiteren Abschnitt aufzulisten. Eine dieser Annahmen könnte sein, dass ein spezifisches Betriebssystem vorhanden ist. Sollte sich diese Annahme ändern, muss die Spezifikation der Software angepasst werden, um den neuen Abhängigkeiten gerecht zu werden. Dieses Kapitel gruppiert einzelne Einträge, die den Strukturmerkmalen Wiederholung und Inhalt entsprechen.

### Kapitel 2.6 Zurückgestellte Anforderungen

In einer Entwicklung können Anforderungen bestehen, die erst in einer späteren Produktversion technisch realisiert werden sollen. Damit die Vollständigkeit dieser Anforderungen jedoch erhalten bleibt, werden diese zurückgestellten Anforderungen passend zusammengefasst. Dieses Kapitel gruppiert einzelne Einträge, die den Strukturmerkmalen Wiederholung und Inhalt entsprechen.[255]

## Kapitel 3. Detaillierte Beschreibungen der Anforderungen

Dieses Kapitel wird für eine Ausführung aller zuvor oberflächlich spezifizierten Anforderungen verwendet, die in einer Detailtiefe beschrieben werden, anhand derer ein Entwickler mit den spezifizierten Informationen die Software realisieren kann. Weiterhin müssen die detaillierten Anforderungen testbar sein, sodass mindestens die Eingangsinformationen und Antworten der Software und daran beteiligte Funktionen definiert werden. Zudem sollte die Lesbarkeit, durch die Struktur und Anordnung der Anforderungen, bestmöglich unterstützt werden. Da dieses Kapitel weitere Unterpunkte strukturiert, wie nachfolgend beschrieben, ist eine gruppierende Struktureigenschaft zuzuweisen.[256]

### Kapitel 3.1 Externe Schnittstellen

Durch die externen Schnittstellen erfolgt eine detaillierte Beschreibung über die verschiedenen Eingangs- und Ausgangsinformationen. Dieses Kapitel weist nur das Strukturmerkmal der Gruppierung auf, da dieses weitere Unterkapitel zusammenfasst, in denen die verschiedenen Schnittstellenarten definiert werden.

### Kapitel 3.1.1 - 3.1.4 Nutzer-, Hardware-, Software-, Kommunikationsschnittstellen

Alle in den externen Schnittstellen enthalten Unterkapitel (Nutzer-, Hardware-, Software- und Kommunikationsschnittstellen) weisen jeweils die gleiche interne Struktur auf und werden deshalb zusammengefasst betrachtet. Die Definition einer einzelnen Schnittstelle enthält zahlreiche Elemente, wie beispielsweise Name der Schnittstelle, Beschreibung des Zwecks, Beendigungsnachrichten und viele weitere, die eine detaillierte Beschreibung in inhaltlicher Form darstellen. Diese Elemente werden in einem Eintrag gruppiert, der eine wiederholende Eigenschaft aufweist, da mehrere Schnittstellen existie-

255. vgl. (IEEE Computer Society, 1998, S. 14f.)

256. vgl. (IEEE Computer Society, 1998, S. 16)

ren können. Ein Schnittstellen-Unterkapitel aggregiert die einzelne Einträge, sodass als Strukturmerkmalen die Gruppierung zugeordnet werden kann.[257]

### Kapitel 3.2 Hauptmerkmale der Software

Funktionale Anforderungen beschreiben die genauen Vorgänge innerhalb der Software. Hierfür werden Voraussetzungen für die Annahme und Verarbeitung von Eingangsinformation erläutert und welche Ausgabeinformation zu erwarten sind. Mit Hilfe einer schrittweisen Beschreibung der Transformation dieser Informationen erfolgt eine Erläuterung über den Zusammenhang zwischen Eingabe- und Ausgabeinformationen. Weiterhin wird der sequentielle Ablauf der einzelnen Funktionen, eine Fehlerbehandlung und Behandlung sonstiger abnormaler Situationen beschrieben. Alle genannten Elemente werden inhaltlich beschrieben und anschließend in einem einzelnen Eintrag gruppiert, der eine wiederholende Eigenschaft aufweist, da mehrere Anforderungen existieren können. Abschließend werden die einzelnen Elemente über das gruppierende Strukturelement in diesem Kapitel zusammengefasst.[258]

### Kapitel 3.3 Performanzanforderungen

Mit den Anforderungen an die Performanz wird die Leistungsfähigkeit der Software beschrieben, die im Bezug auf die Interaktionen mit dieser existieren. Beispielsweise wird beschrieben, wie viele Nutzer gleichzeitig auf die Software zugreifen können oder die Menge an parallel zu verarbeitenden Informationen definiert. Ein Eintrag stellt die Beschreibungen einer Anforderung dar. Dieses Kapitel gruppiert einzelne Einträge, die den Strukturmerkmalen Wiederholung und Inhalt entsprechen.

### Kapitel 3.4 Logische Datenbankanforderungen

Mit Hilfe der Anforderungen an die logische Datenbank erfolgt eine Beschreibung über die Struktur und die Daten, die innerhalb eines Datenbanksystems gespeichert werden. Diese Anforderungen enthalten verschiedene Informationen, unter anderem über Datentypen, Entitäten, Beziehungen und weitere, die von gesetzlichen oder betrieblichen Normen definiert werden. Ein Eintrag stellt die Beschreibungen einer Anforderung dar. Dieses Kapitel gruppiert einzelne Einträge, die den Strukturmerkmalen Wiederholung und Inhalt entsprechen.[259]

### Kapitel 3.5 Einschränkungen des Entwurfs

In diesem Kapitel werden die Limitierungen behandelt, die beispielsweise aus Normen oder Hardware resultieren. Hierzu werden Normen und Standards berücksichtigt, welche für die Entwicklung einer Software wichtige Rahmenbedingung vorgeben. Diese werden in passenden Anforderungen erläutert, wie beispielsweise Bedingungen an das Report-Format oder die Datenbezeichnung. Zusätzlich können weitere Einschränkungen spezifiziert werden. Jeder Eintrag stellt die Beschreibungen einer Anforde-

257. vgl. (IEEE·Computer·Society,·1998,·S.·16)
258. vgl. (IEEE·Computer·Society,·1998,·S.·16)
259. vgl. (IEEE·Computer·Society,·1998,·S.·16f.)

rung dar. Dieses Kapitel gruppiert einzelne Einträge, die den Strukturmerkmalen Wiederholung und Inhalt entsprechen.[260]

### Kapitel 3.6 Softwareattribute

Die Eigenschaften einer Software werden innerhalb der Software-Eigenschaften detailliert formuliert. Zusätzlich muss berücksichtigt werden, dass die einzelnen Anforderungen verifizierbar beschrieben werden. Dieses Kapitel weist nur das Strukturmerkmal der Gruppierung auf, da dieses weitere Unterkapitel zusammenfasst, in denen die verschiedenen Eigenschaften definiert werden.

### Kapitel 3.6.1 - 3.6.5 Zuverlässigkeit, Verfügbarkeit, Sicherheit, Wartbarkeit, Portabilität

Alle diese Eigenschaften einer Software, die in den passenden Unterkapitel formuliert werden, weisen jeweils die gleiche interne Struktur auf und werden deshalb zusammengefasst betrachtet. In diesem Kapitel werden die Faktoren beschrieben, mit denen die Zuverlässigkeit, Verfügbarkeit, Sicherheit, Wartbarkeit und Portabilität der Software sichergestellt werden können. Hierbei handelt es sich um eine natürlichsprachliche Beschreibung der Eigenschaften. Diese Kapitel gruppieren einzelne Einträge, die dem Strukturmerkmal Inhalt entsprechen.[261]

### Kapitel 4. Appendix

Das abschließende Kapitel enthält zusätzliche Informationen, die den Inhalt der Dokumentation vervollständigen oder erweitern und aufgrund ihres Umfanges nicht in die zuvor beschriebene Dokumentstruktur eingegliedert werden können. Dies können beispielsweise Referenzen auf Abbildungen oder externe Quellen sein. Alle Eigenschaften der Anhänge werden natürlichsprachlich beschrieben. Diese Kapitel gruppieren einzelne Einträge, die dem Strukturmerkmal Inhalt entsprechen.

In IEEE/ANSI 830-1998 sind verschiedene Empfehlungen zur Strukturierung der detaillierten Beschreibungen der Anforderungen gegeben. Diese Empfehlungen leiten sich aus projektspezifischen Darstellungsbedürfnissen ab. Demnach ist für die Erstellung einer Software empfehlenswert im Vorfeld zu definieren, welche Strukturierung für die Spezifikation am besten eignet ist.[262] Für die Umsetzung der prototypischen Dokumentvorlage muss deshalb entschieden werden, welche der Darstellungsoptionen in der Dokumentvorlage verwendet werden. Im Folgenden sollen jeweils die verschiedenen projektspezifischen Darstellungsoptionen zusammengefasst und die Strukturmerkmale untersucht werden.

### Variante A oder B

Eine Software kann auf Basis verschiedener Zustände arbeiten, die eine erweiterte oder beschränkende Funktion erlauben, wobei sich der Zugriff auf die Software ändern kann. Wenn eine Software auf Basis von verschiedenen Modi arbeitet, ist eine Gliederung nach System- oder Software-Modus vorzuneh-

260. vgl. (IEEE·Computer·Society,·1998,·S.·17)

261. vgl. (IEEE·Computer·Society,·1998,·S.·17f.)

262. vgl. (IEEE·Computer·Society,·1998,·S.·18)

men, sodass die funktionalen Anforderungen nach Modus geteilt und weiter spezifiziert werden. Sollten auch die Schnittstellen betroffen sein, sind diese ebenfalls unter den jeweiligen Modi zu gliedern.[263]

### Variante C

Wird die Funktionalität der Software in Abhängigkeit ausführender Nutzertypen beschrieben, ist als Strukturierung die Darstellung nach Nutzerklassen zu wählen. In dieser Kapitelstruktur werden die Anforderungen entsprechend der Nutzerklassen gruppiert.[264]

### Variante D

Ist der Ablauf der Software-Funktionen nach realen Objekten ausgelegt ist eine Gliederung nach Objekten oder Klassen vorzunehmen. Sollten Objekte gleiche Eigenschaften oder Funktionen nutzen, sind diese Eigenschaften und Funktionen als Klasse zu bündeln. Innerhalb der Struktur werden die Attribute eines Objekts oder einer Klasse beschrieben und jede genutzte Anforderung detailliert spezifiziert.[265]

### Variante E

Findest eine Gliederung der Software nach den Hauptmerkmalen statt, wird zu jedem Merkmal eine einleitende Beschreibung, eine Stimulus-Antwort-Sequenz und die mit diesem Merkmal assoziierten funktionalen Anforderungen formuliert. Es findet deshalb eine Gruppierung anhand dieser Hauptmerkmale statt, in der die einzelnen Elemente zusammengefasst werden.[266]

### Variante F

Anforderungen können nach Stimulus oder Antwort strukturiert werden. Hierzu werden den Stimuli (bzw. Antworten) die verwendeten funktionalen Anforderungen untergeordnet und anhand der Stimuli gruppiert.[267]

### Variante G

Wenn keine der oben beschriebenen Strukturen auf die Software zutreffen, kann eine Aufteilung nach funktionaler Hierarchie vorgenommen werden. Hierbei wird nach Eingangsinformationen, Ausgangsinformationen oder internen Datenzugriffen gegliedert. In diesen Strukturen erfolgt eine inhaltliche Beschreibung weiterer Elemente.[268]

Für den Einsatz in Spezifikationen können die verschiedenen Strukturen kombiniert werden, um die Anforderungen der Software bestmöglich abzubilden.[269] Die Ergebnisse dieser Analyse sind in der Tabelle in

263. vgl. (IEEE·Computer·Society,·1998,·S.·19-21)
264. vgl. (IEEE·Computer·Society,·1998,·S.·19-22)
265. vgl. (IEEE·Computer·Society,·1998,·S.·19-25)
266. vgl. (IEEE·Computer·Society,·1998,·S.·23)
267. vgl. (IEEE·Computer·Society,·1998,·S.·24)
268. vgl. (IEEE·Computer·Society,·1998,·S.·24ff.)
269. vgl. (IEEE·Computer·Society,·1998,·S.·26)

Anhang 7.3 festgehalten. Diese Tabelle dient als Basisdokument zur Umsetzung der Struktur in ein XML-Schema und die anschließende Umsetzung in eine prototypische Dokumentvorlage.

### Übersicht und Evaluation aktueller Textverarbeitungsprogramme

Nachfolgend wird ein Überblick über Software zur Erstellung von natürlichsprachlichen Software-Spezifikationen gegeben. In diesem Zusammenhang wird erörtert welche Software häufig eingesetzt wird mit der mögliche Dokumentvorlagen erstellt werden können. Das Ergebnis der Evaluation soll als Entscheidungskriterium für die Auswahl der Software gelten, mit der die Dokumentvorlagen prototypisch entwickelt werden. Außerdem wird das Speicherformat des Dokuments für die Weiterverarbeitung der Inhalte erläutert und die Werkzeuge zur Erstellung einer Dokumentvorlage vorgestellt.

Für die Erstellung von natürlichsprachlichen Software-Spezifikationen müssen Anforderungen, Funktionen, Diagramme usw. in einem Dokument zusammengeführt werden. Da es sich hierbei zum Großteil um natürlichsprachliche Erläuterungen handelt, neben tabellarischen und grafischen Darstellungen, wird für die Erstellung einer Spezifikationen meist eine Software zur Textverarbeitung verwendet. Eine solche Software unterstützt den Benutzer bei der Erstellung von natürlichsprachlichen Dokumenten, indem Textformatierungsoptionen, Schriftsätze und weitere Hilfsmittel zur Erstellung von Text bereitgestellt werden.

| Paket | Hersteller | Textverarbeitungs-Software | Aktuelle Version | Lizenz |
|---|---|---|---|---|
| Microsoft Office | Microsoft | Microsoft Word[270] | 2013 | proprietär |
| Apache OpenOffice | Apache Software Foundation | Apache OpenOffice Writer[271] | 4.1.1 | Apache License 2.0 (GNU GPL 3.0) |
| iWork | Apple | Pages[272] | 5.5.2 | proprietär |
| WordPerfect Office | Corel | WordPerfect[273] | x7 | proprietär |

Tabelle 17: Übersicht von Software-Paketen mit Textverarbeitungssoftware

Tabelle 17 zeigt aktuelle Software-Pakete verschiedener Hersteller, die ein Programm zur Textverarbeitung beinhalten. Diese können nicht im Internet ausgeführt werden und erfordern eine lokale Installation beim Benutzer. In der Tabelle wird das Software-Paket, der Hersteller, die enthaltene Textverarbeitungssoftware, die aktuelle Version sowie die Lizenzart gelistet. Microsoft Office sowie WordPerfect Office sind für das Windows Betriebssystem entwickelt worden. Zudem bietet Microsoft eine Version von Microsoft Office für Apple Betriebssysteme an, wohingegen das iWork-Paket nur für Apple Betriebssysteme konzipiert ist. Apache OpenOffice ist eine Plattform-unabhängige Software und kann auf allen Betriebsystem

270. vgl. ("Office·Home·and·Business·2013,"·2013)
271. vgl. ("Why·Apache·OpenOffice,"·2014)
272. vgl. ("Mac·App·Store·-·Pages,"·2014)
273. vgl. ("WordPerfect·Office·Suite,"·2014)

verwendet werden. Neben Apache OpenOffice gibt es noch weitere, auf OpenOffice basierende Open-Source Distributionen, wie beispielsweise LibreOffice und StarOffice. Apache OpenOffice soll an dieser Stelle repräsentativ für diese Distributionen stehen.

Für die Entwicklung der prototypischen Dokumentvorlage wird in einem ersten Schritt evaluiert, welche dieser Pakete am weitesten verbreitet sind, um dadurch eine hohe Akzeptanz des zu entwickelnden Prototyps zu erreichen. Abbildung 68 zeigt eine Statistik vom Januar 2010 über die Verteilung von installierten Office-Paketen bei deutschen Internetnutzern.

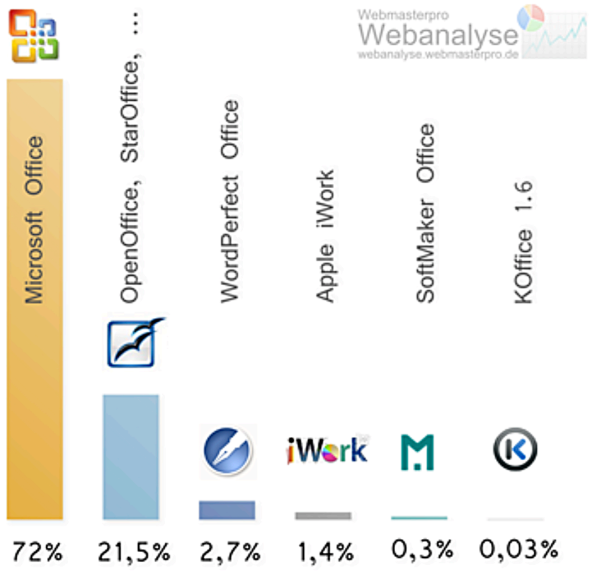



Abb. 68: Verbreitung von Office-Paketen bei deutschen Internetnutzern vom Januar 2010

Diese Daten sind mithilfe von Messprogrammen ermittelt worden, die auf über 100.000 Web-Seiten implementiert wurden, sodass von über einer Millionen Nutzer diese Informationen erhoben werden konnten. Unter den zuvor vorgestellten Paketen sind Microsoft Office, OpenOffice, WordPerfect und iWorks wiederzufinden. Microsoft Office ist hierbei mit einem großen Vorsprung an Installationen in Höhe von 72% hervorzuheben und nimmt damit fast Dreiviertel der Installationen ein. Gefolgt wird Microsoft Office von OpenOffice Installationen oder entsprechenden Distributionen, mit einem Anteil von 21,5%. Alle weiteren Pakete nehmen lediglich einen Anteil von insgesamt 4,13% ein und können von der weiteren Evaluation ausgeschlossen werden.[274]

Aufgrund der starken Verbreitung von Microsoft Office werden im weiteren die verschiedenen Versionen von Microsoft Office evaluiert. Hierzu liefert eine Umfrage unter 148 IT-Entscheidungsträgern aus dem Jahr 2013 passende Informationen, wie in Abbildung 69 gezeigt wird. Hierbei steht im Ergebnis, dass Microsoft Office 2010 mit einem Anteil von 85% fast viermal so oft Verwendung findet wie die aktuelle Version aus dem Jahr 2013. Dies kann insbesondere darauf zurückzuführen sein, dass Microsoft Office 2013 noch nicht solange am Markt verfügbar ist, wie Microsoft Office 2010. Microsoft Office 2007 wur-

274. vgl. ("Why·Apache·OpenOffice,"·2014)

de bei 51% der Befragten angegeben. Die Version Microsoft Office 2003 (oder älter) wird laut der Statistik von 28% der Befragten noch genutzt. Die Macintosh Version von Office erzielt noch 17% in der Befragung.[275] Werden Statistiken über die Verbreitung von Betriebssystemen hinzugezogen, die einen Einfluss auf die Entscheidung zu einem Office-Paket nehmen, zeichnet sich ein ähnliches Bild ab. So gibt netmarketshare.com an, dass 91,48% der untersuchten Desktop-Systemen eine Windows-Installation aufweisen.[276]

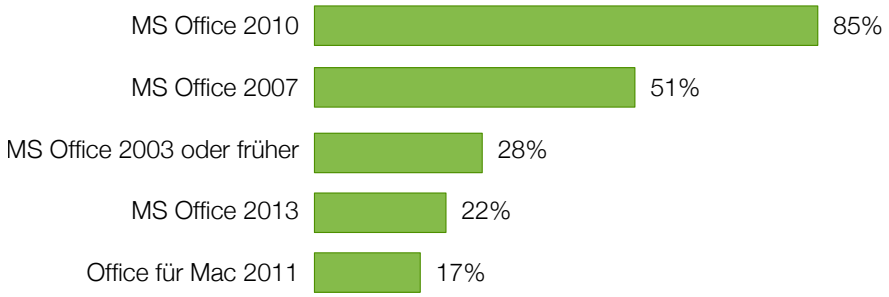



Abb. 69: Aktuell eingesetzte Office Produkte in IT-Unternehmen

Die Evaluation zeigt, dass Microsoft Office eine große Beliebtheit und Verbreitung unter den vorgestellten Software-Paketen aufweist. Ein Entscheidungskriterium stellt hierbei die Kompatibilität von Microsoft Office zu Windows dar. Für die weitere Betrachtung der Software muss entschieden werden, welche Version von Microsoft Office für die Entwicklung benutzt werden soll. Damit eine möglichst hohe Akzeptanz erzielt wird, fällt die Wahl für die prototypische Implementierung der Dokumentvorlage auf die Microsoft Office Version 2010 für das Betriebssystem Windows.

### Konzept und Anforderungen

In diesem Kapitel wird das Konzept zur Erstellung der prototypischen Dokumentvorlage vorgestellt und daran die Anforderungen an das Vorlagendokument von denen einer Spezifikation abgeleitet. Für die Vorbereitung der Entwicklung werden zuerst die Anforderungen an die Entwicklung aufgestellt, anhand derer sich die Erstellung der Dokumentvorlage orientiert und die diese abschließend erfüllt. Die in Kapitel 5.1.1.2 erläuterten Kapitelstrukturen stellen hierbei die Grundlage für die Strukturierung der Dokumentvorlage dar. Da die Nutzerinhalte des Vorlagendokumentes gegen das erstellte Schema geprüft werden sollen, müssen diese auf die jeweiligen XML-Elemente zurückführbar sein, die der Tabelle in Anhang 7.3 entnommen werden können.

Für eine prototypische Realisierung muss jedes identifizierte Strukturmerkmal von einer Funktion erfasst und in die geforderte Dokumentstruktur überführt werden. Hierfür sind innerhalb des Dokuments geeignete Darstellungen der einzelnen Elemente zu definieren.

275. vgl. (Roe, 2013)

276. vgl. ("Operating System Marketshare," 2014)

Die Anforderungen für die Dokumentvorlage lassen sich aus den Kapitel 5.1.1.2 ableiten. Hierbei gilt es zwischen Anforderungen an eine fertige Spezifikation und Anforderungen die an eine Vorlage gestellt werden können zu unterscheiden. Mithilfe dieser Differenzierung werden die erläuterten Attribute und Eigenschaften strukturierter Spezifikationen gemäß der IEEE/ANSI 830-1998 Empfehlungen wie folgt zusammengefasst.

- Das Vorlagendokument sollte eine Kapitelstruktur gemäß IEEE/ANSI 830-1998 realisieren, d.h. in der Vorlage soll die in Kapitel 5.1.1.2 erläuterte Struktur umgesetzt sein und sichergestellt werden, dass diese Struktur bei der Erstellung der Software-Spezifikation eingehalten wird.
- Das Vorlagendokument sollte vollständig sein bezüglich geforderter Angaben und damit verbundener Eingabefelder.
- Das Vorlagendokument sollte auf Konsistenz prüfbar sein, d.h. es soll möglich sein innerhalb der Vorlage erste Konsistenzprüfungen vorzunehmen.
- Das Vorlagendokument sollte mögliche Tests enthalten, bspw "Wurden alle Felder ausgefüllt?".
- Das Vorlagendokument sollte wiederverwendbar sein, d.h. die Vorlage soll im Aufbau und der Implementierung strukturiert und dokumentiert sein, um Anpassungen realisieren zu können.
- Das Vorlagendokument sollte für Folgeprozesse lesbar sein, d.h. die Vorlage soll optimiert sein, um eine Weiterverarbeitung der relevanten Daten zu erleichtern.
- Es soll exemplarisch gezeigt werden, wie einzelne Inhalte ausgelesen und Inhaltsprüfung auf diese gemacht werden können.

Diese Anforderungen sollen innerhalb der prototypischen Dokumentvorlage realisiert werden. Zu diesem Zweck wird in der nachfolgenden Entwicklung auf einzelne Anforderungen eingegangen und herausgehoben, welche Maßnahmen zur Umsetzung der Anforderungen ergriffen wurden.

# 6 Bewertung der Ergebnisse

Zu Beginn des Projektes wurden Testverfahren zur Software-Prüfung ausgewertet, um Grundlagen über theoretische Hintergründe zu erhalten. Hierzu wurden insbesondere sichere Systeme im Zusammenhang mit Fehlerquoten und die damit verbundenen Fehlerentstehungen sowie Fehlerverstärkung im Software-Entwicklungsprozess und den Kostenabschätzungen zur Behebung von Fehlern analysiert. Mit Hilfe dieser Basis sollte ein Grundverständnis über aktuelle Testmethoden, wie beispielsweise statische, dynamische und formale Testmethoden und deren schematischen Ablauf, bezogen auf das schematische Vorgehen (V-Modell) der Software-Entwicklung, aufgezeigt werden. Neben der grundlegenden Analyse von Testmethoden und Verfahrensweisen wurden Normen auf methodische Prozeduren untersucht, die im Zusammenhang mit Software-Prüfung stehen.

Mit dem Ziel der Erforschung von Good Practice Workflows wurden grundlegende Fragen im Zusammenhang mit Software-Prüfung aufgegriffen und mit Hilfe von qualitativen Befragungen behandelt. Insbesondere wurden Aktivitäten berücksichtigt, die Prüfer bzw. Hersteller häufig im Gesamtprozess der

Software-Prüfung durchlaufen müssen, da hier mehrere Iterationen (Erfahrungswerte der Prüfer) notwendig sind, um ein befriedigendes Ergebnis vom Hersteller zu erhalten.

Im Ergebnis der Befragungen stellte sich heraus, dass die Prüfung von C-Software für Embedded Systeme unterstützt werden sollte, da ca. 95% der eingereichten Software vom Hersteller in diesem Format eingereicht werden.

Neben der Festlegung der präferierten Programmsprache wurden Aktivitäten aus dem Prozessdiagramm identifiziert, die durch Werkzeuge auf Hersteller- oder Prüferseite unterstützend eingesetzt werden könnten. Anhand der Befragungen konnten Werkzeuge herausgearbeitet werden, die auf besonders hohe Akzeptanz stoßen.

- Traceabilty
  Unterstützung des Herstellers bei der Verknüpfung von Spezifikation und Quellcode zur eindeutigen Zuordnung von Spezifikationsabschnitten zu entsprechenden Code-Bereichen (bspw. Methoden).
- Versionierung
  Erfassung und Verwaltung verschiedener Dokumentversionen, die durch wiederholte Durchführung von Aktivitäten im Prozessablauf vom Hersteller entstehen können.
- Differenzvisualiserung
  Visualisierung von Differenzen zwischen verschiedenen Dokumentversionen, zur Sicherstellung der Überprüfung aller veränderten Informationen.
- Vollständigkeitsprüfung
  Überprüfung eingereichter Dokumente, insbesondere der Spezifikation, auf formale Vollständigkeit, der einzelnen Kapitelinhalte.

Weiterhin wurde ein Ansatz entwickelt, mit dem die Güte von Formaten zur Beurteilung von Software-Prüfdokumentationen in Form eines Qualitätsindexes möglich werden soll. Im Ergebnis liegt ein Katalog mit gesammelten Kriterien zur Beurteilung der Qualität von Dokumentformaten vor (siehe Kapitel 5.3), anhand dessen eine Bewertung erfolgen kann.

Nach der Konzeption des Qualitätsindex, der damit verbundenen Kriterien und der bisher durchgeführten Befragungen wurde ein Entwurf für ein formales Dokumentmodell konzipiert. Das Ziel war es, ein Format zu spezifizieren, welches sämtliche Eigenschaften und Inhalte eines Prüfvorgangs von Software in einem einzelnen Dokumentmodell abbilden kann. Ausgehend von den gesammelten Informationen und Erkenntnissen erfolgte ein Entwurf für ein Dokumentformat. Dieser Entwurf sieht ein Container-Format vor, das es erlaubt Informationen der Software-Prüfung strukturiert zu vereinen, indem es verschiedenste Dokumentarten aufnimmt, wie beispielsweise Spezifikationsdokumente, Quellcode oder Diagramme.

Im nächsten Schritt wurde auf Basis der vorangegangenen und aktuell durchgeführten Arbeiten eine Bewertung verschiedener vorhandener BUS-Systeme durchgeführt, damit eine Einschätzung über ein geeignetes Basis-System erfolgen kann. Als Grundlage für die Bewertung und Konzeption dienen Anforderungen, die aus den qualitativen Befragungen resultieren, woraus ein Entwurf für das zentrale BUS-System

der SIWOB-Workbench entwickelt wurde und als Grundlage für die prototypische Realisierung fungiert. Mit Hilfe des BUS-Systems sollen letztendlich Prüfprozesse mit eigens entwickelten Werkzeugen unterstützt werden, wobei in diesem Projektverlauf der Fokus auf die Software-Prüfung als Teil einer exemplarischen Baumusterprüfung fällt.

## 6.1 BUS-System

Die etablierten Werkzeuge zur Unterstützung eines Software-Prüfers wurden nach dem Prinzip der serviceorientierten Architektur entwickelt und ordnen sich demnach als Menge von losen Services an. Um diese Services zu adressieren wurde ein BUS-System entwickelt (siehe Kapitel 5.5.1), welches die vorhandenen Werkzeuge verknüpft, sodass dieses zur Unterstützung von Software-Prüfverfahren beiträgt.

Welche Art von BUS-System für eine solche Architektur verwendet werden sollte, wurde im Rahmen dieses Forschungsprojekts erläutert.[277] Unter Gesichtspunkten einer moderneren Architektur stellte sich der Enterprise Service Bus für einen möglichen Einsatz innerhalb des Forschungsprojekts heraus. Dieser verfügt über Techniken die in einer Service orientierten Architektur dringend notwendig sind. Dabei können mögliche Abläufe der Services gesteuert, Daten zwischen einzelnen Services übermittelt oder flexibel neue Werkzeuge hinzugefügt werden.

Auf Basis der Charakteristiken eines Enterprise Service Bus und dem Kriterium Open Source wurde der Open ESB als Enterprise Service Bus verwendet, der die prototypisch realisierten Werkzeuge zur Unterstützung von Software-Prüfverfahren optimal unterstützt. Dieser wurde mit Hilfe einer Nutzwertanalyse ermittelt, welche die zu Grunde liegenden Anforderungen als zu bewertende Kriterien berücksichtigte (siehe Kapitel 5.5.1).

Im weiteren Verlauf konnte die Konzeption und Entwicklung des gesamten BUS-Systems und der Werkzeuge stattfinden. Zunächst wurden benötigte WSDL-Dateien für das BUS-System definiert. Anschließend konnten Prozessketten mit BPEL erstellt und in das BUS-System integriert werden. Daraufhin wurde das Cluster konzipiert und auf den physischen Rechnersystemen konfiguriert. Nachdem das System prototypisch konfiguriert werden konnte, wurden erfolgreich Software-Tests sowie das Debugging einzelner Prozessketten durchgeführt.

Die wissenschaftliche Fragestellung, ob ein BUS-System zur Verknüpfung von Werkzeugen zur Unterstützung von Software-Prüfverfahren prototypisch konzipiert und entwickelt werden kann, wurde im Rahmen dieses Forschungsprojekts unter Verwendung des Open ESB erfolgreich beantwortet. Somit konnte ein redundantes BUS-System entwickelt werden, welches alle geforderten Anforderungen in prototypischem Umfang adressiert. Weiterhin wurde nicht nur ein prototypisches redundantes BUS-System für die Verknüpfung von Werkzeugen zur Unterstützung von Software-Prüfverfahren entwickelt, sondern ebenfalls wichtige Redundanz der etablierten Werkzeuge erzielt.

277. vgl. (Sayegh, 1997, S. 12)

## 6.2 SIWOB Werkzeuge

Das BUS-System stellt die zentrale Infrastruktur dar, an die Werkzeuge gekoppelt werden können. Dadurch wird eine Kommunikation zwischen den einzelnen Werkzeugen möglich und Informationen können untereinander ausgetauscht werden. Die Aufgabe für BUS-System und die entwickelten Werkzeuge liegt in der Unterstützung von Herstellern und Prüfern bei einer Baumusterprüfung im Teilgebiet der Software-Prüfung. Dies umfasst verschiedene Aufgaben, wie beispielsweise die Erzeugung von prüfungsgerechten Spezifikationen oder die Anzeige- und Darstellungsmöglichkeiten von Informationszusammenhängen innerhalb oder zwischen einzelnen Prüfvorgängen. Weiterhin verhält sich das BUS-System für Nutzer transparent und stellt hohe Verfügbarkeit, Flexibilität sowie erhöhte Performance in den Mittelpunkt seiner Funktionalitäten. Im folgenden werden die Ergebnisse der einzelnen Werkzeuge kurz zusammenfasst.

- Repository (Kapitel 5.5.2)

  Das Repository stellt ein Basiswerkzeug von SIWOB dar, mit dem die Verwaltung von Dateien auf Dateisystemebene realisiert wird. Die verwalteten Dateien werden darüber hinaus durch umfangreiche Meta-Daten erweitert, sodass dem Benutzer zusätzliche Informationen bereitgestellt werden können. Grundlegende Funktionen, wie beispielsweise Create-Read-Update-Delete (CRUD) etc., stellen die Basisfunktionen dar, die eine einfache Verwaltung der Dokumente und ihrer Meta-Daten sicherstellen. Neben der Verwaltung von Daten und Meta-Daten ist das Repository auch in der Lage Dateien in verschiedenen Versionen zu verwalten, sodass die Möglichkeit besteht Veränderungen zwischen Dokumenten nachzuhalten.

- Differenzbildung (Kapitel 5.5.5)

  Mit der Differenzbildung kann grundsätzlich jedes einfache Plain-Text Dokument für einen Vergleich verwendet werden, wodurch Dokumente mit Programm-Strukturen (C-Dateien etc.) und einfache Textinformationen abgedeckt werden. Ein Vergleich von komplexen Text-Dokumenten, wie beispielsweise Word-Dokumente, lässt sich nicht oder nur mit deutlichen Einschränkungen realisieren. Die Bildung einer Differenz zwischen zwei ausgewählten Dokumente erfolgt automatisiert, sofern es sich um unterschiedliche Versionen handelt. Weiterhin können gängige Dateiformate miteinander verglichen werden, sodass der Nutzer ohne aufwändige Vorbereitung eine Differenzbildung ausführen kann. Das Ergebnis der Differenzberechnung wird anschließend innerhalb des User Interfaces für den Benutzer visualisiert. Dieses beinhaltet eine Ansicht in der zeilenbasiert dargestellt wird, welche Dokumente Änderungen beinhalten und an welcher Stelle sich diese befinden, sodass der Benutzer eine zusammengefasste Ansicht der Ergebnisse erhält.

- Comment System (Kapitel 5.5.6)

  Das Kommentarsystem bietet Benutzern (Hersteller und Prüfer) die Möglichkeit Dokumente mit Anmerkungen oder Hinweisen zu ergänzen, die im Repository verwaltet werden. Durch diese Option können Benutzer beispielsweise auf fehlerhafte Inhalte hinweisen und diese dokumentieren. Ebenso können klärende Anmerkungen oder Fragen, über unklare Formulierungen, in Kommentarform festgehalten werden. Es besteht die Möglichkeit zu jeder einzelnen Datei einer Baumusterprüfung Kommentare notieren zu können, wie etwa Quellcode-Dateien oder die Systemspezifikation, aber

auch Grafiken und Bauzeichnungen. Kommentierungen des Prüfers dienen insbesondere dazu Fehler, Anmerkungen oder Fragen schriftlich festzuhalten, damit ein Hersteller auf dieser Basis Verbesserungen an seinem Produkt und der Dokumentation durchführen kann.

- Spezifikationsgenerierung (Kapitel 5.5.7)

  Auf Basis etablierter Spezifikationsstandards wurde ein Werkzeug entwickelt, das ein hybrides Spezifikationsformat generiert. Dieses ist auf der einen Seite menschenlesbar und wird in Form eines PDF-Dokumentes zur Verfügung gestellt. Auf der anderen Seite ist dieses Format auch maschinenlesbar, indem es komplexe Informationen in einer XML-Struktur abgebildet. Für die Erstellung einer Spezifikation wurde eine komplexe Schnittstelle für das User Interface konzipiert, sodass ein Hersteller eine intuitive Eingabe-Maske erhält. Diese Maske visualisiert dem Benutzer ein Rahmendokument, in das alle Informationen einer Spezifikation, in einem mehrteiligen Arbeitsschritt, eingeben werden können. Der Abschluss der Erstellung endet immer in der Generierung einer Spezifikation im SIWOB-PDF Format. Hierbei ist hervorzuheben, dass die Erstellung einer Spezifikation durch eine Vollständigkeitsprüfung unterstützt wird. Dies ermöglicht es dem Hersteller ein Dokument zu erstellen, welches formal vollständig ist und nimmt dem Prüfer diese Untersuchung der Vollständigkeit ab, sodass dieser direkt mit der inhaltlichen Prüfung beginnen kann. Für die weitere Bearbeitung des SIWOB-PDF bestehen anschließend zwei Möglichkeiten. Indem das SIWOB-PDF im User Interface geöffnet wird, kann eine vorhandene Spezifikation fertiggestellt oder überarbeitet werden, wobei die maschinenlesbaren Informationen der enthaltenen XML-Datei verwendet werden. Wird das PDF-Dokument mit einem PDF-Viewer (wie z.B. Adobe Acrobat) geöffnet, wird lediglich der menschenlesbare, formatierte Teil sichtbar. Des Weiteren kann ein Prüfer, ebenfalls via SIWOB User Interface, auf den maschinenlesbaren XML-Anteil eines SIWOB PDFs zugreifen. Allerdings wird nur ein eingeschränkter Zugriff gewährt, sodass ein Prüfer keine Möglichkeit erhält inhaltliche Änderungen vorzunehmen. Es ist ihm aber erlaubt Kommentare zu beliebigen Elementen des Dokuments hinzufügen, die elementgenau platziert werden können und dem Hersteller visualisiert werden. Das generierte SIWOB PDF-Dokument kann mit diesen zusätzlichen Informationen zwischen Hersteller und Prüfer ausgetauscht und anhand dieser weiter bearbeitet werden, bis ein für den Prüfer akzeptables Dokument vorliegt.

- Traceability (Kapitel 5.5.8)

  Das Traceability Werkzeug unterstützt Prüfer bei der Auswertung von Quelltexten und der dazu passenden Software-Spezifikation. Dieses Werkzeug implementiert Funktionen, die es dem Hersteller ermöglichen Anforderungen mit korrespondierendem Quellcode-Methoden in Verbindung zu setzten, da diese in unterschiedlichen Dokumenten vorliegen. In einem vorgelagerten Prozessschritt werden hierzu die Quellcode-Dateien mit Hilfe eines Parsers analysiert, der die einzelnen Methoden-Bestandteile einer Quellcode-Datei identifiziert. Anschließend erfolgt durch den Hersteller eine Verknüpfung zwischen den ermittelten Quellcode-Methoden und den Anforderungen aus der Spezifikation. Durch diese Zuordnung von Elementen, wird dem Prüfer die eigentliche Analyse der Quelltexte und der Spezifikation erleichtert, da für diesen die notwendige Verknüpfungsarbeit entfällt. Wenn ein Hersteller diese verknüpften Dokumente beim Prüfer einreicht, kann dieser ohne Zusatzaufwand die einzelnen Anforde-

rungen der Software-Spezifikation hinsichtlich ihrer Konformität zum Quellcode prüfen und Abweichungen sowie Fehler in Form von Anmerkungen notieren.

- User Interface (Kapitel 5.5.3)

  Das User Interface bietet allen Benutzern eine einheitliche Bedienoberfläche, welche gängige Web-Technologien verwendet. Über diese visuelle Schnittstelle erhalten Benutzer Zugriff auf die gesamten Funktionen von SIWOB, die durch die verschiedenen Werkzeuge angeboten werden. Beispiele hierfür stellen die Schnittstelle zur Generierung von Spezifikationen im SIWOB PDF-Format dar, die Verwaltung von Daten über das Repository, die Möglichkeiten zur Ein- und Ausgabe von Dokumenten, die Abfrage von Statusinformationen und viele weitere Optionen. Das User Interface entspricht dabei den aktuellen Anforderungen der Usability-Forschung unter Verwendung des Open Source Webframework Bootstrap. Die Verwendung dieser standardisierten Komponenten ermöglicht eine einheitliche Visualisierung, wodurch den Benutzern eine standardisierte Oberfläche zur Ausgabe im Web-Browser zur Verfügung gestellt wird. Diese unterstützt die Ausgabe für vier unterschiedliche Geräte-Kategorien, dem Smartphone, dem Tablet und dem Desktop-PC mit kleiner sowie großer Anzeigefläche. Dadurch wird die Erwartungskonformität und der Wiedererkennungswert von gleichen Funktionen für die Benutzer von SIWOB verbessert.

## 6.3 Word Spezifikationsvorlage

Zusätzlich zu den Werkzeugen der SIWOB Workbench wurde im Forschungsprojekt eine prototypische Dokumentvorlage entwickelt (siehe Kapitel 5.5.9), die den Prozess der Spezifikationserstellung unterstützen soll.

Die prototypische Dokumentvorlage wurde auf Basis der empfohlenen IEEE/ANSI 830-1998 Struktur realisiert. Zu diesem Zweck wurden Grundlagen von Spezifikationen sowie notwendige Techniken zur Darstellung von Strukturen analysiert, anhand derer die Norm untersucht und eine Gliederung sowie passende Strukturmerkmale abgeleitet wurden. Die Ergebnisse der Analyse wurden in eine formale Auszeichnungssprache (XML) überführt, sodass die Architektur dieser Norm, wie beispielsweise Kapitel, Tabellen, Inhaltselemente usw., in einem Schema-Dokument abgebildet werden konnten.

Anschließend musste die erarbeitete Struktur in eine Vorlage für ein Textverarbeitungsprogramm überführt werden, damit ein Benutzer die Möglichkeit erhält die Spezifikation anhand der vorgegeben Struktur zu erstellen. Mittels einer umfangreichen Evaluation, der Installationszahlen von Textverarbeitungsprogrammen, wurde Microsoft Word 2010 als das Zielprogramm identifiziert, welches die größte Verbreitung aufweist, in der daraufhin die Dokumentvorlage erstellt wurde.

Nach den grundlegenden Analysen wurde die formale XML-Struktur vollständig in ein Word Dokument übertragen und mit zusätzlichen Steuerelementen erweitert. Diese Steuerelemente vereinfachen die Eingaben für den Benutzer dahingehend, dass eine Bearbeitung der Inhalte ausschließlich innerhalb fester Strukturvorgaben erfolgen kann. Ein Benutzer kann mit Hilfe dieser Strukturvorlage alle Inhalte für eine Spezifikation passend eingliedern.

Darüber hinaus können, durch die Formalisierung und Struktureingrenzung der Spezifikation, die Inhalte einer Spezifikation durch geeignete Funktionen überprüft werden. Hierzu wurden Prüffunktionen entwickelt, mit deren Hilfe beispielsweise die Vollständigkeit von Benutzereingaben formal getestet werden können. Zu diesem Zweck werden alle Elemente der Strukturvorlage auf vorhandene oder nicht vorhandene Inhalte verifiziert und die Ergebnisse dem Benutzer mit passenden Hinweisen visualisiert.

Abschließend können die Inhalte und Strukturen der Dokumentvorlage in ein unveränderliches Ausgabeformat (PDF-Dokument) konvertiert und zur Prüfung eingereicht werden. Aufgrund der starken Strukturierung besteht zusätzlich die Option alle Inhalte in ein XML-Dokument als Ausgabeformat zu überführen und dieses für weitere Verarbeitungsschritte in der SIWOB-Workbench bereitzustellen.

Die prototypische in Word entwickelte Dokumentvorlage zeigt, dass die Inhalte einer Spezifikation innerhalb einer definierten Struktur abgebildet und formal geprüft sowie in passende Ausgabeformate gespeichert werden können.

## 6.4 Ausblick

Für zukünftige Vertiefungen der Ergebnisse des Forschungsprojektes SIWOB bieten sich eine Reihe erfolgversprechender, anknüpfender Untersuchungen an.

Allem voran steht die Analyse des entwickelten Prototypen, mit dem Ziel der vollständigen Konsolidierung. Hierzu ist es notwendig, alle Bereiche des Prototypen zu untersuchen, um Optimierungsansätze zu identifizieren und erstrebenswerte Verbesserungsmöglichkeiten gemeinsam mit Prüfinstituten und interessierten Herstellern zu erproben. Bei diesen Erprobungsaktivitäten sollten insbesondere automatisierte Testmethodik zum Einsatz kommen und durch die Beteiligung von Herstellern die Erfordernisse einer späteren Verwendung im praktischen Einsatzumfeld umfänglich berücksichtigt werden.

Das BUS-System, welches anhand des Architekturmodells eines ESB realisiert wurde, stellt einen wichtigen Bestandteil des Prototypen dar, weil es die Verknüpfung der SIWOB-Werkzeuge gewährleistet. Da die verwendeten ESB-Ansätze auf unterschiedlichen Techniken basieren, wäre eine Vereinheitlichung der Architektur mit etablierten Standards eine weitere, sinnvolle Maßnahme zur Konsolidierung der Testinfrastruktur.

Parallel zur technischen Konsolidierung des SIWOB-Ansatzes wäre die Untersuchung möglicher Hürden und Eintrittsbarrieren für einen SIWOB-Einsatz aus Unternehmenssicht eine weitere sehr sinnvolle Maßnahme. Insbesondere sollten dabei Bereitstellungstechniken für SIWOB evaluiert werden. Kandidaten für solche Bereitstellungstechniken wären beispielsweise der Einsatz eines Application Servers, einer unabhängigen Anwendung oder anderer Technologien. Das Ergebnis der Untersuchungen sollte einen Best-Practice-Ansatz liefern, aus dem abgeleitet werden kann, wie SIWOB mit den verfügbaren Werkzeugen aufgebaut sein sollte, damit die Bereitschaft für einen Einsatz innerhalb von Unternehmen möglichst groß wird.

Ausgehend von den prototypischen SIWOB-Ergebnissen sollte bei Konsolidierungsmaßnahmen zwischen grundlegenden und optionalen Funktionen unterschieden werden. Zu diesem Zweck sollten weitere Her-

steller und Prüfer befragt werden, damit analysiert werden kann, von welchen prototypisch entwickelten Werkzeugen und Funktionen der größte Nutzen erwartet wird. Auf Basis dieser Analyse könnte eine Konsolidierung betrieben werden, die solche Funktionen priorisiert, mit denen der größte, potentielle Nutzen assoziiert wird.

Eine weitere Alternative für Konsolidierungsmaßnahmen könnte eine mobile Software für den flexiblen Einsatz vorsehen, beispielsweise als USB-Stick. Mit diesem Ansatz könnten alternative Konzepte in Betracht gezogen werden, die mit einer lokalen oder verteilten Installation von SIWOB nicht möglich wären. Beispielsweise könnten die Hürden für eine Verwendung der SIWOB-Werkzeuge gesenkt werden, da keine aufwändige Installation oder Integration in vorhandene Infrastrukturen notwendig wären und gleichzeitig eine Abgrenzung zwischen SIWOB-Software und internen Daten realisiert werden würde. Zusätzlich könnte die Wartbarkeit der Software vereinfacht werden, indem eine Verteilung von Updates wiederum per USB-Stick erfolgen könnte. Ein derartiger Ansatz bedarf einer Evaluation mit entsprechenden, beteiligten Parteien, sodass eine bestmögliche Lösung vorbereitet werden könnte.

Neben den bereits genannten Aspekten, bietet die prototypisch realisierte Word Spezifikationsvorlage eine Reihe weiterer Möglichkeiten zur Konsolidierung der SIWOB-Ergebnisse. Insbesondere sollte die Verwendung moderner Programmiersprachen als Ersatz für das aktuell eingesetzte Visual Basic for Applications untersucht werden. Weiterhin wäre eine Vergrößerung der Einsatzplattformen über die Grenzen von Microsoft Word hinaus erstrebenswert. Für ein größtmögliches Einsatzgebiet dieser Formularvorlage sollten etablierte Textverarbeitungsprogramme mit nicht-proprietären Lizenzen in Betracht gezogen und Techniken herausgefiltert werden, welche auch eine einheitliche Vorlage für Open Source Textverarbeitungen zulassen. In diesem Zusammenhang sollte eine Analyse der einzelnen Arbeitsschritte der Vorlage stattfinden, damit Optimierungen einzelner Methoden folgen könnten. Abschließend müsste auch die Formularvorlage für Spezifikationen mit interessierten Partnern erprobt werden.

Die ausgeführten Ansatzpunkte für mögliche, weitere Untersuchungen, Optimierungen und sonstige Konsolidierungsmaßnahmen zeigen die Potentiale von SIWOB auf. Diese erstrecken sich in vielfältige Richtungen und zeigen, dass der grundlegende SIWOB-Ansatz für verschiedene Einsatzfelder optimiert und evaluiert werden könnte.

# 7 Anhang

## 7.1 XSD-Schema für eine Software-Spezifikation

XSD-Schema konzipiert nach IEEE 830 und Befragungsergebnissen als XSD-Baum in XML-Form

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified">
    <xs:element name="specification">
        <xs:complexType>
            <xs:sequence>
                <xs:element name="document-control">
                    <xs:complexType>
                        <xs:sequence>
                            <xs:element name="approval" type="xs:string"/>
                            <xs:element name="change-control">
                                <xs:complexType>
                                    <xs:sequence>
                                        <xs:element name="initial-release" type="xs:string"/>
                                        <xs:element name="current-release" type="xs:string"/>
                                        <xs:element name="last-review" type="xs:string"/>
                                        <xs:element name="next-review" type="xs:string"/>
                                        <xs:element name="next-update" type="xs:string"/>
                                    </xs:sequence>
                                </xs:complexType>
                            </xs:element>
                            <xs:element name="revision-history">
                                <xs:complexType>
                                    <xs:sequence>
                                        <xs:element name="revision-list">
                                            <xs:complexType>
                                                <xs:choice maxOccurs="unbounded" minOccurs="0">
                                                  <xs:element name="revision">
                                                  <xs:complexType>
                                                  <xs:sequence>
                                                  <xs:element name="revision-number"
                                                  type="xs:string"/>
                                                  <xs:element name="revision-date" type="xs:string"/>
                                                  <xs:element name="description-of-change"
                                                  type="xs:string"/>
                                                  <xs:element name="author-modifier"
                                                  type="xs:string"/>
                                                  </xs:sequence>
                                                  </xs:complexType>
                                                  </xs:element>
                                                </xs:choice>
                                            </xs:complexType>
                                        </xs:element>
                                    </xs:sequence>
                                </xs:complexType>
                            </xs:element>
                            <xs:element name="distributor">
                                <xs:complexType>
                                    <xs:sequence>
                                        <xs:element name="distribution-list">
                                            <xs:complexType>
                                                <xs:choice maxOccurs="unbounded" minOccurs="0">
                                                  <xs:element name="distribution">
                                                  <xs:complexType>
                                                  <xs:sequence>
                                                  <xs:element name="recipient-name" type="xs:string"/>
                                                  <xs:element name="recipient-organisation"
                                                  type="xs:string"/>
                                                  <xs:element name="distribution-method"
                                                  type="xs:string"/>
                                                  </xs:sequence>
                                                  </xs:complexType>
                                                  </xs:element>
                                                </xs:choice>
                                            </xs:complexType>
                                        </xs:element>
                                    </xs:sequence>
                                </xs:complexType>
```

```
            </xs:element>
        </xs:sequence>
    </xs:complexType>
</xs:element>
<xs:element name="introduction">
    <xs:complexType>
        <xs:sequence>
            <xs:element name="introduction-content" type="xs:string"/>
            <xs:element name="introduction-children">
                <xs:complexType>
                    <xs:sequence>
                        <xs:element name="purpose" type="xs:string"/>
                        <xs:element name="scope" type="xs:string"/>
                        <xs:element name="definitions-acronyms-and-abbrevations">
                            <xs:complexType>
                                <xs:sequence>
                                  <xs:element
                                  name="definitions-acronyms-and-abbrevations-content"
                                  type="xs:string"/>
                                  <xs:element
                                  name="definitions-acronyms-and-abbrevations-children">
                                  <xs:complexType>
                                  <xs:sequence maxOccurs="1">
                                  <xs:element name="definition-list">
                                  <xs:complexType>
                                  <xs:choice maxOccurs="unbounded" minOccurs="0">
                                  <xs:element name="definition">
                                  <xs:complexType>
                                  <xs:sequence>
                                  <xs:element name="term" type="xs:string"/>
                                  <xs:element name="definition" type="xs:string"/>
                                  </xs:sequence>
                                  </xs:complexType>
                                  </xs:element>
                                  </xs:choice>
                                  </xs:complexType>
                                  </xs:element>
                                  <xs:element name="acronym-list">
                                  <xs:complexType>
                                  <xs:choice maxOccurs="unbounded" minOccurs="0">
                                  <xs:element name="acronym">
                                  <xs:complexType>
                                  <xs:sequence>
                                  <xs:element name="term" type="xs:string"/>
                                  <xs:element name="acronym" type="xs:string"/>
                                  </xs:sequence>
                                  </xs:complexType>
                                  </xs:element>
                                  </xs:choice>
                                  </xs:complexType>
                                  </xs:element>
                                  </xs:sequence>
                                  </xs:complexType>
                                  </xs:element>
                                </xs:sequence>
                            </xs:complexType>
                        </xs:element>
                        <xs:element name="references">
                            <xs:complexType>
                                <xs:sequence>
                                  <xs:element name="references-content"
                                  type="xs:string"/>
                                  <xs:element name="references-children">
                                  <xs:complexType>
                                  <xs:sequence>
                                  <xs:element name="references-list">
                                  <xs:complexType>
                                  <xs:choice maxOccurs="unbounded" minOccurs="0">
```

```
                                    <xs:element name="reference">
                                    <xs:complexType>
                                    <xs:sequence>
                                    <xs:element name="author" type="xs:string"/>
                                    <xs:element name="title" type="xs:string"/>
                                    <xs:element name="subtitle" type="xs:string"/>
                                    <xs:element name="number" type="xs:string"/>
                                    <xs:element name="date" type="xs:string"/>
                                    <xs:element name="publisher" type="xs:string"/>
                                    <xs:element name="url" type="xs:string"/>
                                    <xs:element name="place" type="xs:string"/>
                                    <xs:element name="year" type="xs:string"/>
                                    <xs:element name="pages" type="xs:string"/>
                                    </xs:sequence>
                                    </xs:complexType>
                                    </xs:element>
                                    </xs:choice>
                                    </xs:complexType>
                                    </xs:element>
                                    </xs:sequence>
                                    </xs:complexType>
                                    </xs:element>
                                </xs:sequence>
                            </xs:complexType>
                        </xs:element>
                        <xs:element name="overview" type="xs:string"/>
                    </xs:sequence>
                </xs:complexType>
            </xs:element>
            <xs:element name="introduction-comments">
                <xs:complexType>
                    <xs:choice maxOccurs="unbounded" minOccurs="0">
                        <xs:element name="introduction-comment">
                            <xs:complexType>
                                <xs:sequence>
                                  <xs:element name="id" type="xs:int"/>
                                  <xs:element name="date" type="xs:string"/>
                                  <xs:element name="user" type="xs:string"/>
                                  <xs:element name="email" type="xs:string"/>
                                  <xs:element name="comment" type="xs:string"/>
                                  <xs:element name="processed" type="xs:boolean"/>
                                </xs:sequence>
                            </xs:complexType>
                        </xs:element>
                    </xs:choice>
                </xs:complexType>
            </xs:element>
        </xs:sequence>
    </xs:complexType>
</xs:element>
<xs:element name="overall-description">
    <xs:complexType>
        <xs:sequence>
            <xs:element name="overall-description-content" type="xs:string"/>
            <xs:element name="overall-description-children">
                <xs:complexType>
                    <xs:sequence>
                        <xs:element name="product-perspective">
                            <xs:complexType>
                                <xs:sequence>
                                  <xs:element name="product-perspective-content"
                                  type="xs:string"/>
                                  <xs:element name="product-perspective-children">
                                  <xs:complexType>
                                  <xs:sequence>
                                  <xs:element name="detailed-methodology"
                                  type="xs:string"/>
                                  <xs:element name="system-interfaces">
```

```
<xs:complexType>
<xs:sequence>
<xs:element name="system-interfaces-content"
type="xs:string"/>
<xs:element name="system-interfaces-children">
<xs:complexType>
<xs:sequence>
<xs:element name="system-interfaces-list">
<xs:complexType>
<xs:choice maxOccurs="unbounded" minOccurs="0">
<xs:element name="system-interface">
<xs:complexType>
<xs:sequence>
<xs:element name="specification" type="xs:string"/>
<xs:element name="description" type="xs:string"/>
</xs:sequence>
</xs:complexType>
</xs:element>
</xs:choice>
</xs:complexType>
</xs:element>
</xs:sequence>
</xs:complexType>
</xs:element>
</xs:sequence>
</xs:complexType>
</xs:element>
<xs:element name="user-interfaces"
type="xs:string"/>
<xs:element name="hardware-interfaces">
<xs:complexType>
<xs:sequence>
<xs:element name="hardware-interfaces-content"
type="xs:string"/>
<xs:element name="hardware-interfaces-children">
<xs:complexType>
<xs:sequence>
<xs:element name="hardware-interfaces-list">
<xs:complexType>
<xs:choice maxOccurs="unbounded" minOccurs="0">
<xs:element name="hardware-interface">
<xs:complexType>
<xs:sequence>
<xs:element name="specification" type="xs:string"/>
<xs:element name="description" type="xs:string"/>
</xs:sequence>
</xs:complexType>
</xs:element>
</xs:choice>
</xs:complexType>
</xs:element>
</xs:sequence>
</xs:complexType>
</xs:element>
</xs:sequence>
</xs:complexType>
</xs:element>
<xs:element name="software-interfaces">
<xs:complexType>
<xs:sequence>
<xs:element name="software-interfaces-content"
type="xs:string"/>
<xs:element name="software-interface-children">
<xs:complexType>
<xs:sequence>
<xs:element name="software-product-list">
<xs:complexType>
<xs:choice maxOccurs="unbounded" minOccurs="0">
```

```
                <xs:element name="software-product">
                <xs:complexType>
                <xs:sequence>
                <xs:element name="name" type="xs:string"> </xs:element>
                <xs:element name="mnemonic" type="xs:string"> </xs:element>
                <xs:element name="specification-number"
                type="xs:string"> </xs:element>
                <xs:element name="version-number" type="xs:string"
                > </xs:element>
                <xs:element name="source" type="xs:string">
                </xs:element>
                </xs:sequence>
                </xs:complexType>
                </xs:element>
                </xs:choice>
                </xs:complexType>
                </xs:element>
                <xs:element name="interfaces-list">
                <xs:complexType>
                <xs:choice maxOccurs="unbounded" minOccurs="0">
                <xs:element name="interface">
                <xs:complexType>
                <xs:sequence>
                <xs:element name="discussion" type="xs:string"/>
                <xs:element name="definition" type="xs:string"/>
                </xs:sequence>
                </xs:complexType>
                </xs:element>
                </xs:choice>
                </xs:complexType>
                </xs:element>
                </xs:sequence>
                </xs:complexType>
                </xs:element>
                </xs:sequence>
                </xs:complexType>
                </xs:element>
                <xs:element name="communication-interfaces"
                type="xs:string"/>
                <xs:element name="memory-constrains"
                type="xs:string"/>
                <xs:element name="operations" type="xs:string"/>
                <xs:element name="side-adaption-requirements"
                type="xs:string"/>
                </xs:sequence>
                </xs:complexType>
                </xs:element>
            </xs:sequence>
        </xs:complexType>
    </xs:element>
    <xs:element name="product-functions">
        <xs:complexType>
            <xs:sequence>
                <xs:element name="product-functions-content"
                type="xs:string"/>
                <xs:element name="product-functions-children">
                <xs:complexType>
                <xs:sequence>
                <xs:element name="product-functions-list">
                <xs:complexType>
                <xs:choice maxOccurs="unbounded" minOccurs="0">
                <xs:element name="product-function">
                <xs:complexType>
                <xs:sequence>
                <xs:element name="Particulars" type="xs:string"/>
                </xs:sequence>
                </xs:complexType>
                </xs:element>
```

```
                                    </xs:choice>
                                    </xs:complexType>
                                    </xs:element>
                                    </xs:sequence>
                                    </xs:complexType>
                                    </xs:element>
                                </xs:sequence>
                            </xs:complexType>
                        </xs:element>
                        <xs:element name="user-characteristics" type="xs:string"/>
                        <xs:element name="constraints" type="xs:string"/>
                        <xs:element name="assumptions-and-dependencies"
                            type="xs:string"/>
                        <xs:element name="apportioning-of-requirements"
                            maxOccurs="1" type="xs:string"/>
                    </xs:sequence>
                </xs:complexType>
            </xs:element>
            <xs:element name="overall-description-comments">
                <xs:complexType>
                    <xs:choice maxOccurs="unbounded" minOccurs="0">
                        <xs:element name="overall-description-comment">
                            <xs:complexType>
                                <xs:sequence>
                                  <xs:element name="id" type="xs:int"/>
                                  <xs:element name="date" type="xs:string"/>
                                  <xs:element name="user" type="xs:string"/>
                                  <xs:element name="email" type="xs:string"/>
                                  <xs:element name="comment" type="xs:string"/>
                                  <xs:element name="processed" type="xs:boolean"/>
                                </xs:sequence>
                            </xs:complexType>
                        </xs:element>
                    </xs:choice>
                </xs:complexType>
            </xs:element>
        </xs:sequence>
    </xs:complexType>
</xs:element>
<xs:element name="specific-requirements">
    <xs:complexType>
        <xs:sequence>
            <xs:element name="specific-requirements-content" type="xs:string"/>
            <xs:element name="specific-requirements-children">
                <xs:complexType>
                    <xs:sequence>
                        <xs:element name="external-interfaces">
                            <xs:complexType>
                                <xs:sequence>
                                  <xs:element name="external-interfaces-content"
                                  type="xs:string"/>
                                  <xs:element name="external-interfaces-children">
                                  <xs:complexType>
                                  <xs:sequence>
                                  <xs:element name="user-interfaces">
                                  <xs:complexType>
                                  <xs:sequence>
                                  <xs:element name="user-interfaces-content"
                                  type="xs:string"/>
                                  <xs:element name="user-interfaces-children">
                                  <xs:complexType>
                                  <xs:sequence>
                                  <xs:element name="user-interfaces-list">
                                  <xs:complexType>
                                  <xs:choice maxOccurs="unbounded" minOccurs="0">
                                  <xs:element name="user-interface">
                                  <xs:complexType>
                                  <xs:sequence>
```

```
<xs:element name="name-of-item" type="xs:string"/>
<xs:element name="description-of-purpose"
type="xs:string"/>
<xs:element
name="source-of-input-or-destination-of-output"
type="xs:string"/>
<xs:element
name="valid-range-accuracy-and-or-tolerance"
type="xs:string"/>
<xs:element name="units-of-measure"
type="xs:string"/>
<xs:element name="timing" type="xs:string"/>
<xs:element
name="relationships-to-other-inputs-outputs"
type="xs:string"/>
<xs:element name="screen-formats-organisation"
type="xs:string"/>
<xs:element name="windows-formats-organisation"
type="xs:string"/>
<xs:element name="data-formats" type="xs:string"/>
<xs:element name="command-formats"
type="xs:string"/>
<xs:element name="end-messages" type="xs:string"/>
</xs:sequence>
</xs:complexType>
</xs:element>
</xs:choice>
</xs:complexType>
</xs:element>
</xs:sequence>
</xs:complexType>
</xs:element>
</xs:sequence>
</xs:complexType>
</xs:element>
<xs:element name="hardware-interfaces">
<xs:complexType>
<xs:sequence>
<xs:element name="hardware-interfaces-content"
type="xs:string"/>
<xs:element name="hardware-interfaces-children">
<xs:complexType>
<xs:sequence>
<xs:element name="hardware-interfaces-list">
<xs:complexType>
<xs:choice maxOccurs="unbounded" minOccurs="0">
<xs:element name="hardware-interface">
<xs:complexType>
<xs:sequence>
<xs:element name="name-of-item" type="xs:string"/>
<xs:element name="description-of-purpose"
type="xs:string"/>
<xs:element
name="source-of-input-or-destination-of-output"
type="xs:string"/>
<xs:element
name="valid-range-accuracy-and-or-tolerance"
type="xs:string"/>
<xs:element name="units-of-measure"
type="xs:string"/>
<xs:element name="timing" type="xs:string"/>
<xs:element
name="relationships-to-other-inputs-outputs"
type="xs:string"/>
<xs:element name="screen-formats-organisation"
type="xs:string"/>
<xs:element name="windows-formats-organisation"
type="xs:string"/>
```

```
<xs:element name="data-formats" type="xs:string"/>
<xs:element name="command-formats"
type="xs:string"/>
<xs:element name="end-messages" type="xs:string"/>
</xs:sequence>
</xs:complexType>
</xs:element>
</xs:choice>
</xs:complexType>
</xs:element>
</xs:sequence>
</xs:complexType>
</xs:element>
</xs:sequence>
</xs:complexType>
</xs:element>
<xs:element name="software-interfaces">
<xs:complexType>
<xs:sequence>
<xs:element name="software-interfaces-content"
type="xs:string"/>
<xs:element name="software-interfaces-children">
<xs:complexType>
<xs:sequence>
<xs:element name="software-interfaces-list">
<xs:complexType>
<xs:choice maxOccurs="unbounded" minOccurs="0">
<xs:element name="software-interface">
<xs:complexType>
<xs:sequence>
<xs:element name="name-of-item" type="xs:string"/>
<xs:element name="description-of-purpose"
type="xs:string"/>
<xs:element
name="source-of-input-or-destination-of-output"
type="xs:string"/>
<xs:element
name="valid-range-accuracy-and-or-tolerance"
type="xs:string"/>
<xs:element name="units-of-measure"
type="xs:string"/>
<xs:element name="timing" type="xs:string"/>
<xs:element
name="relationships-to-other-inputs-outputs"
type="xs:string"/>
<xs:element name="screen-formats-organisation"
type="xs:string"/>
<xs:element name="windows-formats-organisation"
type="xs:string"/>
<xs:element name="data-formats" type="xs:string"/>
<xs:element name="command-formats"
type="xs:string"/>
<xs:element name="end-messages" type="xs:string"/>
</xs:sequence>
</xs:complexType>
</xs:element>
</xs:choice>
</xs:complexType>
</xs:element>
</xs:sequence>
</xs:complexType>
</xs:element>
<xs:element name="communication-interfaces">
<xs:complexType>
<xs:sequence>
```

```
                <xs:element
                name="communication-interfaces-content"
                type="xs:string"/>
                <xs:element
                name="communication-interfaces-children">
                <xs:complexType>
                <xs:sequence>
                <xs:element name="communication-interfaces-list">
                <xs:complexType>
                <xs:choice maxOccurs="unbounded" minOccurs="0">
                <xs:element name="communication-interface">
                <xs:complexType>
                <xs:sequence>
                <xs:element name="name-of-item" type="xs:string"/>
                <xs:element name="description-of-purpose"
                type="xs:string"/>
                <xs:element name="source-input-output"
                type="xs:string"/>
                <xs:element name="valid-accuracy-or-tolerance"
                type="xs:string"/>
                <xs:element name="units-of-measure"
                type="xs:string"/>
                <xs:element name="timing" type="xs:string"/>
                <xs:element name="relationships-inputs-outputs"
                type="xs:string"/>
                <xs:element name="screen-formats-organisation"
                type="xs:string"/>
                <xs:element name="windows-formats-organisation"
                type="xs:string"/>
                <xs:element name="data-formats" type="xs:string"/>
                <xs:element name="command-formats"
                type="xs:string"/>
                <xs:element name="end-messages" type="xs:string"/>
                </xs:sequence>
                </xs:complexType>
                </xs:element>
                </xs:choice>
                </xs:complexType>
                </xs:element>
                </xs:sequence>
                </xs:complexType>
                </xs:element>
                </xs:sequence>
                </xs:complexType>
                </xs:element>
                </xs:sequence>
                </xs:complexType>
                </xs:element>
            </xs:sequence>
        </xs:complexType>
    </xs:element>
    <xs:element name="functional-requirements">
        <xs:complexType>
            <xs:sequence>
                <xs:element name="functional-requirements-content"
                type="xs:string"/>
                <xs:element
                name="functional-requirements-children">
                <xs:complexType>
                <xs:sequence>
                <xs:element name="functional-requirements-list">
                <xs:complexType>
                <xs:choice maxOccurs="unbounded" minOccurs="0">
                <xs:element name="functional-requirement">
                <xs:complexType>
                <xs:sequence>
                <xs:element name="safety" type="xs:boolean"/>
                <xs:element name="feature" type="xs:string"/>
```

```
<xs:element name="funcionality" type="xs:string"/>
<xs:element name="description" type="xs:string"/>
<xs:element name="validity-checks-on-the-inputs"
type="xs:string"/>
<xs:element name="exact-sequence-of-operations"
type="xs:string"/>
<xs:element name="resp-to-abn-situations">
<xs:complexType>
<xs:sequence>
<xs:element name="resp-to-abn-situations-content"
type="xs:string"/>
<xs:element name="resp-to-abn-situations-children">
<xs:complexType>
<xs:sequence>
<xs:element name="overflow" type="xs:string"/>
<xs:element name="communication-facilities"
type="xs:string"/>
<xs:element name="error-handling-and-recovery"
type="xs:string"/>
</xs:sequence>
</xs:complexType>
</xs:element>
</xs:sequence>
</xs:complexType>
</xs:element>
<xs:element name="effect-of-parameters"
type="xs:string"/>
<xs:element name="relationship-out-in">
<xs:complexType>
<xs:sequence>
<xs:element name="relationship-out-in-content"
type="xs:string"/>
<xs:element name="relationship-out-in-children">
<xs:complexType>
<xs:sequence>
<xs:element name="input-output-sequences"
type="xs:string"/>
<xs:element name="formulas-for-in-out-conversion"
type="xs:string"/>
</xs:sequence>
</xs:complexType>
</xs:element>
</xs:sequence>
</xs:complexType>
</xs:element>
</xs:sequence>
<xs:attribute name="id" type="xs:string"/>
<xs:attribute name="number" type="xs:int"/>
</xs:complexType>
</xs:element>
</xs:choice>
</xs:complexType>
</xs:element>
</xs:sequence>
</xs:complexType>
</xs:element>
<xs:element
name="functional-requirements-comments">
<xs:complexType>
<xs:choice maxOccurs="unbounded" minOccurs="0">
<xs:element name="functional-requirements-comment">
<xs:complexType>
<xs:sequence>
<xs:element name="id" type="xs:int"/>
<xs:element name="date" type="xs:string"/>
<xs:element name="user" type="xs:string"/>
<xs:element name="email" type="xs:string"/>
<xs:element name="comment" type="xs:string"/>
```

```
                                <xs:element name="processed" type="xs:boolean"/>
                                </xs:sequence>
                                </xs:complexType>
                                </xs:element>
                                </xs:choice>
                                </xs:complexType>
                                </xs:element>
                            </xs:sequence>
                        </xs:complexType>
                    </xs:element>
                    <xs:element name="performance-requirements" type="xs:string"
                        > </xs:element>
                    <xs:element name="design-constraints">
                        <xs:complexType>
                            <xs:sequence>
                              <xs:element name="design-constraints-content"
                              type="xs:string"/>
                              <xs:element name="design-constraints-children">
                              <xs:complexType>
                              <xs:sequence>
                              <xs:element name="standards-compliance"
                              type="xs:string"/>
                              <xs:element name="logical-database-requirements"
                              type="xs:string"> </xs:element>
                              </xs:sequence>
                              </xs:complexType>
                              </xs:element>
                            </xs:sequence>
                        </xs:complexType>
                    </xs:element>
                    <xs:element name="software-system-attributes">
                        <xs:complexType>
                            <xs:sequence>
                              <xs:element
                              name="software-system-attributes-content"
                              type="xs:string"/>
                              <xs:element
                              name="software-system-attributes-children">
                              <xs:complexType>
                              <xs:sequence>
                              <xs:element name="reliability" type="xs:string"/>
                              <xs:element name="availability" type="xs:string"/>
                              <xs:element name="security" type="xs:string"/>
                              <xs:element name="maintainability"
                              type="xs:string"/>
                              <xs:element name="portability" type="xs:string"/>
                              </xs:sequence>
                              </xs:complexType>
                              </xs:element>
                            </xs:sequence>
                        </xs:complexType>
                    </xs:element>
                    <xs:element name="other-requirements" type="xs:string">
                    </xs:element>
                </xs:sequence>
            </xs:complexType>
        </xs:element>
    </xs:sequence>
</xs:complexType>
</xs:element>
<xs:element name="supporting-information">
    <xs:complexType>
        <xs:sequence>
            <xs:element name="supporting-information-content" type="xs:string"/>
            <xs:element name="supporting-information-children">
                <xs:complexType>
                    <xs:sequence>
                        <xs:element name="table-of-contents" type="xs:string"/>
```

```
                                    <xs:element name="index-overview">
                                        <xs:complexType>
                                            <xs:sequence>
                                              <xs:element name="index-list">
                                              <xs:complexType>
                                              <xs:choice maxOccurs="unbounded" minOccurs="0">
                                              <xs:element name="index">
                                              <xs:complexType>
                                              <xs:sequence>
                                              <xs:element name="term" type="xs:string"/>
                                              <xs:element name="description" type="xs:string"/>
                                              </xs:sequence>
                                              </xs:complexType>
                                              </xs:element>
                                              </xs:choice>
                                              </xs:complexType>
                                              </xs:element>
                                            </xs:sequence>
                                        </xs:complexType>
                                    </xs:element>
                                    <xs:element name="appendixes">
                                        <xs:complexType>
                                            <xs:sequence>
                                              <xs:element name="appendix-content"
                                              type="xs:string"/>
                                              <xs:element name="appendix-children">
                                              <xs:complexType>
                                              <xs:sequence>
                                              <xs:element name="appendix-list">
                                              <xs:complexType>
                                              <xs:choice maxOccurs="unbounded" minOccurs="0">
                                              <xs:element name="appendix">
                                              <xs:complexType>
                                              <xs:sequence>
                                              <xs:element name="description" type="xs:string"/>
                                              <xs:element name="document" type="xs:string"/>
                                              </xs:sequence>
                                              </xs:complexType>
                                              </xs:element>
                                              </xs:choice>
                                              </xs:complexType>
                                              </xs:element>
                                              </xs:sequence>
                                              </xs:complexType>
                                              </xs:element>
                                            </xs:sequence>
                                        </xs:complexType>
                                    </xs:element>
                                </xs:sequence>
                            </xs:complexType>
                        </xs:element>
                    </xs:sequence>
                    <xs:attribute name="projectName" type="xs:string"/>
                    <xs:attribute name="fileName" type="xs:string"/>
                    <xs:attribute name="inspectorName" type="xs:string"/>
                    <xs:attribute name="documentType" type="xs:string"/>
                    <xs:attribute name="companyName" type="xs:string"/>
                    <xs:attribute name="authorName" type="xs:string"/>
                    <xs:attribute name="documentVersion" type="xs:string"/>
                    <xs:attribute name="documentState" type="xs:string"/>
                    <xs:attribute name="documentDate" type="xs:dateTime"/>
                    <xs:attribute name="creationDate" type="xs:dateTime"/>
                    <xs:attribute name="startDate" type="xs:dateTime"/>
                </xs:complexType>
            </xs:element>
```

```
    <xs:element name="comment-children">
        <xs:complexType>
            <xs:sequence>
                <xs:element name="comment-list">
                    <xs:complexType>
                        <xs:choice maxOccurs="unbounded" minOccurs="0">
                            <xs:element name="comment">
                                <xs:complexType>
                                    <xs:sequence>
                                        <xs:element name="id" type="xs:int"/>
                                        <xs:element name="date" type="xs:string"/>
                                        <xs:element name="user" type="xs:string"/>
                                        <xs:element name="email" type="xs:string"/>
                                        <xs:element name="comment" type="xs:string"/>
                                        <xs:element name="processed" type="xs:boolean"/>
                                    </xs:sequence>
                                </xs:complexType>
                            </xs:element>
                        </xs:choice>
                    </xs:complexType>
                </xs:element>
            </xs:sequence>
        </xs:complexType>
    </xs:element>
</xs:schema>
```

# 7.2 XML-Struktur für eine Software-Spezifikation

Aus XSD-Schema generiertes leeres Objekt einer Spezifikation in XML-Form.

```
<?xml version="1.0" encoding="UTF-8"?>
<ns2:specification xmlns:ns2="http://org.hbrs.siwob.types/">
    <document-control>
        <approval></approval>
        <change-control>
            <initial-release></initial-release>
            <current-release></current-release>
            <last-review></last-review>
            <next-review></next-review>
            <next-update></next-update>
        </change-control>
        <revision-history>
            <revision-list/>
        </revision-history>
        <distributor>
            <distribution-list/>
        </distributor>
    </document-control>
    <introduction>
        <introduction-content></introduction-content>
        <introduction-children>
            <purpose></purpose>
            <scope></scope>
            <definitions-acronyms-and-abbrevations>
                <definitions-acronyms-and-abbrevations-content></definitions-acronyms-and-abbrevations-content>
                <definitions-acronyms-and-abbrevations-children>
                    <definition-list/>
                    <acronym-list/>
                </definitions-acronyms-and-abbrevations-children>
            </definitions-acronyms-and-abbrevations>
            <references>
                <references-content></references-content>
                <references-children>
                    <references-list/>
                </references-children>
            </references>
            <overview></overview>
        </introduction-children>
        <introduction-comments/>
    </introduction>
    <overall-description>
        <overall-description-content></overall-description-content>
        <overall-description-children>
            <product-perspective>
                <product-perspective-content></product-perspective-content>
                <product-perspective-children>
                    <detailed-methodology></detailed-methodology>
                    <system-interfaces>
                        <system-interfaces-content></system-interfaces-content>
                        <system-interfaces-children>
                            <system-interfaces-list/>
                        </system-interfaces-children>
                    </system-interfaces>
                    <user-interfaces></user-interfaces>
                    <hardware-interfaces>
                        <hardware-interfaces-content></hardware-interfaces-content>
                        <hardware-interfaces-children>
                            <hardware-interfaces-list/>
                        </hardware-interfaces-children>
                    </hardware-interfaces>
                    <software-interfaces>
                        <software-interfaces-content></software-interfaces-content>
                        <software-interface-children>
                            <software-product-list/>
                            <interfaces-list/>
                        </software-interface-children>
                    </software-interfaces>
```

```
                <communication-interfaces></communication-interfaces>
                <memory-constrains></memory-constrains>
                <operations></operations>
                <side-adaption-requirements></side-adaption-requirements>
            </product-perspective-children>
        </product-perspective>
        <product-functions>
            <product-functions-content></product-functions-content>
            <product-functions-children>
                <product-functions-list/>
            </product-functions-children>
        </product-functions>
        <user-characteristics></user-characteristics>
        <constraints></constraints>
        <assumptions-and-dependencies></assumptions-and-dependencies>
        <apportioning-of-requirements></apportioning-of-requirements>
    </overall-description-children>
    <overall-description-comments/>
</overall-description>
<specific-requirements>
    <specific-requirements-content></specific-requirements-content>
    <specific-requirements-children>
        <external-interfaces>
            <external-interfaces-content></external-interfaces-content>
            <external-interfaces-children>
                <user-interfaces>
                    <user-interfaces-content></user-interfaces-content>
                    <user-interfaces-children>
                        <user-interfaces-list/>
                    </user-interfaces-children>
                </user-interfaces>
                <hardware-interfaces>
                    <hardware-interfaces-content></hardware-interfaces-content>
                    <hardware-interfaces-children>
                        <hardware-interfaces-list/>
                    </hardware-interfaces-children>
                </hardware-interfaces>
                <software-interfaces>
                    <software-interfaces-content></software-interfaces-content>
                    <software-interfaces-children>
                        <software-interfaces-list/>
                    </software-interfaces-children>
                </software-interfaces>
                <communication-interfaces>
                    <communication-interfaces-content></communication-interfaces-content>
                    <communication-interfaces-children>
                        <communication-interfaces-list/>
                    </communication-interfaces-children>
                </communication-interfaces>
            </external-interfaces-children>
        </external-interfaces>
        <functional-requirements>
            <functional-requirements-content></functional-requirements-content>
            <functional-requirements-children>
                <functional-requirements-list/>
            </functional-requirements-children>
            <functional-requirements-comments/>
        </functional-requirements>
        <performance-requirements></performance-requirements>
        <design-constraints>
            <design-constraints-content></design-constraints-content>
            <design-constraints-children>
                <standards-compliance></standards-compliance>
                <logical-database-requirements></logical-database-requirements>
            </design-constraints-children>
        </design-constraints>
        <software-system-attributes>
            <software-system-attributes-content></software-system-attributes-content>
```

```
                <software-system-attributes-children>
                    <reliability></reliability>
                    <availability></availability>
                    <security></security>
                    <maintainability></maintainability>
                    <portability></portability>
                </software-system-attributes-children>
            </software-system-attributes>
            <other-requirements></other-requirements>
        </specific-requirements-children>
    </specific-requirements>
    <supporting-information>
        <supporting-information-content></supporting-information-content>
        <supporting-information-children>
            <table-of-contents></table-of-contents>
            <index-overview>
                <index-list/>
            </index-overview>
            <appendixes>
                <appendix-content></appendix-content>
                <appendix-children>
                    <appendix-list/>
                </appendix-children>
            </appendixes>
        </supporting-information-children>
    </supporting-information>
</ns2:specification>
```

## 7.3 Tabellarische Darstellung der Strukturmerkmale

| Kurzbeschreibung: | Merkmal | XML-Element |
|---|---|---|
| **Einleitung** | g | introduction |
| **Zweck des Dokuments** | g | scope |
| Eintrag | i | scope-entry |
| **Umfang des Dokuments** | g | scope |
| Eintrag | i | scope-entry |
| Name des Produkts | i | product-name |
| Kernfunktionalitäten | i | core-functions |
| Ziele und Nutzen | i | use-and-goals |
| **Begriffsdefinitionen** | g | definitions |
| Eintrag | g, w | definitions-entry |
| Begriff | i | term |
| Begriffsdefinition | i | term-definition |
| **Abkürzungen** | g | acronyms |
| Eintrag | g, w | acronyms-entry |
| Abkürzung | i | acronym |
| Abkürzungsdefinition | i | acronym-definition |
| **Referenzen** | g | references |
| Eintrag | g, w | references-entry |
| Referenz Titel | i | references-title |
| Referenz Datum | i | references-date |
| Veröffentlichende Organisation | i | publishing-organization |
| Quelle des Dokuments | i | source |
| **Überblick** | g | overview |
| Eintrag | i | overview-entry |
| | | |
| **Allgemeine Beschreibungen** | | overall-description |
| **Produktperspektive** | g | product-perspective |
| **Systemschnittstelle** | g | system-interfaces |
| Eintrag | g, w | system-interfaces-entry |
| Name | i | system-interface-name |
| Beschreibung | i | system-interface-description |
| **Nutzerschnittstelle** | g | user-interfaces |
| Eintrag | g, w | user-interfaces-entry |
| Name | i | user-interface-name |
| Konfiguration | i | user-interface-configuration |
| Optimierungs Aspekte | i | user-interface-optaspect |
| **Hardwareschnittstelle** | g | hardware-interfaces |
| Eintrag | g, w | hardware-interfaces-entry |
| Name | i | hardware-interface-name |
| Beschreibung | i | hardware-interface-description |
| **Softwareschnittstelle** | g | software-interfaces |
| Eintrag | g, w | software-interfaces-entry |
| Name | i | software-interface-name |
| Abkürzung | i | software-interface-acronym |
| Spezifikation | i | software-interface-specification |
| Version | i | software-interface-version |
| Quelle | i | software-interfacesource |
| Beschreibung | i | software-interfacedescription |
| **Kommunikationsschnittstelle** | g | comm-interfaces |
| Eintrag | g, w | comm-interfaces-entry |
| Name | i | comm-interface-name |
| Beschreibung | i | comm-interface-description |

| | | |
|---|---|---|
| **Speichernutzung** | g | memory |
| Eintrag | i, w | memory-entry |
| **Softwareeinsatz** | g | operations |
| Eintrag | i, w | operations-entry |
| **Installation und Initialisierung** | g | side-adaptation |
| Eintrag | i, w | side-adaptation-entry |
| **Produktfunktionen** | g | product-functions |
| Eintrag | i, w | product-functions-entry |
| **Nutzercharakteristika** | g | user-characteristics |
| Eintrag | i, w | user-characteristics-entry |
| **weitere Beschränkungen** | g | constraints |
| Eintrag | i, w | constraints-entry |
| **Annahmen und Abhängigkeiten** | g | assumptions-dependencies |
| Eintrag | i, w | assumptions-dependencies-entry |
| **Zurückgestellte Anforderungen** | g | apportioned-requirements |
| Eintrag | i, w | apportioned-requirements-entry |
| | | |
| **Detaillierte Anforderungen** | g | specifics-reqruirements |
| **Externe Schnittstellen** | g | external-interfaces |
| **Nutzerschnittstellen** | g | det-user-interfaces |
| Eintrag | g, w | det-user-interface-entry |
| Name der Schnittstelle | i | name-of-item |
| Beschreibung des Zwecks | i | description-of-purpose |
| Quelle oder Ziel des Signals/Antwort | i | source-of-input-ordestination-of-output |
| Gültigkeitsbereich, Genauigkeit, Toleranz | i | valid-range-accuracy-and-or-tolerance |
| Maßeinheiten | i | timing |
| Abstimmung | i | relationships-to-other-inputs-outputs |
| Beziehung zu anderen Ein-/Ausgangssignalen | i | screen-formats-organisation |
| Bildschirmformate | i | window-formats-organisation |
| Bildbereichformate | i | screen-formats-organisation |
| Datenformate | i | window-formats-organisation |
| Befehlsformate | i | data-formats |
| Beendigungsnachricht | i | end-message |
| eindeutiger Identifizierer | i | ID |
| **Hardwareschnittstellen** | g | det-hardware-interfaces |
| Eintrag | g, w | det-hardware-interface-entry |
| Name der Schnittstelle | i | name-of-item |
| Beschreibung des Zwecks | i | description-of-purpose |
| Quelle oder Ziel des Signals/Antwort | i | source-of-input-ordestination-of-output |
| Gültigkeitsbereich, Genauigkeit, Toleranz | i | valid-range-accuracy-and-or-tolerance |
| Maßeinheiten | i | timing |
| Abstimmung | i | relationships-to-other-inputs-outputs |
| Beziehung zu anderen Ein-/Ausgangssignalen | i | screen-formats-organisation |
| Bildschirmformate | i | window-formats-organisation |
| Bildbereichformate | i | screen-formats-organisation |
| Datenformate | i | window-formats-organisation |
| Befehlsformate | i | data-formats |
| Beendigungsnachricht | i | end-message |
| eindeutiger Identifizierer | | ID |
| **Softwareschnittstellen** | g | det-software-interfaces |
| Eintrag | g, w | det-software-interface-entry |
| Name der Schnittstelle | i | name-of-item |
| Beschreibung des Zwecks | i | description-of-purpose |
| Quelle oder Ziel des Signals/Antwort | i | source-of-input-ordestination-of-output |

| | | |
|---|---|---|
| Quelle oder Ziel des Signals/Antwort | i | source-of-input-ordestination-of-output |
| Gültigkeitsbereich, Genauigkeit, Toleranz | i | valid-range-accuracy-and-or-tolerance |
| Maßeinheiten | i | timing |
| Abstimmung | i | relationships-to-other-inputs-outputs |
| Beziehung zu anderen Ein-/Ausgangssignalen | i | screen-formats-organisation |
| Bildschirmformate | i | window-formats-organisation |
| Bildbereichformate | i | screen-formats-organisation |
| Datenformate | i | window-formats-organisation |
| Befehlsformate | i | data-formats |
| Beendigungsnachricht | i | end-message |
| eindeutiger Identifizierer | | ID |
| **Kommunikationsschnittstellen** | g | det-comm-interfaces |
| Eintrag | g, w | det-comm-interface-entry |
| Name der Schnittstelle | i | name-of-item |
| Beschreibung des Zwecks | i | description-of-purpose |
| Quelle oder Ziel des Signals/Antwort | i | source-of-input-ordestination-of-output |
| Gültigkeitsbereich, Genauigkeit, Toleranz | i | valid-range-accuracy-and-or-tolerance |
| Maßeinheiten | i | timing |
| Abstimmung | i | relationships-to-other-inputs-outputs |
| Beziehung zu anderen Ein-/Ausgangssignalen | i | screen-formats-organisation |
| Bildschirmformate | i | window-formats-organisation |
| Bildbereichformate | i | screen-formats-organisation |
| Datenformate | i | window-formats-organisation |
| Befehlsformate | i | data-formats |
| Beendigungsnachricht | i | end-message |
| eindeutiger Identifizierer | | ID |
| **Hauptmerkmale des System** | g | system-features |
| Eintrag | g, w | system-features-entry |
| Name | i | feature-name |
| Beschreibung | i | introduction-purpose |
| Stimulus/Antwort Sequenz | i | stimulus-response-sequence |
| **Assoziierte funktionale Anforderungen** | g | assoc-functional |
| Eintrag | g, w | assoc-functional-entry |
| Name der funktionalen Anforderungen | i | functional-requirement-name |
| Beschreibung | i | description |
| Validitätsprüfung von Inputs | i | validity-checks-on-inputs |
| Ablaufsequenz der Operationen | i | exact-sequence-of-operations |
| Fehlerbehandlungen | i | responses-to-abnormal-situations |
| Einfluss von Parametern | i | effect-of-parameters |
| In-/Output Verhältnis | i | relationships-to-other-inputs-outputs |
| eindeutiger Identifizierer | i | ID |
| **Anforderungen an die Performanz** | g | performance-requirements |
| Eintrag | i, w | performance-requirements-entry |
| **Anforderungen der Datenbank** | g | logical-database |
| Eintrag | i, w | logical-database-entry |
| **Einschränkungen des Entwurfs** | g | design-constraints |
| **Standards** | g | standards-compliance |
| Eintrag | i, w | standards-compliance-entry |
| **weitere Einschränkungen** | g | other-constraints |
| Eintrag | i, w | other-constraints-entry |
| **Softwaresystem Attribute** | g | software-system-attributes |
| **Zuverlässigkeit** | g | reliability |
| Eintrag | i | reliability-entry |

| | | |
|---|---|---|
| **Zuverlässigkeit** | g | reliability |
| Eintrag | i | reliability-entry |
| **Verfügbarkeit** | g | availability |
| Eintrag | i | availability-entry |
| **Sicherheit** | g | security |
| Eintrag | i | security-entry |
| **Wartbarkeit** | g | maintainability |
| Eintrag | i | maintainability-entry |
| **Portabilität** | g | portability |
| Eintrag | i | portability-entry |
| **weitere Anforderungen** | g | other-requirements |
| Eintrag | i | other-requirements-entry |
| | | |
| **Appendix** | g | appendix |
| Eintrag | i | appendix-entry |
| | | |
| Varianten A-G zur Darstellung von Spezifikationen nach Gegenstandstyp | | |
| Die einzelnen Elemente der Funktionalen Anforderungen weichen nicht von den bereits definierten ab | | |
| **Variante A:** | | |
| Funktionale Anforderungen | g | |
| Modus | g | |
| Eintrag | g, w | |
| Funktionale Anforderungen | g | |
| Eintrag | g, w | |
| | | |
| **Variante B:** | | |
| Funktionale Anforderungen | g | |
| Modus | g | |
| Eintrag | g, w | |
| Externe Schnittstellen | g, w | |
| (siehe Auflistung der ext. Schnittstellen) | g | |
| Funktionale Anforderungen | g | |
| Eintrag | g, w | |
| | | |
| **Variante C:** | | |
| Funktionale Anforderungen | g | |
| User-Class | g | |
| Eintrag | g, w | |
| Funktionale Anforderungen | g | |
| Eintrag | g, w | |
| | | |
| **Variante D:** | | |
| Klassen/Objekte | g | |
| Klasse/Objekt | g | |
| Eintrag | g | |
| Attribute | g | |
| Eintrag | g, w | |
| Funktionale Anforderungen | g | |
| Eintrag | g, w | |
| | | |
| **Variante E:** | | |
| Siehe: Hauptmerkmale | | |
| | | |

| | | |
|---|---|---|
| **Variante F:** | | |
| Funktionale Anforderungen | g | |
| Stimulus | g | |
| Eintrag | g, w | |
| Funktionale Anforderungen | g | |
| Eintrag | g, w | |
| | | |
| **Variante G:** | | |
| Funktionale Anforderungen | g | |
| Antwort | g | |
| Eintrag | g, w | |
| Funktionale Anforderungen | g | |
| Eintrag | g, w | |

# 8 Quellenverzeichnis

Bath, G., & McKay, J. (2011). **Praxiswissen Softwaretest - Test Analyst und Technical Test Analyst: Aus- und Weiterbildung zum Certified Tester - Advanced Level nach ISTQB-Standard.** Heidelberg: dpunkt-Verl.

Bien, A. (2009). **Real world Java EE Patterns: rethinking best practices.** London: press.adam-bien.com.

Bommer, C., Spindler, M., & Barr, V. (2008). **Softwarewartung: Grundlagen, Management und Wartungstechniken.** Heidelberg: Dpunkt-Verl.

Bootstrap (Hrsg.). (2014a). About: Learn about the project's history, meet the maintaining teams, and find out how to use the Bootstrap brand. [Web page] Retrieved from http://getbootstrap.com/about/

Bootstrap (Hrsg.). (2014b). Bootstrap is the most popular HTML, CSS, and JS framework for developing responsive, mobile first projects on the web. [Web page] Retrieved from http://getbootstrap.com

Bootstrap (Hrsg.). (2014c). CSS: Global CSS settings, fundamental HTML elements styled and enhanced with extensible classes, and an advanced grid system. [Web page] Retrieved from http://getbootstrap.com/css/

Bootstrap (Hrsg.). (2014d). Getting started: An overview of Bootstrap, how to download and use, basic templates and examples, and more. [Web page] Retrieved from http://getbootstrap.com/getting-started/

Börcsök, J. (2011). **Funktionale Sicherheit: Grundzüge sicherheitstechnischer Systeme.** Berlin: VDE Verl.

BSI. (2010). Functional safety of electrical/electronic/programmable electronic safety-related systems - Part 3: Software requirements. In **IEC 61508:2010.** Bonn: BSI.

Bundesamt für Sicherheit in der Informationstechnik (Hrsg.). (2009). **SOA-Security-Kompendium Sicherheit in Service-orientierten Architekturen**. Bundesamt für Sicherheit in der Informationstechnik. Retrieved from https://www.bsi.bund.de/cae/servlet/contentblob/486838/publicationFile/30662/SOA-Security-Kompendium_pdf.pdf

Burbiel, H. (2007). **SOA und Webservices in der Praxis.** München: Franzis Verlag.

Chappell, D. A. (2004). **Enterprise service bus.** Sebastopol, Calif.: O'Reilly.

Collins-Sussman, B., Fitzpatrick, B. W., Lichtenberg, K., & Pilato, C. M. (2007). **Versionskontrolle mit Subversion - Software-Projekte intelligent koordinieren** (Vol. 2). Köln: O'Reilly Verlag.

Dahm, M. (2006). **Grundlagen der Mensch-Computer-Interaktion.** München: Pearson Studium.

Deutsches Institut für Normung. (2008a). Ergonomie der Mensch-System-Interaktion – Teil 110: Grundsätze der Dialoggestaltung (ISO 9241-110:2006). In **DIN EN ISO 9241-110.** Berlin: Beuth Verlag.

Deutsches Institut für Normung. (2008b). Sicherheit von Maschinen - Sicherheitsbezogene Teile von Steuerungen - Teil 1: Allgemeine Gestaltungsleitsätze (ISO 13849-1:2006). In **DIN EN ISO 13849-1:2008-12.** Berlin: Beuth Verlag.

Dussa-Zieger. (2011, June). ISO/IEC 29119 - Die neue Testnorm. **SQ-Magazin**, (19), 6-7. Retrieved from https://www.asqf.de/sq-magazin-fuer-gaeste/items/ausgabe-19-juni-2011.html.

Eastlake 3rd, & Jones. (2001). US Secure Hash Algorithm 1 (SHA1). **RFC 3174 - US Secure Hash Algorithm 1 (SHA1).** Retrieved from tools.ietf.org: http://tools.ietf.org/html/rfc3174.html

Ebert, C. (2012). **Systematisches Requirements Engineering Anforderungen ermitteln, spezifizieren, analysieren und verwalten**. dpunkt.verlag. Retrieved from http://public.eblib.com/choice/publicfullrecord.aspx?p=995324

Eckert, C. (2013). **IT-Sicherheit Konzepte - Verfahren - Protokolle** (8 ed.). München: Oldenbourg.

Engels, G., Goedicke, M., Goltz, U., Rausch, A., & Reussner, R. (2009). Design for Future – Legacy-Probleme von morgen vermeidbar? **Informatik-Spektrum**, **32**(5), 393-397. Retrieved from http://dx.doi.org/10.1007/s00287-009-0356-3

Finger, P., & Zeppenfeld, K. (2009). **SOA und WebServices.** Berlin; Heidelberg: Springer.

Flanagan, C., & Qadeer, S. (2002). Predicate Abstraction for Software Verification. **SIGPLAN Not.**, **37**(1), 191-202. Retrieved from ACM: http://doi.acm.org/10.1145/565816.503291

Fraser, N. (2012). Google Diff Patch Match - Diff, Match and Patch libraries for Plain Text. [Web page] Retrieved from https://code.google.com/p/google-diff-match-patch/

Free Software Foundation. (2013). GNU Diffutils. [Web page] Retrieved from www.gnu.org: http://www.gnu.org/software/diffutils/

Free Software Foundation Inc. (Hrsg.). (2014). Various Licenses and Comments about Them. **Various Licenses and Comments about Them** [Web page]. Retrieved from http://www.gnu.org/licenses/license-list.en.html

Frick, D., Servaes, I., Mehrtens, M., Söhnchen, P. G., Stegemerten, B., Treibert, R., . . . Mülder, W. (2009). **Masterkurs Wirtschaftsinformatik: Kompakt, praxisnah, verständlich-12 Lern-und Arbeitsmodule.** Wiesbaden: Springer.

Frühauf, K., Ludewig, J., & Sandmayr, H. (2007). **Software-Prüfung: eine Anleitung zum Test und zur Inspektion** (Vol. 6). Zürich: Vdf, Hochschulverl. an der ETH Zürich.

Gürsoy, H. (2011). Über den Status quo verteilter Versionsverwaltungssysteme. [Web page] Retrieved from www.heise.de: http://www.heise.de/developer/artikel/Ueber-den-Status-quo-verteilter-Versionsverwaltungssysteme-1285649.html

Haenel, V., & Plenz, J. (2011). **Git Verteilte Versionsverwaltung für Code und Dokumente.** München: Open Source Press.

Hammerschall, U. (2005). **Verteilte Systeme und Anwendungen: Architekturkonzepte, Standards und Middleware-Technologien.** München; Boston [u.a.]: Pearson Studium.

Heinecke, A. M. (2004). **Mensch-Computer-Interaktion.** Leipzig: Carl Hanser Verlag.

Helmbrecht, U. (2009). **BSI Technische Richtlinie 03125: Vertrauenswürdige elektronische Langzeitspeicherung** (1 ed.). Bonn: BSI.

Hering, N. (2005). **Vom Denken und von Denkmaschinen: über die Grenzen des Verstehens zwischen Gehirn und Prozessor.** Bad Honnef: Hippocampus-Verl.

Hoffmann, D. W. (2013). **Software-Qualität**. Berlin: Springer Vieweg.

IEEE Computer Society. (1998). IEEE Recommended Practice for Software Requirements Specifications. **IEEE Std 830-1998**, 1-40. Retrieved from http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=720574&isnumber=15571.

Liam R. E. Quin. (2014). XML Essentials. **XML Essentials** [Web page]. Retrieved from http://www.w3.org/standards/xml/core

Liebhart, D. (2007). **SOA goes real: service-orientierte Architekturen erfolgreich planen und einführen.** München [u.a.]: Hanser.

Liggesmeyer, P. (2009). **Software-Qualität: Testen, Analysieren und Verifizieren von Software** (Vol. 2). Heidelberg: Spektrum, Akad. Verl.

Löw, P., Pabst, R., & Petry, E. (2010). **Funktionale Sicherheit in der Praxis: Anwendung von DIN EN 61508 und ISO/DIS 26262 bei der Entwicklung von Serienprodukten** (1 ed.). Heidelberg: Dpunkt.Verlag.

Ludewig, J., & Lichter, H. (2010). **Software Engineering Grundlagen, Menschen, Prozesse, Techniken.** Heidelberg: dpunkt-Verl.

Mac App Store - Pages. (2014). Mac App Store - Pages. [Web page] Retrieved from https://itunes.apple.com/us/app/pages/id409201541?mt=12&ls=1

Marinschek, & Kurz. (2010). **JavaServer Faces 2.0: Grundlagen und erweiterte Konzepte.** Heidelberg: dpunkt-Verl.

Martin, R. C. (2003). Agile Software Development: Principles, Patterns, and Practices. Prentice Hall PTR. Retrieved from http://dl.acm.org/citation.cfm?id=515230.

Mike Somekh, Mark Foster, Rastislav Kanocz. (2009). Glassfish Enterprise Service Bus (ESB) High Availability and Clustering. Retrieved from http://www.open-esb.net/files/OpenESB Documents/Papers/glassfish_esb_ha_wp.pdf

MuleSoft Inc. (Hrsg.). (2014a). Service Orchestration: Making SOA Work. [Web page] MuleSoft. Retrieved from https://www.mulesoft.com/resources/esb/service-orchestration-and-soa

MuleSoft Inc. (Hrsg.). (2014b). Service Orchestration and SOA. [Web page] MuleSoft. Retrieved from https://www.mulesoft.com/resources/esb/service-orchestration-and-soa

Myers, E. W. (1986). AnO (ND) difference algorithm and its variations. Algorithmica, 1(1-4), 251-266.

Nadkarni, Prakash. (1999). The EAV/CR Model of Data Representation. [Web page] Retrieved from http://ycmi.med.yale.edu/nadkarni/EAV_CR_frame.htm

Naumenko, D. (2013). Java Diff Utils - The DiffUtils library for computing diffs, applying patches, generationg side-by-side view in Java. [Web page] Retrieved from https://code.google.com/p/java-diff-utils/

Nielsen, J. (1993). Usability engineering. Boston: Academic Press.

Nilo Mitra. (2007). SOAP Version 1.2 Part 0: Primer (Second Edition). [Web page] Retrieved from http://www.w3.org/TR/soap12-part0/

Office Home and Business 2013. (2013). Office Home and Business 2013. [Web page] Retrieved from https://products.office.com/de-DE/home-and-business

Opensource.org (Hrsg.). (2014). About Open Source Licenses. 2014 [Web page]. Retrieved from http://opensource.org/licenses

Operating System Marketshare. (2014). Operating system marketshare. [Web page] Retrieved from http://www.netmarketshare.com/operating-system-market-sha- re.aspx?qprid=8&qpcustomd=0

Papazoglou, M. (2008). Web services: principles and technology. Harlow, England; New York: Pearson/Prentice Hall.

Passing, M., & Dressler, F. (2006). Experimental Performance Evaluation of Cryptographic Algorithms on Sensor Nodes. IEEE Xplore, 882-887. Retrieved from IEEE.

Petmecky, T. (2011). RIA - Rich Internet Applications und ihre Einsatzmöglichkeiten. Retrieved from http://www.onm.de/local/media/downloads/whitepapers/lesestoff_01_rich_internet_applications.pdf.

Pohl, K. (2008). Requirements Engineering: Grundlagen, Prinzipien, Techniken. Heidelberg: Dpunkt-Verl.

Pressman, R. S. (2005). Software Engineering: A Practitioner's Approach (6 ed.). New York: McGraw-Hill Higher Education.

Rahm, E. (1994). Mehrrechner-Datenbanksysteme: Grundlagen der verteilten und parallelen Datenbankverarbeitung. Leipzig, München: Erhard Rahm.

Rivest, R. (1992). The MD5 Message-Digest Algorithm. RFC 1321 - the MD5 Message-Digest Algorithm. Retrieved from tools.ietf.org: http://tools.ietf.org/html/rfc1321

Roberto Chinnic, Jean-Jacques Moreau, A. R. S. W. (2007). Web Services Description Language (WSDL) Version 2.0 Part 1: Core Language. [Web page] Retrieved from http://www.w3.org/TR/wsdl20/

Robertson, S., & Robertson, J. (2006). Mastering the requirements process. Upper Saddle River, NJ: Addison-Wesley.

Roe, D. (2013). Microsoft Office Is Still the Productivity Suite Leader. [Web page] Retrieved from http://www.cmswire.com/cms/document-management/microsoft-of-fice-is-still-the-productivity-suite-leader-022908.php

Ron Ten-Hove, Peter Walker. (2005). JSR-000208 Java Business Integration Evaluation 1.0 Final Release. Sun Microsystems, Inc. Retrieved from http://download.oracle.com/otndocs/jcp/jbi-1.0-fr-eval-oth-JSpec/

Röder, Franke, Müller, & Przybylski. (2009). Ein Kriterienkatalog zur Bewertung von Anforderungsspezifikationen. Softwaretechnik-Trends, 29(4).

Rupp, C. (2009). Requirements-Engineering und -Management professionelle, iterative Anforderungsanalyse für die Praxis. München; Wien: Hanser.

Sayegh, A. (1997). Corba : Standard, Spezifikationen, Entwicklung. Cambridge: O'Reilly.

Schmidt, R. F. (2013). Software Engineering: Architecture-driven Software Development. Elsevier.

Scholz, P. (2005). Softwareentwicklung eingebetteter Systeme Grundlagen, Modellierung, Qualitätssicherung. Berlin: Springer Verlag.

Seffah, Donyaee, Kline, & Padda. (2006). Usability measurement and metrics: A consolidated model. Software Quality Journal, 14(2), 159-178.

SELFHTML E.V. (Hrsg.). (2014). Einführung in JavaScript und DOM. [Web page] Retrieved from http://de.selfhtml.org/javascript/intro.htm

Shneiderman, B., & Plaisant, C. (2004). Designing The User Interface: Strategies For Effective Human-Computer Interaction. Boston: Pearson/Addison Wesley.

Sneed, H. M., Baumgartner, M., & Seidl, R. (2012). Der Systemtest von den Anforderungen zum Qualitätsnachweis. München: Hanser Verlag.

Stöcker, C. (2013). Mapping the Internet: A Hacker's Secret Internet Census. [Web page] Hamburg: Spiegel Online. Retrieved from http://www.spiegel.de/international/world/hacker-measures-the-internet-illegally-with-carna-botnet-a-890413.html

The Apache Software Foundation (Hrsg.). (2014). Commons Virtual File System. [Web page] Retrieved from http://commons.apache.org/proper/commons-vfs/index.html

Urbanek, M. (2010). JavaServer Faces - JSF verstehen und praktisch einsetzen. Herdecke; Witten: W3L-Verl.

Van den Broeck, G. (2011). Daisydiff - A Java library to compare HTML files. [Web page] Retrieved from https://code.google.com/p/daisydiff/

Vigenschow, U. (2010). Testen von Software und Embedded Systems: professionelles Vorgehen mit modellbasierten und objektorientierten Ansätzen (Vol. 2). Heidelberg: dpunkt-Verlag.

W3Schools (Hrsg.). (2014a). HTML Responsive Web Design. [Web page] Retrieved from http://www.w3schools.com/html/html_responsive.asp

W3Schools (Hrsg.). (2014b). XML Validator. [Web page] Retrieved from http://www.w3schools.com/xml/xml_validator.asp

Weil, D. (2011). Java EE 6 - Enterprise-Anwendungsentwicklung leicht gemacht. Frankfurt am Main: entwickler press.

Why Apache OpenOffice. (2014). Why Apache OpenOffice. [Web page] Retrieved from http://www.openoffice.org/why/index.html

WordPerfect Office Suite. (2014). WordPerfect Office Suite. [Web page] Retrieved from http://www.wordperfect.com/us/pages/12100162.html?pgid=12100162

Woywode, Mädche, Wallach, & Plach. (2012). Gebrauchstauglichkeit von Anwendungssoftware als Wettbewerbsfaktor für kleine und mittlere Unternehmen (KMU): Abschlussbericht.

Wratil, P., & Kieviet, M. (2010). Sicherheitstechnik für Komponenten und Systeme (2 ed.). Berlin / Offenbach: VDE Verlag GmbH.