Gottfried Wilhelm
Leibniz Universität Hannover
Fakultät für Elektrotechnik und Informatik
Institut für Praktische Informatik
Fachgebiet Software Engineering

Entwicklung eines Tools zur Unterstützung der Kundenkommunikation nach Abschluss von studentischen Softwareprojekten

Bachelorarbeit

im Studiengang Informatik

von

Cedric Wellhausen

Prüfer: Prof. Dr. rer. nat. Kurt Schneider
Zweitprüfer: Dr. rer. nat. Jil Klünder
Betreuer: Jakob Droste, Lukas Nagel

Hannover, 14.08.2023

Erklärung der Selbstständigkeit

Hiermit versichere ich, dass ich die vorliegende Bachelorarbeit selbständig und ohne fremde Hilfe verfasst und keine anderen als die in der Arbeit angegebenen Quellen und Hilfsmittel verwendet habe. Die Arbeit hat in gleicher oder ähnlicher Form noch keinem anderen Prüfungsamt vorgelegen.

Hannover, den 14.08.2023

Cedric Wellhausen

Zusammenfassung

Für das Softwareprojekt des Fachgebiets Software Engineering müssen die Projektergebnisse an die Kunden zuverlässig zugestellt werden. Der Arbeitsablauf besteht dabei für jedes Projekt aus mindestens einem Herunter– und Hochladevorgang, damit die Kunden auf die Projektergebnisse ihrer Teams leicht Zugriff erhalten können. Die händische Durchführung dieses Arbeitsablaufes ist langwierig, da das manuelle Heraussuchen und Übertragen der Projektdateien das Bedienen von drei Benutzeroberflächen erfordert. In dieser Arbeit wurden der beschriebene Arbeitsablauf und weitere Nebenläufigkeiten zu einem automatischen Prozess remodelliert. Es wurden dazu gängige Praktiken des Software– und Requirements Engineering eingesetzt, darunter beispielsweise die Anforderungserhebung durch Interviews. Ziel des eingesetzten Requirements Engineering Prozesses war es, die Auftraggeber bestmöglich in die Entwicklung der Software einzubeziehen. Speziell angefertigte Konzeptpapiere spielten dabei eine wichtige Rolle für die Interpretation der Anforderungen und die frühzeitige Identifizierung von Missverständnissen. Mock–Ups und Prototypen wurden entwickelt, um die verhandelten und dokumentierten Anforderungen zu validieren. Die Implementierung der Software basierte auf den Ergebnissen des Requirements Engineering Prozesses, um die Wünsche der Auftraggeber zu berücksichtigen. Qualitätssichernde Maßnahmen, wie Usabilitytests, wurden durchgeführt, um Mängel in der Bedienbarkeit zu beheben. Am Ende der Arbeit hat sich gezeigt, dass die entwickelte Software den Anforderungen des Fachgebiets Software Engineering genügt und eine vielversprechende Alternative zum manuellen Arbeitsablauf darstellt.

Abstract

For the software project of the Software Engineering department, the project results must be reliably delivered to the customers. The workflow for each project consists of at least one download and upload process, so that customers can easily access their team's project results. The manual execution of this workflow is time-consuming, as it requires manually searching for and transferring project files, involving the operation of three user interfaces. In this work, the described workflow and other intricacies were remodeled into an automated process. Common practices of Software and Requirements Engineering were employed, including requirement gathering through interviews. The goal of the applied Requirements Engineering process was to involve the stakeholders as effectively as possible in the software development. Specially crafted concept papers played an important role in interpreting the requirements and identifying misunderstandings early on. Mock-ups and prototypes were developed to validate the negotiated and documented requirements. The implementation of the software was based on the results of the Requirements Engineering process to accommodate the stakeholders' desires. Quality assurance measures, such as usability tests, were conducted to address usability issues. At the end of the work, it was demonstrated that the developed software meets the requirements of the Software Engineering field and represents a promising alternative to the manual workflow.

Inhaltsverzeichnis

Kapitel 1

Einleitung

1.1 Motivation

In der heutigen Welt wird uns vieles erleichtert oder abgenommen. Dies steht in direktem Zusammenhang mit der weit verbreiteten Aufgabenautomatisierung [8]. Software hat nicht unerheblich zu dieser Entwicklung beigetragen und tut dies weiterhin. Dies macht Sinn, wenn man bedenkt, dass Software– und Computersysteme ihren Ursprung im Wunsch zur Vereinfachung und Automatisierung haben [32]. Grund dafür war ein Bestreben nach Verlässlichkeit, Einfachheit und Effizienz in der Durchführung von mathematischen Berechnungen [32]. Automatisierung durchdringt heute das alltägliche Leben, aber ebenso geschäftlichen Erfolg [5, 8, 23]. Auch für den Hochschulbereich birgt Digitalisierung ein großes Potential [5]. Die durch Software mögliche Aufgabenautomatisierung ist im Lehrbetrieb ebenso anwendbar. Insbesondere, wenn es sich um lange oder repetitive Aufgaben handelt.

1.2 Problemstellung

Im Informatikstudium an der Gottfried Wilhelm Leibniz Universität Hannover ist es ein zentraler Bestandteil, am Softwareprojekt teilzunehmen [15]. Diese Lehrveranstaltung wird zu jedem Wintersemester vom Fachgebiet Software Engineering angeboten [1]. Innerhalb des Projektes wird den Studierenden ein erster, umfassender Eindruck von der Arbeit als Softwareentwickler in einem Team geboten. Dieses praxisorientierte Modul ermöglicht eine gemeinsame Arbeit an jeweils einer Software, die für – in der Regel – echte Kunden entwickelt wird. Dabei können die Studierenden ihr bereits im Studium erlerntes Wissen aus dem Bereich Software Engineering einsetzen und ihre Fähigkeiten unter Beweis stellen. Was dieses Modul besonders wertvoll macht, ist die Tatsache, dass die am Ende vorliegenden Projektergebnisse der Teams von den Kunden in tatsächlicher Weise eingesetzt werden können.

Von hoher Relevanz ist daher, dass die Ergebnisse an die Kunden möglichst effizient, durch die Leitung des Softwareprojektes, übergeben werden.

Aus Gesprächen mit der Leitung des Softwareprojektes ging hervor, dass es die derzeitige Praxis ist, die Projektergebnisse eines jeden Teams einzeln herunterzuladen, um diese dann über E–Mail an die Kunden zu versenden. Bis eine E–Mail versendet werden kann, müssen mehrere, manuelle Schritte durchlaufen werden. Diese Arbeit ist, laut der Leitung des Softwareprojektes, ein langwieriger Prozess und leicht fehleranfällig. Es kann beispielsweise dazu kommen, dass ein Kunde durch versehentliche Unachtsamkeit, die Links zu den Ergebnissen eines anderen Kunden mitgeteilt bekommt. Dies kann das Vertrauen in das Fachgebiet Software Engineering vermindern. Des Weiteren sorgen die angesprochenen manuellen Herunter– und Hochladevorgänge sowie das Umschreiben der E–Mails für Zeitverluste.

1.3 Lösungsansatz

In dieser Bachelorarbeit wird daher eine Software nach gängigen Software Engineering Prinzipien, wie der agilen Softwareentwicklung und dem Requirements Engineering, entwickelt. Dabei werden die zeitintensiven Zwischenschritte durch Automatisierung vereinfacht und vereinheitlicht. Insbesondere beinhaltet dies die Automatisierung der Herunter– und Hochladevorgänge, die Auswahl der Branches mit den Projektergebnissen, die Erstellung von Links sowie das schlussendliche Versenden der E–Mails und ihre Generierung. Durch diesen Grad der Automatisierung kann der Zeitverlust stark eingegrenzt werden und eliminiert viele Möglichkeiten zur menschlichen Erzeugung von Fehlern.

1.4 Struktur der Arbeit

Zunächst wird das grundlegende Wissen aufgearbeitet, das zum allgemeinen Verständnis der Arbeit benötigt wird. Dabei wird außerdem auf entsprechend verwandte Arbeiten und Themengebiete eingegangen, die eine Einordnung in den wissenschaftlichen und technologischen Kontext bieten. Im Anschluss wird der Entwicklungsprozess der Software beschrieben, der dieser Arbeit zugrunde liegt und dabei die eingesetzten Techniken des Software Engineering erläutert. Darauf folgend wird die Implementierung der Software beschrieben. Dies umfasst implementierungsspezifische Details, wie die Struktur der Software, getroffene Entscheidungen und Alternativen. Danach wird ein Einblick in die Qualitätssicherung geboten. Schlussendlich wird ein Fazit gezogen und ein Ausblick in die Zukunft gegeben.

Kapitel 2

Grundlagen und verwandte Arbeiten

Der Begriff Prozessautomatisierung bezeichnet die voll– oder teilautomatische Durchführung von Arbeitsabläufen mithilfe eines Softwaresystems [13, 27]. Dabei können sowohl einfache, isolierte Teilschritte als auch komplexe, umfangreiche Prozesse automatisiert werden [13]. Die Prozessautomatisierung, die im Rahmen dieser Arbeit entwickelt wurde, interagiert mit vielen verschiedenen Randsystemen und nutzt spezifische Verfahren, um die einzigartige Arbeitsweise des Systems zu ermöglichen und seinen geforderten Zweck zu erfüllen.

In diesem Kapitel werden die grundlegenden Kenntnisse vermittelt, die nötig für das Nachvollziehen der vorliegenden Arbeit sind. Des Weiteren wird aufgezeigt, wie die verwendeten Technologien und Praktiken im Zusammenhang mit der Prozessautomatisierung stehen. Dazu werden spezielle, anwendungsnahe Beispiele erläutert.

2.1 Extensible Markup Language

Die Extensible Markup Language (XML) ist eine hierarchisch aufgebaute Auszeichnungssprache. Sie wurde entwickelt, um Daten in einer strukturierten Form darzustellen, die leicht von Mensch und Maschine lesbar ist [26]. Im Vergleich zu unstrukturierten Textdateien ermöglicht XML eine, auf Eltern–Kind–Basis aufgebaute, Anordnung von Daten. Daraus resultiert eine Baumstruktur, die eine flexible und präzise Repräsentation von Datenobjekten ermöglicht [26]. Jene Flexibilität erlaubt es XML ein Werkzeug für verschiedene Probleme zu sein [27]. Dies spiegelt sich auch im Bereich der Datenmodellierung wieder [27]. Ein weiterer Vorteil des XML Formats liegt darin, dass es plattformunabhängig auf verschiedenen Systemen eingesetzt werden kann [16]. Dies macht XML zu einem geeigneten Format für die Speicherung und Übertragung von Daten und kann dabei

die Kommunikation zwischen unterschiedlichen Systemen und Anwendungen ermöglichen [16].

Um Daten einer Anwendung zu speichern oder zu übertragen, müssen die entsprechenden Datenobjekte serialisiert werden [26]. Das resultierende Format stellt die Daten dann hierarchisch durch spitz–geklammerte Elemente, auch Tags genannt, dar. In der Regel besteht ein XML–Objekt aus einem öffnenden und einem schließenden Tag. Dazwischen können weitere Daten liegen, die dann Kindelemente des umschließenden Elements sind. Diese können wieder in Form von XML–Objekten oder Rohdaten, wie Zahlen oder Strings, vorliegen. XML kann dabei eine Vielzahl von textuell kodierten Datentypen unterscheiden [26]. Die für diese Arbeit relevanten Datentypen sind: Strings, Zahlen und Wahrheitswerte. Aus den dargestellten Fakten lässt sich schließen, dass das XML–Format eine robuste Arbeitsgrundlage in Hinsicht auf Wiederverwendbarkeit und Zuverlässigkeit in der Datenübertragung und –modellierung ist.

XML für Multilevel Business Artifacts

Multilevel Business Artifacts (MBA) setzen XML in der Darstellung und Verwaltung ein [27]. Die MBAs helfen dabei, Arbeitsabläufe präziser darzustellen, indem sie komplexe Arbeitsabläufe zusammenfassen. Dies unterstützt die Entwicklung von Softwaresystemen, welche Geschäftsprozesse automatisieren sollen. Die klassischen Business Artifacts können durch die Verwendung einer hierarchischen Repräsentation erweitert werden. Da diese nur für kleinere Anwendungen funktionieren, müssen komplizierte Anwendungen aus mehreren einzelnen Business Artifacts aufgebaut werden. Mittels einer XML–Erweiterung, dem State Chart XML, kann das gewöhnliche Modell mehrere Business Artifacts zu einem mehrstufigen Modell, dem MBA, zusammengefasst werden. XML kann solch eine hierarchische Struktur bieten und damit ermöglichen, dass komplexe Arbeitsabläufe einfacher und in direkterer Form, mitsamt ihrer Abhängigkeiten, modelliert werden können.

2.2 JavaScript Object Notation

Eine Alternative zu XML kann die JavaScript Object Notation (JSON) bieten. Diese ist ein auf Text basierendes Datenformat, das ebenfalls die Übertragung und Speicherung von Daten in einer strukturierten Form ermöglicht [31]. Die Syntax ist für Menschen genauso leicht lesbar [18, 31]. Dabei werden Daten in Form von Schlüssel–Wert–Paar Zuordnungen in den JSON–Objekten abgelegt und erlauben eine eindeutige Identifizierung [31]. Als Datentypen können vor allem Strings, Zahlen und Wahrheitswerte erkannt werden. Ein besonderes Merkmal des Formats ist allerdings die direkte Unterstützung von Arrays und der wissenschaftlichen Notation von Zahlen mithilfe von Zehnerpotenzen [31].

JSON–Objekte werden durch geschweifte Klammern begrenzt und sind, anders als in XML, nicht explizit typisiert [31]. Zwischen diesen Klammern befinden sich die Schlüssel–Wert–Paare, welche die Daten des Objektes darstellen [31]. Schlüssel werden mithilfe eines Stringliterals benannt. Ihre Werte können dann beliebig, ohne weitere Deklaration eines Typs, angegeben werden. Da Werte wiederum auch Objekte sein können, ist eine beliebige Verschachtelung möglich, mithilfe derer zusammengesetzte Datenobjekte dargestellt werden können [31].

JSON ist ein beliebtes Datenformat und findet vielfältige Anwendungszwecke in unterschiedlichen Bereichen [22, 28]. Durch die direkte Integration in JavaScript ist JSON ein wichtiger Teil des modernen Internets [31]. Viele Schnittstellen, die über das Internet ansprechbar sind, kommunizieren mithilfe von JSON [28]. Die Sprache arbeitet mit geringem sprachlichen Überschuss, im Vergleich zu der bereits beschriebenen Sprache XML. Mit dieser leichtgewichtigen Darstellung eignet sich das Format entsprechend auch für große Datenmengen [22].

Übertragungsformat JSON in der Prozessautomatisierung

In der heutigen Zeit haben sich Representational State Transfer (REST) APIs als weit verbreitete Architektur für Schnittstellen und Softwaresysteme durchgesetzt [10]. Einer der großen Einsatzgebiete der REST Architektur liegt in den webbasierten Diensten [11]. Das JSON Format spielt dabei eine wichtige Rolle, da es inzwischen zu den häufigst verwendeten Datenübertragungsformaten im Internet zählt [28].

REST APIs bieten eine Vielzahl von Nutzungsmöglichkeiten. Dies beruht unter anderem auf ihrem einfachen und einheitlichem Interface [10, 35]. Unter den vielseitigen Anwendungsdomänen findet sich auch die Prozessautomatisierung [17]. REST APIs erlauben dabei zum Beispiel den Zugriff auf Dienste und Geräte innerhalb der Automatisierung und ermöglichen nahtlose Kommunikation über eine einheitliche Schnittstelle [17].

Ein Vorteil der Nutzung von REST APIs basierend auf dem JSON Format ist dabei, dass die Flexibilität des Formats und der Schnittstelle eine breite Interoperabilität zwischen beliebig aufgebauten Systemen ermöglicht [11]. Folglich lassen sich Systeme leichter ineinander integrieren und es wird eine Umgebung geschaffen, in der auch komplexe Geschäftsprozesse nahtlos miteinander arbeiten können.

2.3 Requirements Engineering

Das Requirements Engineering nimmt eine zentrale Rolle im Softwareentwicklungsprozess ein [24]. Dabei umfasst es eine Vielzahl von Techniken, die eingesetzt werden, um Anforderungen an ein Softwaresystem zu erheben, analysieren und dokumentieren [24]. Der Umgang mit den Anforderungen

ist eng mit dem Erfolg eines Projekts verbunden und kann sich direkt auf weitere finanzielle Zusatzbelastungen auswirken [33]. Dies äußert sich beispielsweise darin, dass die entstandenen Missverständnisse korrigiert werden müssen. Es werden mehrere Unterprozesse im Requirements Engineering Prozess durchlaufen, die einen angemessenen Umgang mit den Anforderungen ermöglichen. Dies unterstützt, Missverständnisse frühzeitig zu erkennen. Insbesondere sind die Prozesse der Erhebung, Dokumentation, Verhandlung und Verfolgung der Anforderungen zentrale Bestandteile [34]. Ein weiteres Ziel des Requirements Engineering Prozesses ist es, die Stakeholder mit ihren konkreten Zielen kennenzulernen und zu verstehen, um klare Zielvorstellungen an das System zu erheben [24].

Stakeholder

Ein Stakeholder im Software Requirements Engineering umfasst jede Person oder Gruppe, die in einer beliebigen Weise von der Entwicklung der Software betroffen ist [7]. Dabei kann Betroffenheit beispielsweise durch die Bedienung der Software auftreten. Es ist wichtig anzumerken, dass Stakeholder aus verschiedenen Fachrichtungen stammen sowie unterschiedliche Interessen und Hintergründe haben können [33]. Zu ihnen gehören beispielsweise die Auftraggeber und Endnutzer [29]. Die Stakeholder sollten entsprechend bei der Softwareentwicklung berücksichtigt werden, da die Software direkte oder indirekte Auswirkungen auf diese Gruppen haben kann [33].

In einigen Fällen kann es passieren, dass die verschiedenen Interessen der Stakeholder nicht miteinander kompatibel sind [33]. Beispielsweise wünscht sich eine Interessengruppe eine einfache Benutzeroberfläche, während sich eine andere Interessengruppe einen großen Umfang von Funktionen und Experten–Modi wünscht. Solche Konflikte müssen entsprechend in einem Einigungsprozess diskutiert und aufgelöst werden.

Es ist notwendig, dass zunächst die relevanten Stakeholder identifiziert werden, bevor diese in den Prozess des Requirements Engineering und der Softwareentwicklung mit eingebunden werden können. Sharp et al. [29] betonen dabei, dass der erste Schritt zur Identifikation der Stakeholder die Erfassung der *Grundlegenden Stakeholder* sei. Diese sind die Endnutzer, Entwickler, Gesetzgeber und Auftraggeber [29]. Aus diesen vier Gruppen *Grundlegender Stakeholder* können weitere Stakeholder identifiziert werden. Dies geschieht durch das Nachvollziehen erfolgender Interaktionen mit anderen Personen oder Personengruppen. Dadurch kann sichergestellt werden, dass am Ende die Software alle Stakeholder zufriedenstellt [24].

Da die Stakeholder im Verlauf des Entwicklungsprozesses dazu lernen und sich weiterentwickeln, ist es wichtig zu berücksichtigen, dass sich ihre Wünsche und Interessen mittel– bis langfristig verändern können [24]. Weil Stakeholder eine wertvolle Quelle für Erkenntnisse und Informationen über den Zweck der Software und ihre Anforderungen sind, ist es von hoher

Relevanz, sie aktiv mit einzubinden sowie ihre Bedürfnisse zu kennen und zu berücksichtigen [24, 29].

Anforderungserhebung

Mithilfe der Stakeholder können Anforderungen an das System nicht nur gewonnen, sondern auch verfeinert werden. Daraus lässt sich eine umfassende und präzise Basis für die Softwareentwicklung schaffen [24]. Die Erhebung von Anforderungen ist jedoch keine triviale Aufgabe [24]. Ein Grund dafür ist, dass es keine perfekte, allgemeingültige Methode zur Erhebung von Anforderungen gibt [24]. Eine sorgfältige Abwägung und Anpassung der Methoden an die spezifischen Anforderungen und Eigenschaften des Projekts ist erforderlich [24, 33]. Die Qualität der erhobenen Anforderungen spielt eine wichtige Rolle für den Erfolg eines Projektes [33]. Schlecht oder nicht erhobene Anforderungen können beispielsweise für fehlende oder falsch umgesetzte Funktionen sorgen. Es ist daher von großer Bedeutung, geeignete Ermittlungsmethoden für die Anforderungserhebung zu finden, um den spezifischen Ansprüchen des jeweiligen Projektes gerecht zu werden [24, 33]. Ein Beispiel für solche Ermittlungsmethoden ist das klassische Interview, in welchem die Stakeholder nach ihren Vorstellungen und Bedürfnissen befragt werden. [24]. Die verschiedenen Methoden haben dabei jeweils ihre eigenen Vor– und Nachteile [24]. Die Wahl solcher Ermittlungsmethoden ist in sich eine komplexe Aufgabe und der erste Schritt der Anforderungserhebung [24].

Relevanz in der Prozessautomatisierung

Im Bereich der Geschäftsprozesse und Prozessautomatisierung nimmt der Requirements Engineering Prozess eine betrachtenswerte Stellung ein und sollte eng mit dem Prozessdesign gekoppelt werden, wie von Weidlich et al. [34] hervorgehoben wird. Die Verzahnung dieser beiden Prozesse ist von hoher Relevanz, da ihre Entkopplung zu Problemen führen kann. Ist beispielsweise die Kommunikation zwischen den Prozessdesignern und den Requirements Ingenieuren nicht ausreichend, können Funktionalitäten in der Software fehlen oder doppelt auftreten. Fehlen spezifizierte Funktionen, nehmen die Prozessdesigner und die Requirements Ingenieure an, die jeweils andere Rolle würde dafür verantwortlich sein. Genau gegenteilig ist es, wenn Funktionen doppelt auftreten. Dabei wird von den Prozessdesignern und Requirements Ingenieuren angenommen, nur sie selbst wären dafür verantwortlich. Die Unabhängigkeit dieser beiden Prozesse kann zu falschen Annahmen und Anforderungen führen. Jedoch ist dies nur eine Teilmenge der möglichen Probleme, die entstehen können. Korrekt und sorgfältig durchgeführtes Requirements Engineering sollte mit dem Prozessdesign eng zusammenarbeiten, um einen positiven Einfluss auf die Entwicklung von Prozessautomatisierungen durch Softwaresysteme zu haben.

2.4 Agile Softwareentwicklung

Die agile Softwareentwicklung ist ein Sammelbegriff für verschiedene Praktiken, Methoden und Prozesse, welche den Prinzipien und Ideen des *Agilen Manifestes* folgen [2]. Zu den bekannteren dieser Methoden zählen beispielsweise *Scrum* und *Extreme Programming* (XP). Der agile Ansatz ist aus der Notwendigkeit heraus entstanden, mit Veränderungen innerhalb der Entwicklung flexibel umgehen zu können [19]. Die Idee der Agilität stellt dabei einen entsprechenden Gegenpol zu traditionellen Entwicklungsprozessen dar [19]. Im traditionellen Ansatz unterzeichnen Auftragnehmer und –geber jeweils einen Vertrag, der den Entwicklungsprozess, die zu lösenden Aufgaben und die zur Abnahme geforderten Mindestanforderungen spezifizieren [19]. Im agilen Ansatz gibt es auch Verträge – da ohne Verträge keine rechtliche Bindung eingegangen werden kann – die Teilhabe der Kunden wird jedoch über ausgiebige Vertragsverhandlungen gestellt [2].

In der agilen Softwareentwicklung werden die Interaktionen zwischen den Menschen im Entwicklungsprozess und die Teilhabe der Betroffenen daran mit besonderem Stellenwert belegt [2, 19]. Dies spiegelt sich auch im *Agilen Manifest* wieder. Die verschiedenen Arten der Kommunikation werden dabei unterschiedlich bewertet, sodass *Face–to–Face* Kommunikation gegenüber anderen Kommunikationswegen bevorzugt wird [19]. Der persönlichen Kommunikation wird besonderer Wert zugeschrieben, was sich auch in den Praktiken der agilen Softwareentwicklung erkennen lässt [19]. Beispielsweise gibt es im Scrum das sogenannte *Daily–Scrum Meeting* und im XP das *Stand–Up Meeting* [2, 19]. Konzeptionell sind sich die beiden ähnlich und betonen die regelmäßige, zwischenmenschliche Kommunikation. Das *Daily–Scrum Meeting* ist strukturiert aufgebaut, während das *Stand–Up Meeting* keinem genaueren Ablauf folgt [2].

Der agile Ansatz ist in der heutigen Softwareentwicklung weit verbreitet und erprobt [3]. Der Grad, zu dem agile Softwareentwicklung eingesetzt werden sollte, ist jedoch von den Faktoren innerhalb eines Projektes abhängig [3]. Die Verwendung eines agilen Ansatzes hat also zum Vorteil, dass die Software in Kollaboration mit den eigentlichen Nutzern und Betroffenen entwickelt werden kann und im Verlaufe der Entwicklung schnelle und flexible Reaktionen auf Veränderungen möglich sind.

2.5 Analytic Hierarchy Process

Der Analytic Hierarchy Process (AHP) ist eine einfach gehaltene Methode, die mit dem Ziel entwickelt wurde, kriterienbasierte Entscheidungen zu treffen [25]. Der Erfinder Thomas L. Saaty [20, 25] hat dafür eine strukturierte Vorgehensweise entworfen, die mithilfe mathematischer Berechnungen Kriterien paarweise gegeneinander abwägt. Dies ermöglicht eine

klare und annäherungsweise korrekte Abbildung der Gewichtungen in einem Entscheidungsprozess [20]. Der hierarchische Aspekt drückt sich dabei durch die abstrakte, baumstrukturartige Herangehensweise aus. Im einfachsten Fall ist diese aus drei Ebenen aufgebaut: dem Ziel, den Kriterien und den Alternativen [25]. Mithilfe des Entscheidungsprozesses soll das Ziel erreicht werden. Formal sind die Kriterien die relevanten Einflussfaktoren, mithilfe derer eine Entscheidung getroffen werden kann [25]. Die Alternativen sind die Instanzen, welche die Lösungskandidaten der Entscheidungsfindung darstellen [25].

Im Ermittlungsprozess werden paarweise Vergleiche der Kriterien von den verschiedenen Alternativen angestellt, anhand derer eine prozentuale Gewichtung errechnet wird [25]. Die resultierende Gewichtung gibt dabei an, zu welchem Grad eine Alternative eine gute Entscheidung für das zu erreichende Ziel ist [25]. Das Verfahren hat den Nachteil, dass im Voraus eine Expertenschätzung durchgeführt werden muss [20]. Dort werden die einzelnen Kriterien paarweise gewichtet. Dieser Prozess der Schätzung ist nicht standardisiert und beruht auf den gesammelten Erfahrungen der Experten [20]. Dennoch eignet sich die Methode, um komplexe Entscheidungen zu treffen und diese zu formalisieren, da mit guten Schätzungen effizient Entscheidungen mit vielen Variablen gefällt werden können [20, 25].

Kapitel 3

Der Entwicklungsprozess

„*Customer collaboration over contract negotiation*"[2] ist einer der zentralen Punkte im *Agilen Manifest*. In dieser Arbeit wurde unter anderem das Ziel verfolgt, die Kunden in den Entwicklungsprozess der Software bestmöglich zu integrieren. Dies erlaubt den Kunden, den aktuellen Entwicklungszustand des Projekts zu jeder Zeit nachvollziehen zu können und gegebenenfalls Änderungen oder weitere Wünsche zu äußern. Damit dieses Ziel erreicht werden konnte, wurde der Entwicklungsprozess speziell auf das zugrunde liegende Projekt und die zur Verfügung stehenden Mittel abgestimmt und stellt eine Kombination aus den traditionellen und agilen Ansätzen dar. Mit dieser Symbiose ging einher, dass die Vorteile beider Ansätze, darunter das strukturierte und dokumentierte Vorgehen sowie die Flexibilität, effektiv genutzt werden konnten. Dies half dabei, die speziellen Anforderungen an das Projekt effektiv umzusetzen.

Zu Beginn wurde auf eine traditionelle und ausführliche Durchführung des Requirements Engineering gesetzt und streng dokumentiert. So konnte ein klar nachvollziehbarer und eindeutig zurückverfolgbarer Überblick über die Ideen und Wünsche der Kunden gewonnen werden. In der eigentlichen Implementierungsphase wurde auf agile Methoden zurückgegriffen. Diese haben eine flexible Entwicklung der Software und Steuerung durch den Kunden erlaubt.

In diesem Kapitel werden die eingesetzten Verfahren und Praktiken im Detail erläutert. Der Aufbau des Kapitels spiegelt dabei den tatsächlichen Ablauf des Entwicklungsprozesses wieder, um den Prozess der Entwicklung chronologisch nachvollziehbar zu erklären.

3.1 Requirements Engineering

Am Anfang des Entwicklungsprozesses wurden die Anforderungen an die Software und die Arbeitsweise mittels gängiger Techniken des Requirements Engineering erhoben und verfeinert. Dazu gehören Interviews, Anforderun-

gen erahnen, Prototyping, das Erstellen von Modellen, die Entwicklung von Use Cases und Analysen. Es wurde eine strukturierte Vorgehensweise gewählt, um die Anforderungen bestmöglich beherrschbar zu halten. Dies umfasste zunächst die Erhebung, dann die Interpretation und weiter die Verhandlung von Anforderungen. Durch die detaillierte Anwendung der Requirements Engineering Praktiken konnte ein klares Bild der benötigten und gewünschten Funktionen gewonnen werden. Im weiteren Verlauf dieser Arbeit wird gezeigt, dass dies für die Implementierung eine große Hilfestellung war und dazu beigetragen hat, die Kundenzufriedenheit im Entwicklungsprozess zu verbessern. So konnten zeitintensive Änderungswünsche in der Implementierung zu großen Teilen vermieden werden.

Dennoch wurde im späteren Verlauf des Entwicklungsprozesses das Requirements Engineering nicht vernachlässigt. Anforderungen wurden weiterhin erhoben und verfeinert. Dies war jedoch ein nebenläufiger Prozess. Der Fokus lag danach auf einer möglichst präzisen Umsetzung des geplanten Funktionsumfangs. Um sicherzustellen, dass die Software den Anforderungen der Kunden entspricht, wurden verschiedene Techniken eingesetzt. Darunter fallen zum Beispiel Mock–Ups und Prototypen. In den nächsten Abschnitten werden die verschiedenen, eingesetzten Praktiken erläutert. Es wird dabei auf die für diese Arbeit wichtigen Aspekte eingegangen und betont, weshalb das Vorgehen die Entwicklung der Software unterstützt hat.

Stakeholderanalyse

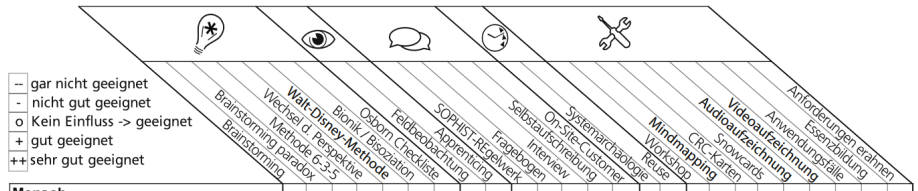
Da für die Anforderungserhebung ein Wissen über die Stakeholder benötigt wird, wurde zunächst im kleinen Umfang die Stakeholderanalyse durchgeführt. Konkret konnten die Stakeholder mittels des vorausgehenden Wissens über das Softwareprojekt und der Struktur des Fachgebiets Software Engineering sowie weiteren Gesprächen mit den Kunden analysiert werden. Als Stakeholder gelten in diesem Projekt diejenigen Personen, welche ein berechtigtes Interesse an dem Erfolg der Software haben [7]. Manche dieser Stakeholder sind sich über die Existenz der Software allerdings nicht bewusst. Dennoch werden diese als Stakeholder aufgelistet, da sie ein Misserfolg der Software beeinflussen kann. In der folgenden Tabelle 3.1 werden die Stakeholder zusammen mit ihren Interessen aufgezählt.

Entwicklerteams	Möchten, dass die Software korrekt dem Kunden zu Verfügung gestellt wird.
SWP Kunden	Möchten ihre Software zuverlässig erhalten.
FG Software Engineering	Möchte einen reibungslosen Ablauf des SWPs und den guten Ruf behalten.

Tabelle 3.1: Die Stakeholder und ihre (induzierten) Wünsche.

Anforderungserhebung

Die Erhebung der Anforderungen ist ein notwendiger Schritt, um die Ziele und Erwartungen der Stakeholder zu verstehen [29]. Unter Anwendung verschiedener Techniken werden die Stakeholder dabei unterstützt, ihre persönlichen Wünsche und Anforderungen an das System preiszugeben. Auf Grundlage der Entscheidungsmatrix 3.1 nach Rupp et al. [24] konnten entsprechend passende Techniken ausgewählt und zum Einsatz gebracht werden.



-- gar nicht geeignet
- nicht gut geeignet
o Kein Einfluss -> geeignet
+ gut geeignet
++ sehr gut geeignet

	Brainstorming	Brainstorming paradox	Methode 6-3-5	Wechsel d. Perspektive	**Walt-Disney-Methode**	Bionik / Bisoziation	Osborn Checkliste	Feldbeobachtung	Apprenticing	SOPHIST-Regelwerk	Fragebogen	Interview	Selbstaufschreibung	On-Site-Customer	Systemarchäologie	Workshop	Reise	**Mindmapping**	CRC-Karten	Showcards	**Audioaufzeichnung**	**Videoaufzeichnung**	Anwendungsfälle	Essenzbildung	Anforderungen erahnen
Mensch																									
geringe Motivation	-	--	--	--	--	--	-	+	-	o	+	+	--	--	o	o	+	-	o	+	o	+	o	o	++
schlechte kommunikative Fähigkeiten	-	-	-	-	o	o	-	-	++	++	o	-	--	-	o	o	-	o	-	-	o	o	+	o	+
implizites Wissen	+	++	++	+	+	+	+	++	++	++	--	o	--	+	o	o	o	+	o	o	o	+	o	o	++
geringes Abstraktionsvermögen	-	--	-	-	-	++	-	++	++	+	+	+	-	-	o	o	o	-	-	-	o	o	-	++	+
divergierende Stakeholder-**Meinungen**	-	+	-	-	-	o	+	-	-	+	++	o	-	--	o	o	+	-	+	+	o	o	+	++	o
problematische Gruppendynamik	--	--	+	o	--	o	+	--	++	-	o	o	+	+	o	o	--	-	-	o	--	--	o	o	o
Organisatorische Rahmenbedingungen																									
Neuentwicklung	++	+	++	++	++	+	+	o	o	o	+	+	+	+	--	o	+	++	++	++	o	o	o	+	+
Altsystemerweiterung	o	+	o	o	o	o	++	+	+	o	+	+	+	+	++	+	+	o	+	+	o	o	o	o	+
Individualentwicklung	o	o	o	o	o	o	+	+	+	o	+	+	++	++	+	o	+	o	+	o	o	o	o	o	+
Produktentwicklung	++	+	++	++	++	+	+	-	--	o	+	-	-	--	+	o	o	++	o	-	-	-	o	o	+
fixiertes, knappes Projektbudget	o	o	-	o	o	--	-	-	--	+	++	+	+	-	--	++	o	o	o	o	++	-	o	o	++
hohe Verteilung d. Stakeholder	--	--	--	-	-	-	o	o	o	o	+	-	++	-	o	o	--	-	--	--	o	o	o	o	o
schlechte Verfügbarkeit d. Stakeholder	-	-	+	-	-	-	-	++	--	o	+	-	--	--	o	++	--	-	--	--	++	++	o	o	++
hohe Zahl von Stakeholdern	o	-	-	o	o	-	-	-	--	o	++	--	--	--	o	o	--	o	--	--	-	-	+	+	o
Fachlicher Inhalt der Anforderungen																									
hohe Kritikalität des Systems	o	o	+	+	+	-	+	+	--	++	+	+	+	++	++	-	+	o	+	+	++	+	o	+	-
großer Systemumfang	o	o	o	o	o	o	--	++	+	o	--	+	+	+	+	++	+	o	-	+	o	o	++	++	+
hohe Komplexität der Systemabläufe	+	-	o	+	+	+	+	--	--	o	--	+	+	+	+	+	+	++	o	o	o	o	+	o	o
geringe Beobachtbarkeit	+	o	o	+	o	++	+	--	+	o	+	+	+	+	+	+	+	+	o	o	o	--	o	+	o
nicht funktionale Anforderungen	-	-	+	+	o	+	o	-	+	o	--	-	-	-	+	+	-	o	+	o	o	+	o	o	o
unbekanntes Fachgebiet	o	o	+	o	-	-	o	+	++	o	--	+	++	+	++	--	+	o	o	o	o	o	+	o	--
abstrakte Anforderungen	+	+	+	+	+	+	++	--	-	+	++	++	+	+	-	o	+	+	++	++	o	o	++	++	-
detaillierte Anforderungen	-	-	-	-	-	-	o	+	++	++	o	++	+	++	++	++	o	-	-	-	+	+	-	o	++

Abbildung 3.1: Entscheidungsmatrix zur Findung geeigneter Anforderungserhebungsmethoden [24].

Das Auswahlverfahren der, in dieser Arbeit verwendeten, Methoden beruht auf der Funktionsweise der Entscheidungsmatrix 3.1. Dabei wird spaltenweise untersucht, ob die beschriebenen Aussagen der Zeilen zutreffen. Nach der Auszählung kann erkannt werden, welche dieser Methoden am geeignetsten für die Erhebung der Anforderungen im Kontext des Projektes sind. Die gewählten Methoden waren dabei zum einen das *Interview* und zum anderen das *Erahnen der Anforderungen*. Die Interviews wurden wöchentlich mit den Kunden abgehalten. Mithilfe von vorbereiteten Fragen und Tagesordnungen konnten die Anforderungen strukturiert erhoben werden. Über diese Phase hinweg wurden außerdem Anforderungen erahnt und

den Kunden vorgeschlagen. Einer dieser Vorschläge war es, die E–Mails direkt über die Software zu versenden. Die meisten Vorschläge wurden als Anforderungen akzeptiert und konnten die Ideen der Kunden anregen.

Verhandlungen und Rücksprachen

Nachdem die Anforderungen erhoben wurden, fand im Rahmen der anschließenden Analysephase eine Verhandlung über die erhobenen Anforderungen mit den Kunden statt. Das Ziel war es dabei, die Anforderungen genauer zu verstehen und mögliche Konflikte frühzeitig anzugehen. Die Kommunikation erfolgte in mündlicher Form, während weiterer Interviews, und elektronisch über E–Mail. In dieser Phase wurden insbesondere jene Anforderungen diskutiert, die aus Sicht der Implementierung fragwürdig oder missverständlich erschienen. Bei diesen Diskussionen wurden die Kunden angeregt, miteinander in einen Austausch zu treten. Zu den Diskussionen ist es in den meisten Interviews auch gekommen und gelang unter anderem durch das gezielte Stellen von Fragen. Dies diente dazu, einen umfassenden Ideenaustausch zu fördern. So konnte eine gemeinsame Basis geschaffen werden.

Im Rahmen der Analyse wurden zudem Anforderungen in eine Gegenüberstellung gebracht. Dabei mussten die Kunden aus einer Liste von möglichen Anforderungen eine handvoll auswählen, die sie priorisieren. So konnte den Kunden eine neue Sichtweise auf das Projekt gegeben werden und half dabei, eine nutzerorientierte Liste von Anforderungen zu erstellen. Eingesetzt wurde dies beispielsweise bei der Auswahl der priorisierten Qualitätsanforderungen nach der ISO 25010 [1]. Dabei wurden den Kunden die, in der ISO spezifizierten, Qualitätsmerkmale präsentiert. Daraufhin sollten sie wählen, welche dieser Merkmale sie priorisieren würden.

Am Ende des Requirements Engineering Prozesses konnten über zwanzig Anforderungen ermittelt werden. Diese wurden sortiert nach Verwandtschaft und Anforderungsart. Unterschieden wurde dazu in funktionelle und nicht–funktionelle Anforderungen. In der nachfolgenden Tabelle 3.2 ist ein Auszug dieser Anforderungen. Die Gesamtheit der Anforderungen befinden sich im Anhang A.1.

[R01]	Das Programm muss eine bestehende XML–Datei lesen können, um daraus Projekt– und Kundeninformationen abzuleiten.
[R02]	Ein SE–Mitarbeiter soll der Anwendung einen Gitlab–Server übergeben können.
[R18]	Das Programm soll fehlertolerant sein.

Tabelle 3.2: Auszug aus den Anforderungen.

[1] https://iso25000.com/index.php/en/iso-25000-standards/iso-25010

Es konnte sich außerdem ein weiterer wichtiger Aspekt der Anforderungsanalyse im Verlauf bemerkbar machen. Durch die Priorisierung und Gegenüberstellung von Anforderungen zeigte sich, dass manche potentiellen Anforderungen nicht berücksichtigt werden müssen. Darunter fällt beispielsweise die Unterstützung weiterer Betriebssysteme. Für die Implementierung der Software konnten deshalb Abstriche gemacht werden.

Abstriche

Diese Abstriche beziehen sich auf Funktionen oder Anforderungen, die entweder nicht von ausreichender Relevanz für die Nutzer oder für das Softwaresystem und seinen Zweck sind. Es ist äußerst sinnvoll, Abstriche zu machen [4]. Dadurch können in Projekten wichtige Ressourcen wie Zeit und Kosten eingespart werden. Dies erlaubt, wichtigere Anforderungen zu priorisieren und effizienter umsetzen zu können. Ob und welche Abstriche gemacht werden, findet im Rahmen einer gemeinsamen Diskussion zwischen den Stakeholdern und den Entwicklern statt. In der Verhandlungsphase konnten mit den Kunden die folgenden Aspekte als unkritisch bewertet werden: Das *Durchfallen eines Teams*, die *Sicherheit des Systems* und die *Übertragbarkeit auf Betriebssysteme, die nicht Windows sind.*

Damit die Abstriche die essentiellen Funktionalitäten und Anforderungen des Systems nicht negativ beeinträchtigen, wurden die Abstriche jeweils sorgfältig abgewogen. So konnte für jeden einzelnen Abstrich sichergestellt werden, dass keine Behinderung der gegenwärtigen und zukünftigen Operation stattfinden wird.

Konzeptpapiere

Im Verlaufe des Requirements Engineering Prozesses wurden im Rahmen der relevanten Arbeitspakete, wie die Umsetzung der automatischen Branchauswahl, detaillierte Konzeptpapiere entwickelt (Anhang B). Diese Konzeptpapiere verfolgten den Zweck, Ideen und Vorschläge für die Umsetzung der entsprechenden Anforderungen zu formulieren und zeitgleich potentielle Schwierigkeiten und Risiken aufzuzeigen. Die Konzeptpapiere haben den gesamten Entwicklungsprozess begleitet und erfüllten unterdessen zwei weitere Funktionen.

Zum einen dienten die Konzeptpapiere als eine wichtige Referenz während der Implementierung. Dort boten sie eine umfangreiche Darstellung der geplanten Lösungsansätze und haben den Grundstein für die technische Umsetzung der Anforderungen gelegt. Die Konzeptpapiere enthielten tiefgreifende Beschreibungen der Funktionalitäten, Designentscheidungen und technische Aspekte, die für das Verständnis und die Entwicklung wichtig waren. Dadurch haben sie eine klare, einheitliche Richtung während des Entwicklungsprozesses vorgegeben.

Zum anderen fungierten die Konzeptpapiere als eine Interpretation der erhobenen Anforderungen der Kunden. Sie konnten als wichtige Kommunikationsgrundlage dienen und halfen bei der Validierung der interpretierten und dokumentierten Anforderungen. Vor den wöchentlichen Meetings wurden die Konzeptpapiere den Kunden präsentiert, um in den Sitzungen gemeinsam Vorschläge und Entscheidungen zu diskutieren. Durch die Erläuterung und Visualisierung der geplanten Lösungen, konnten potentielle Missverständnisse vermieden werden. So wurde die aktive Teilnahme der Kunden am Entwicklungsprozess weiter gefördert.

Die regelmäßige Vorlage der Konzeptpapiere war eine Hilfestellung für die Kunden, um den Entwicklungsprozess transparent nachvollziehen zu können und die Lösungen, ihren Wünschen entsprechend, zu prüfen und Fragen zu stellen. Dieser iterative Austausch trug dazu bei, dass die Software den Erwartungen und Anforderungen bestmöglich entspricht. Innerhalb des Requirements Engineering Prozesses dieser Arbeit waren die Konzeptpapiere ein zentrales Instrument der Interpretation und Validierung von Anforderungen, um eine zielgerichtete Entwicklung zu stärken.

Mock–Ups und Prototypen

Vor der eigentlichen Implementierung der Software wurden mithilfe von Mock–Ups in der Validierungsphase grundlegende Konzepte und ein Verständnis der Benutzeroberfläche weiterentwickelt. Die Mock–Ups haben eine große Rolle für spätere Layout–Entscheidungen gespielt, indem sie den Aufbau der Benutzeroberfläche grob skizzierten. So konnte leicht ermittelt werden, welche Funktionen kontextuell miteinander verbunden sind und somit gemeinsam dargestellt werden müssen. Die Skizzen der Benutzeroberfläche zeigten außerdem auf, welche Funktionen in der Zukunft noch benötigt werden. Die Navigation der Benutzeroberfläche wurde in den ersten Skizzen zunächst nicht betrachtet. Durch weitere Tests und Gedankenspiele mit den Mock–Ups wurde die Integration einer Seitenleiste in späteren Iterationen für sinnvoll erachtet.

Bei der Erstellung der Mock–Ups wurde der Fokus auf verschiedene Aspekte gelegt. Zum einen war es wichtig, das visuelle Erscheinungsbild grob zu erfassen und ein Nährboden für weitere Ideen zu bieten. Dazu wurden die Elemente der Benutzeroberfläche platzhalterartig dargestellt. Dies hat Spielraum für weitere Kreativität geboten. Ziel der Mock–Ups war es nicht, ästhetisch ansprechend zu wirken, wie in Abbildung 3.2 gezeigt wird. Zum anderen boten die Mock–Ups die Möglichkeit, verschiedene Design Ideen auszuprobieren, bevor diese implementiert werden. Mithilfe der Mock–Ups konnte so in der Implementierung Zeit eingespart werden, da das Design und die Gruppierung von Funktionen bereits durch das Requirements Engineering abgedeckt und von den Kunden zu einem gewissen Teil akzeptiert wurden.

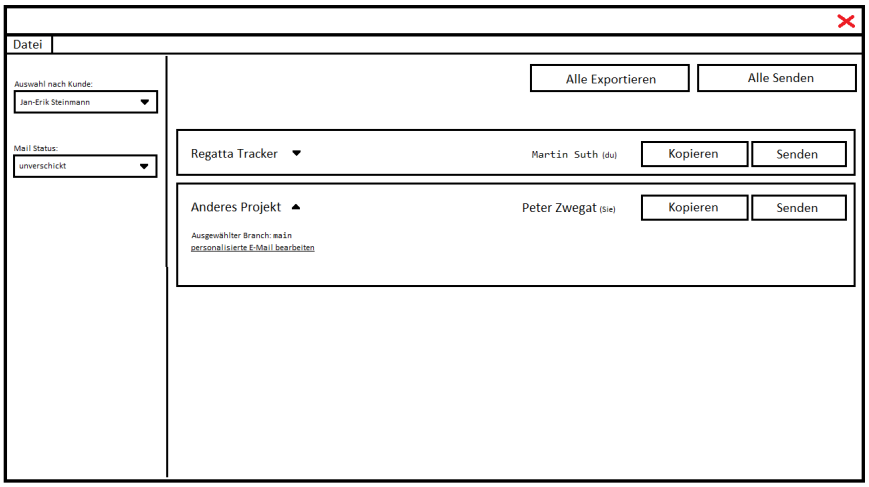


Abbildung 3.2: Eine erste Skizze des User Interfaces in der Hauptansicht.

Die Mock–Ups dienten danach als Grundlage für entsprechende Prototypen. Diese wurden entwickelt, um das Design der Software weiter zu verfeinern und die Anforderungen an die Funktionen und die Benutzeroberfläche noch besser verstehen zu können. Des Weiteren boten die Prototypen eine Möglichkeit, die Interpretation der Anforderungen den Kunden visuell zu präsentieren. Im Rahmen der Validierung erhielten die Kunden somit eine weitere Gelegenheit, in den Entwicklungsprozess einzugreifen und diesen in die gewünschte Richtung zu lenken. Die Prototypen durchliefen verschiedene Stufen. Begonnen wurde mit einer Art *Papier–Prototypen* (Abb. 3.3), welche in digitaler Form angefertigt wurden und gingen bis hin zu programmierten *Proof–of–Concepts*.

Die Papier–Prototypen wurden verwendet, um die zuvor gewählten Designentscheidungen in Hinsicht auf ihre Benutzerfreundlichkeit und ihrer intuitiven Verständlichkeit zu testen. Dazu simulierten diese verschiedene Interaktionsabläufe, welche direkt von den Kunden ausprobiert werden konnten. Ihre Entwicklung erlaubte, frühzeitig Missverständnisse und potentielle Probleme zu erkennen. Die Papier–Prototypen durchliefen einen iterativen Entwicklungszyklus und wurden den Ergebnissen der Diskussionen entsprechend angepasst. Sie trugen maßgeblich zur Entwicklung der Software bei, indem sie die Struktur und Vision der Software geformt haben.

Die Proof–of–Concepts wurden versetzt zu den Papier–Prototypen entwickelt und dienten dem Zweck, einzelne Konzepte der Problemstellung zu erforschen und im Anschluss mit den Kunden zu diskutieren. Der Fokus lag dabei auf einzelnen, spezifischen Aspekten der umfassenden Software, wie beispielsweise die Generierung des Mailtextes. Es war nicht das Ziel der Proof–of–Concepts, ein vollständiges Bild einer fertigen Software darzustellen. Wichtiger war es, die funktionellen Anforderungen an die

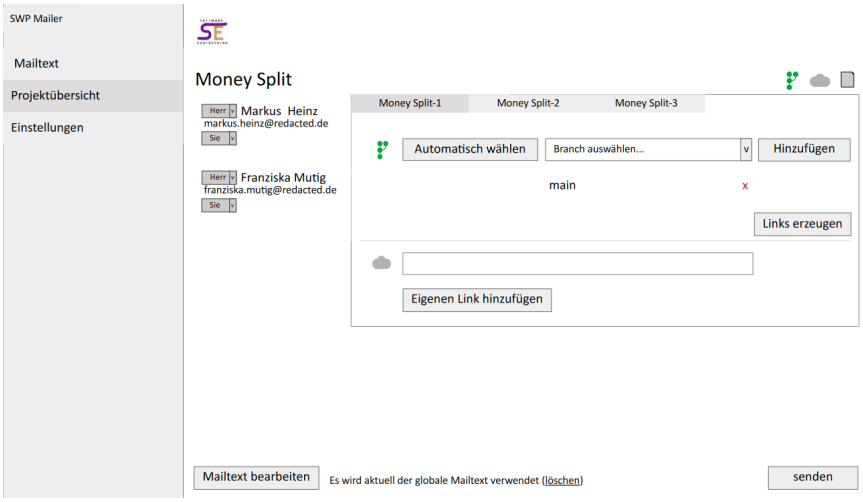


Abbildung 3.3: Ein digitaler Papier–Prototyp, der die grundlegenden Funktionen darstellen soll. Die Abbildung zeigt die Projektansicht des Prototypen.

Software zu veranschaulichen und exemplarisch erste Lösungsansätze für die zugrunde liegenden Probleme zu präsentieren.

Durch die Entwicklung der Proof–of–Concepts konnten die ausgewählten Aspekte untersucht werden und gemeinsam mit den Kunden bezüglich ihrer Sinnhaftigkeit diskutiert werden. Außerdem boten sie den Kunden einen Einblick in die potentiellen Möglichkeiten der Software. Abschließend halfen die Proof–of–Concepts dabei, das Vertrauen der Kunden zu gewinnen und steigerte ihre Zufriedenheit im Laufe des Entwicklungsprozesses.

3.2 Agiler Ansatz

In den bisherigen Kapiteln wurde erläutert, dass eine enge Zusammenarbeit mit den Kunden gewünscht war und gefördert wurde. Enge Zusammenarbeit mit dem Kunden alleine reicht allerdings nicht, um eine agile Softwareentwicklung durchzuführen [2, 19]. Zu einer agilen Softwareentwicklung zählen außerdem Flexibilität im Bereich der Anforderungen und der Entwicklung, rege zwischenmenschliche Interaktionen während des Entwicklungsprozesses und eine funktionierende Software auszuliefern [2]. Die Vorteile der agilen Entwicklungsmethoden wurden bereits eingehend erläutert. In dieser Arbeit wird vor allem Wert auf die Kundenzufriedenheit gelegt. Dazu wurden vier Praktiken der agilen Softwareentwicklung eingesetzt. Dies sind die *User Stories* und *Story Cards* sowie das Verfahren der *inkrementellen Entwicklung* und Ansätze von *Scrum*. In den folgenden Abschnitten wird genauer auf die verwendeten Praktiken eingegangen und wie sie im Rahmen dieser Arbeit

eingesetzt wurden.

User Stories

User Stories sind eine erprobte Methode, um Anforderungen innerhalb eines Projektes aufzuzeichnen [21]. Mithilfe von User Stories werden Funktionen der Software beschrieben, die relevant für einen Benutzer oder eine Gruppe von Benutzern sind [6]. Der Umfang einer solchen User Story ist dabei klein und oberflächlich gehalten, weshalb technische Details ausgelassen werden [6]. Die Idee der User Story ist, dass sie mithilfe der Entwickler von den jeweiligen Benutzern geschrieben wird. Die Motivation dafür ist, dass die Benutzer selbst die gewünschten Funktionen am besten verstehen [6].

In dieser Arbeit besteht eine einzelne User Story aus drei Teilen: *Wer möchte die Funktion?*, *Welche Funktion ist gemeint?* und *Warum wird diese Funktion benötigt?* Mithilfe dieser Struktur gelingt eine genaue Nachvollziehbarkeit des Ursprungs einer Funktion und wieso diese gebraucht wird. Die User Stories werden auf sogenannte Story Cards geschrieben [6]. Diese wurden im Rahmen des durchgeführten *Planning Game* mit Story Points geschätzt und von den Kunden anschließend priorisiert. Mithilfe des Planning Game können die Stakeholder mit den Entwicklern zusammen die nächste Implementierungsphase gestalten [19]. Dabei werden die Funktionen, die auf den Story Cards stehen, ihrer Komplexität nach abgewogen und können anschließend von den Kunden entsprechend für die Implementierungsphase ausgewählt werden [19]. Die Priorisierung der Story Cards wurde mithilfe eines Kanban–Boards durchgeführt. In der Tabelle 3.3 sind beispielhaft zwei User Stories, die in der Arbeit verwendet wurden, abgebildet. Eine Liste aller dokumentierten User Stories wurde dem Anhang A.2 beigefügt.

[US01]	Als SE–Mitarbeiter möchte ich eine bestehende XML–Datei laden können, damit ich Projekt– und Kundeninformationen importieren kann.
[US02]	Als SE–Mitarbeiter möchte ich Konfigurationen vornehmen und einen Account einstellen können, damit die Projektdaten von dort heruntergeladen werden können.

Tabelle 3.3: Auszug der User Stories.

Kanban

Kanban ist eine weit verbreitete Methode der agilen Softwareentwicklung [30]. Mittels Kanban kann die Menge an Funktionen, die gleichzeitig bearbeitet werden, begrenzt werden [14, 30]. Außerdem wird die Methode dadurch charakterisiert, dass Arbeit nicht vergeben, sondern selbst genommen wird [14].

Das eingesetzte Kanban–Board war in drei Spalten aufgeteilt: *Unerledigt*, *In Bearbeitung* und *Fertig*. Darin einsortiert waren die jeweiligen Story Cards. Während des Verlaufes der Entwicklung wurden die Story Cards dann entsprechend in die Spalten eingeteilt. Das Kanban–Board wurde für die jeweiligen Implementierungsphasen auf 20 Story Points beschränkt. Da die Entwicklung schneller vorangeschritten ist als ursprünglich geschätzt, konnten allerdings, unter Absprache mit den Kunden, weitere Story Cards während der Implementierungsphasen hinzugenommen werden.

Scrum

Ein Framework, welches sich mit Kanban gut kombinieren lässt, ist Scrum. Das liegt daran, dass Scrum zwei Backlogs hat, auf denen User Stories priorisiert werden. Scrum sieht dabei einen Arbeitsablauf vor, in dem zwei Zyklen ineinandergreifen [19]. Mithilfe von Scrum kann die Produktentwicklung verwaltet werden [19]. Das Wichtige an Scrum sind allerdings die Praktiken [19]. Dabei gibt es einen sogenannten Product Backlog. Dieser ist eine priorisierte Warteschlange, die vom Product Owner verwaltet wird. Der Product Backlog umfasst dabei alle Funktionen, die der Auftraggeber umgesetzt haben möchte [19]. Von diesem werden sich zu Beginn eines Sprints einige der Funktionen, in Rücksichtnahme auf die vorangegangene Priorisierung, genommen und auf den sogenannten Sprint Backlog gelegt. Dieser wird über die Dauer eines Sprints von den Entwicklern abgearbeitet. Ein Sprint stellt dabei den ersten, großen Zyklus dar. Dieser dauert nach Koch [19] normalerweise dreißig Tage. Innerhalb dieses Sprints ist das Team selbstverwaltet und wird nicht von außen gestört. Des Weiteren wird jeden Tag ein *Daily–Scrum Meeting* durchgeführt. Dieses stellt den zweiten, inneren Zyklus dar. Dabei werden die wichtigsten Dinge des letzten Tages und die zukünftige Entwicklung besprochen. Die drei wichtigen Themen sind [19]: *Was wurde seit dem letzten Treffen gemacht?*, *Welche Probleme oder Schwierigkeiten gab es dabei?* und *Was wird bis zum nächsten Treffen unternommen?*

Scrum ist deutlich umfangreicher, als in dieser Arbeit beschrieben. Die genannten Praktiken sind jedoch der relevante Kern, die innerhalb der Entwicklung der Software durchgeführt wurden. Das Daily–Scrum Meeting wurde dabei umgestaltet zu einem Weekly–Scrum Meeting und der Product Backlog, der durch die Kunden priorisiert wurde, entsprach dem bereits beschriebenen Kanban–Board. Hinzu kommt, dass die Sprints dieser Arbeit eine Dauer von vierzehn Tage hatten, anstelle von dreißig Tagen, da nicht genügend Zeit für drei Sprints mit jeweils dreißig Tagen vorhanden war. Durch diesen gestaffelten Aufbau wurde den Kunden allerdings ermöglicht, einen regelmäßigen Blick in die Entwicklung der Software zu werfen und weitere Maßnahmen zu planen oder gegebenenfalls einzulenken.

Kapitel 4

Automatisierung des Prozesses

Der im Kapitel 1 angesprochene Prozess ist langwierig. Diesen Prozess automatisch auszuführen, ist die Hauptaufgabe der zu entwickelnden Software. Um diese Automatisierung implementieren zu können, wurde vorausgehend der manuelle Prozess auf potentielle Automatisierungsmöglichkeiten untersucht. Dieses Kapitel beschreibt den manuellen Prozess in seinen Feinheiten und erläutert die gefundenen Möglichkeiten zur Automatisierung. Auf Basis dieser wurde die Verhaltensweise der Software modelliert. Anschließend wird der neue, automatische Prozess vorgestellt und erklärt.

4.1 Der manuelle Prozess

Der manuelle Prozess besteht aus mehreren, einzelnen Schritten. Dazu zählt beispielsweise der Übertragungsprozess zwischen Gitlab und OwnCloud. Der Prozess ist langwierig, da jedes Projekt potentiell mehrere Teams haben kann. Der Prozess muss jedoch nicht nur auf Projekt– sondern auch auf Teamebene durchgeführt werden. Dieser Prozess wird durchgeführt, um den Kunden des Softwareprojektes die jeweiligen Projektergebnisse über E–Mail zuzustellen. Damit diese Links erzeugt werden können, müssen zunächst die richtigen Abgabebranches der Teams im Gitlab–Repository ausgewählt werden. Der Grund dafür ist, dass im Softwareprojekt der endgültige Abgabebranch den Studierenden nicht vorgegeben wird. Nachdem dies geschehen ist, müssen die Abgabebranches heruntergeladen werden, um diese in die OwnCloud hochladen zu können. Dies erlaubt eine Archivierung der Daten und die Erstellung der Links, auf welche die Kunden Zugriff erhalten werden. Diese Links müssen anschließend in die E–Mail eingefügt werden. Des Weiteren muss die E–Mail mit den korrekten Namen der Kunden versehen werden. In manchen Fällen gibt es Kunden, die persönlich angesprochen werden, da diese in persönlichem Kontakt mit dem Fachgebiet stehen. Damit die E–Mail sich natürlicher liest, werden deshalb die E–Mailtexte dieser Kunden speziell angepasst, indem die Nachricht mit persönlicher Anrede

verfasst wird. Die folgende Darstellung 4.1 soll den manuellen Prozess verständlich in einem Bild zusammenfassen.

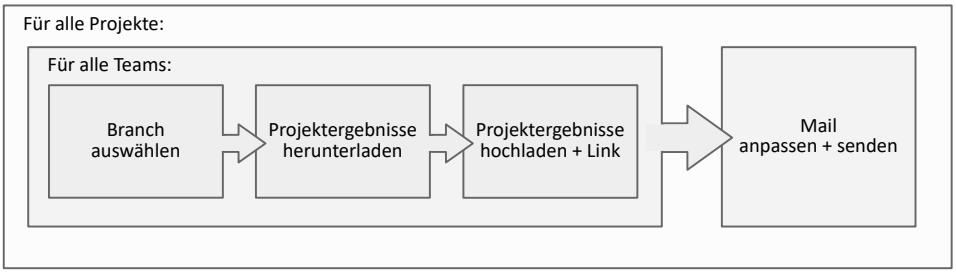


Abbildung 4.1: Darstellung des manuellen Prozesses.

Dieser Prozess kann mehrere Stunden dauern. Durch die repetitiven Vorgänge, beispielsweise beim Umschreiben der E–Mail oder den Hoch– und Herunterladevorgängen, können sich leicht menschliche Fehler einschleichen, da das Ausführen immer gleicher Arbeit zu Konzentrationsverlust führt [9]. In diesem Fall kann es passieren, dass die Kunden des Softwareprojektes nicht die korrekten Projektergebnisse erhalten oder der E–Mailtext grammatikalisch falsch geschrieben ist.

Automatisierungspotential

Durch die Automatisierung dieses Prozesses kann daher eine Maßnahme geschaffen werden, menschliche Fehler zu minimieren. Vor allem wird dadurch der zeitliche Aufwand des Verfahrens drastisch reduziert, da die Software alle Projekte gleichzeitig bearbeiten kann und nicht strikt an den sequentiellen Ablauf gebunden ist. Für die Organisation des Verlaufs dieser Arbeit und eine gute Kommunikation mit den Kunden, wurde zunächst herausgearbeitet, welche Teile des manuellen Prozesses automatisiert werden können. Anfänglich hat sich schnell gezeigt, dass die Automatisierung der Hoch– und Herunterladevorgänge viel Zeit sparen wird, da diese Teile des Gesamtprozesses besonders langwierig sind. Dies hängt damit zusammen, dass die Projekte bis zu mehreren Hunderten Megabyte groß werden können und die Gitlab–Seiten mit mindestens drei Ladezeiten navigiert werden müssen. Des Weiteren bot sich eine Automatisierung des Sendens der E–Mails an. Dies war ursprünglich kein Teil dieser Arbeit, wurde den Kunden jedoch vorgeschlagen, um eine vollständige und zufriedenstellende Nutzungserfahrung zu bieten. Das automatische Generieren der E–Mailtexte ist ein notwendiger Aspekt. Dies war für die Kunden von Beginn an eine Anforderung an das System. Diesbezüglich wurden mehrere Ansätze diskutiert. Einer dieser Ansätze war beispielsweise, zwei Texte vorzuschreiben, die dann

für die Kunden des Softwareprojektes als E–Mailtext entsprechend eingesetzt werden. Eine Automatisierung der Branchauswahl wurde ebenfalls als ein weiterer möglicher Aspekt identifiziert. Dadurch kann die manuelle Auswahl der Branches auf Gitlab für jedes Team eingespart werden. Zum Ende hin wurde in Betracht gezogen, den manuellen Prozess ganzheitlich durch einen einzelnen Knopfdruck zu automatisieren. Dadurch würde der stundenlange Prozess auf eine einzelne Eingabe reduziert werden.

4.2 Automatischer Prozess

Nach einer gründlichen Evaluierung der potentiellen Automatisierungsmöglichkeiten konnte der neue Prozess den Anforderungen entsprechend modelliert werden. Dabei wurde festgelegt, dass eine vollständige Automatisierung durch einen einzigen Knopfdruck nicht umgesetzt wird. Der Hauptgrund dafür liegt darin, dass eine solche vollständige Automatisierung dem Benutzer die Kontrolle über das System entziehen würde. Dieses Problem konnte durch eine Aufteilung des Prozesses in drei Unterprozesse gelöst werden. Durch die Aufteilung können die Kunden den Ablauf und Zustand des Prozesses besser einsehen und bei Problemen einlenken. So kann, durch geringfügig mehr Aufwand, der Prozess bedienungsfreundlicher gestaltet werden. Der erste Schritt dieses neuen Prozesses besteht in der automatischen Auswahl des Abgabebranches. Hierbei wird heuristisch der geeignete Abgabebranch für jedes Team über den Analytic Hierarchy Process ermittelt. Im nächsten Schritt erfolgt das automatische Hoch– und Herunterladen dieser Branches aus dem Gitlab in die OwnCloud. Schließlich erfolgt der letzte Schritt, bei dem die E–Mails versendet werden. Diese drei Funktionen sollen über jeweilige Auslöser separat steuerbar sein. Für Ausnahmen kann in diesen Prozess weiter eingegriffen werden, indem die Schritte ganz oder in Teilen manuell ausgeführt werden. Der automatische Prozess wird dadurch nicht gestört, vielmehr ist dies eine Erweiterung der Funktionsweise. Die Generierung der E–Mails soll ebenfalls automatisiert werden. Der Benutzer hat dazu die Möglichkeit, einen allgemeinen Text für alle Projekte vorzuschreiben. Beim Senden der E–Mail wird dieser den jeweiligen Projekten entsprechend angepasst. Die Abbildung 4.2 zeigt die Benutzerinteraktionen mit dem System für den automatischen Prozess.

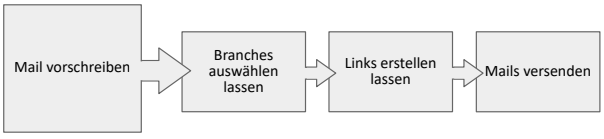


Abbildung 4.2: Darstellung der manuellen Eingaben im automatischen Prozess.

Durch den entwickelten Ablauf wird die Effizienz in Hinsicht auf die Ressource Zeit gesteigert und menschliche Fehler werden durch ihn minimiert, da die Anzahl repetitiver Aufgaben verringert wurde. Insgesamt bietet der Prozess eine Kombination aus Automatisierung und Benutzerkontrolle.

Kapitel 5

Implementierung

Der Kern dieser Arbeit liegt in der Implementierung der zu erstellenden Software. Dieses Kapitel wird sich mit den Einzelheiten eben dieser Implementierung befassen und dabei strukturiert auf einzelne Schritte und Entscheidungen eingehen.

5.1 Eingabedaten

Da die Software das letzte Glied einer Toolchain im Arbeitsablauf des Fachgebiets ist, müssen die Daten aus dem vorausgehendem Glied importiert werden. Weil dieses zur Speicherung von Daten das XML Format verwendet, wurde ein XML–Parser in die Software eingepflegt. Es konnte dazu der JSON–Java [1] Parser eingesetzt werden. Dieser erlaubt sowohl das Parsen von JSON als auch XML Daten. Da die Ausgabedaten des vorausgehenden Tools nicht alle notwendigen Informationen, wie beispielsweise die über das Geschlecht der Kunden des Softwareprojektes und die Art der Anrede, musste mit den Kunden eine leicht angepasste Version ausgehandelt werden. Eine vereinfachte Fassung davon befindet sich im Anhang A.4. Aus der angepassten Datei können die Projekte in der Software erstellt werden.

5.2 Architektur

Die Implementierung der Software erfolgte hauptsächlich unter Verwendung der Programmiersprache *Java*. Mithilfe verschiedener Design Patterns wurden das Grundgerüst und die Kernfunktionalitäten aufgebaut. Beispielsweise ist das Observer Pattern mehrfach für verschiedene Funktionen eingesetzt worden. Java erlaubt eine Plattformunabhängigkeit, die bei der Entwicklung der Software von Vorteil war, da die Entwicklung auf verschiedenen Systemen stattgefunden hat. Zusätzlich wurde der Python Interpreter *Jython* [2] in

[1] https://github.com/stleary/JSON-java
[2] https://www.jython.org/

die Software integriert, um das Laden von benutzerdefinierten Plugins zu ermöglichen. Einige spezifische Funktionen wurden bereits im Voraus als Plugins in Form von Python–Skripten implementiert, damit diese im Betrieb der Software leicht angepasst werden können. Ein Beispiel dafür sind die Parameter, die für die automatische Generierung des E–Mail Textes relevant sind. Weiter ist es möglich, Teilsysteme mittels der Plugins, in ihrer Gänze, beliebig zu ersetzen. Diese Teilsysteme sind der heuristische Auswahlvorgang, die APIs für Gitlab und OwnCloud, der Eingabe–Datei Importierer und der Exportierer für E–Mailtexte. Mithilfe des Benutzerhandbuches und den bereitgestellten Beispielen sollten die Anwender der Software dazu in der Lage sein, eigene Plugins zu schreiben, solange diese über die nötigen Python–Kenntnisse verfügen. Die Plugin–Architektur entstand aus dem Wunsch der Kunden heraus, eine noch höhere Anpass– und Wartbarkeit für die Software zu ermöglichen. Für die bereits oben genannten Module wurde demnach eine Schnittstelle errichtet, um diese entsprechend beeinflussen zu können. So können Änderungen der externen APIs später leicht behandelt werden, ohne dass eine umfangreiche Überarbeitung des gesamten Systems erforderlich ist. Dadurch bleibt die Software auch bei zukünftigen Änderungen des Umfelds und Weiterentwicklungen flexibel und nutzbar.

5.3 Automatisierung

Für die Implementierung der in Kapitel 4 besprochenen Automatisierungen kamen unterschiedliche Techniken, beispielsweise REST API–Zugriffe, zum Einsatz. Grund dafür sind die unterschiedlichen Systeme, mit denen die Teilprozesse interagieren und die verschiedenen Aufgaben, die sie erfüllen sollen. Für eine strukturierte Übersicht der Implementierung wird in diesem Teil auf die einzelnen Aspekte jeweils getrennt eingegangen.

Auswahl des Abgabebranches

Die Auswahl des Abgabebranches geschieht mithilfe des bereits beschriebenen Analytic Hierarchy Process. Für die bestmögliche Ermittlung eines Abgabebranches wurden vier Kriterien eines Branches untersucht. Dazu zählen: *Name des Branches*, *Aktualität des Branches*, *Anzahl der Commits* auf dem Branch und ob der Branch der *Default–Branch* ist. Die Aktualität bezieht sich dabei auf die Branches mit den jüngsten Commits. Um zu prüfen, ob die ausgewählten Kriterien in einem Zusammenhang mit der Auswahl des Abgabebranches stehen, wurden Hypothesen und Gegenhypothesen aufgestellt und mithilfe von Kontingenztabellen der Chi–Quadrat Test durchgeführt. Als Beispiel wurde die Hypothese geprüft, dass der Name eines Branches keinen Einfluss darauf hat, dass dieser als Abgabebranch fungiert. Für jedes Kriterium wurde in gleicher Form eine Hypothese und eine entsprechende Gegenhypothese aufgestellt. Als Vergleichswerte wurden

die tatsächlich ausgewählten Abgabebranches aus dem Wintersemester 2022/2023 verwendet. Es konnte mit dem Test festgestellt werden, dass die untersuchten Kriterien für die Wahl zum Abgabebranch relevant sind.

Damit die Auswahl der Abgabebranches durch einfach auswertbare Wahrheitswerte entschieden werden kann, werden die Kriterien zu leichter entscheidbaren Formulierungen geändert. Für bessere Verständlichkeit werden diese Kriterien in einen Vektor übertragen. Dieser stellt außerdem die Reihenfolge für die späteren Spalten und Zeilen der Matrizen dar, die im AHP benötigt werden. Wir nennen den Vektor A und die im Verlaufe der Berechnung entstehenden Matrizen folgen dem Schema $A \times A$.

$$A = \begin{pmatrix} \text{Branchame enthält „main”} \\ \text{Branch ist am aktuellsten} \\ \text{Branch hat meiste Commits} \\ \text{Branch ist „default”–Branch} \end{pmatrix} \tag{5.1}$$

Der erste Schritt ist nun, eine Vergleichsmatrix M_V aufzustellen. Dies wurde nach ausführlichen Abwägungen händisch vorgenommen. Diese Matrix muss nur einmal aufgestellt werden und ist dann allgemeingültig für alle Projekte. M_V beschreibt die paarweisen Beziehungen der Kriterien aus A. Gelesen wird die Matrix für einen Eintrag $v_{ij} \in M_V$ mit der Spalte $a_i \in A$ und der Zeile $a_j \in A$:

„Zeile a_j ist v_{ij} mal so wichtig wie Spalte a_i”

$$M_V = \begin{pmatrix} 1 & \frac{1}{3}* & \frac{1}{3} & \frac{1}{2} \\ 3 & 1 & 1 & \frac{3}{2} \\ 3 & 1 & 1 & \frac{3}{2} \\ 2 & \frac{2}{3} & \frac{2}{3} & 1 \end{pmatrix} \tag{5.2}$$

Aus der Matrix M_V ist ablesbar, dass alle Kriterien mit sich selbst gepaart den Wert 1 erhalten. Ein Zahlenwert von 1 bedeutet Gleichwertigkeit. Paare (a, a) mit $a \in A$ werden per Definition gleichwertig sein, da trivial ist, dass beispielsweise die *Aktualität des Branches* gleichwertig zu der *Aktualität des Branches* ist. Der Zahlenwert 2 hingegen entspricht einer leicht höheren Wichtigkeit von a im Paar (a, b) mit $a, b \in A$ wobei a das Kriterium der Zeile ist und b das der Spalte. Die jeweiligen transponierten Stellen in der Matrix entsprechen dem Kehrwert der nicht–transponierten Stelle. Dies ergibt sich daraus, dass das Kriterium a eine Wertigkeit x gegenüber von b hat. Andersherum hat b im Umkehrschluss eine Wertigkeit von $\frac{1}{x}$ gegenüber a. Als ein Beispiel liest sich die mit Stern $(*)$ markierte Zelle wie folgt:

> „Ob der Name eines Branches „main" enthält, ist $\frac{1}{3}$ mal so wichtig wie, als wenn der Branch der aktuellste Branch ist."

Ziel ist es, aus der Matrix M_V einen Prioritätsvektor zu berechnen. Dieser Prioritätsvektor sagt zum einen aus, wie die Gewichtung der einzelnen Kriterien untereinander ist und zum anderen gibt er eine Rechengrundlage für die Wahrscheinlichkeit der jeweiligen Branches, ein Abgabebranch zu sein. Um diesen Vektor zu erhalten, kann ein einfaches Verfahren verwendet werden, welches im Verlaufe dieses Kapitels beschrieben wird. Zunächst muss dazu die Matrix spaltenweise durch ihren Spaltensummenvektor normalisiert werden. Zu beachten ist dabei, dass mit Normalisierung nicht die Normierung des Vektors gemeint ist. Die Normierung eines Vektors erzeugt einen Vektor gleicher Richtung mit der Länge 1. Im AHP meint die Normalisierung, dass die Summe der Vektoreinträge 1 ergibt, welches nicht die gleiche mathematische Operation ist. Für diesen Vorgang wird der Spaltensummenvektor V_S gebildet:

$$V_S^T = (9,\ 3,\ 3,\ 4.5) \tag{5.3}$$

Für die spaltenweise Normalisierung werden die Einträge der Spalten von M_V nun mit dem entsprechenden Eintrag des Vektors V_S dividiert. Daraus resultiert die Matrix M_S. Aus ihren Zeilen kann nun ein (noch) nicht–normalisierter Prioritätsvektor V_{nP} gebildet werden. Dazu werden zeilenweise die Summen der Matrix M_S gebildet.

$$M_S = \begin{pmatrix} \frac{1}{9} & \frac{1}{6} & \frac{1}{9} & \frac{1}{9} \\ \frac{1}{3} & \frac{1}{3} & \frac{1}{3} & \frac{1}{3} \\ \frac{1}{3} & \frac{1}{3} & \frac{1}{3} & \frac{1}{3} \\ \frac{2}{9} & \frac{2}{9} & \frac{2}{9} & \frac{2}{9} \end{pmatrix} \tag{5.4}$$

$$V_{nP} = \begin{pmatrix} \frac{1}{2} \\ \frac{4}{3} \\ \frac{4}{3} \\ \frac{8}{9} \end{pmatrix} \tag{5.5}$$

Mittels des Vektors V_{nP} können bereits Schlüsse über die Gewichtung gezogen werden. Damit eine sinnhafte, prozentuale Bestimmung möglich wird, muss dieser allerdings ebenfalls noch normalisiert werden. Dazu wird die Summe aller Einträge aus V_{nP} gebildet. Die Summe wird dann als Divisor für den Vektor V_{nP} verwendet. Das Resultat der Rechnung ist

der normalisierte Prioritätsvektor V_P. Dieser kann nun für die Bewertung der Branches verwendet werden, um entsprechend einen Abgabebranch automatisch zu bestimmen.

$$V_P = \begin{pmatrix} 0,123 \\ 0,329 \\ 0,329 \\ 0,219 \end{pmatrix} \tag{5.6}$$

Mithilfe des Prioritätsvektors V_P ist eine Rangfolge der vier Kriterien erkennbar.

Aktuellster Branch $\equiv$ Meiste Commits $\succ$ Default $\succ$ Name enthält „main"

Der Vektor V_P kann nun auf jeden Branch eines Teams angewendet werden. Dazu wird überprüft, ob die jeweilige Bedingung eines Aspektes erfüllt ist. Bei Erfüllung der Bedingung wird die Wahrscheinlichkeit aus der entsprechenden Zeile des Aspektes im Vektor V_P zur gesamten Wahrscheinlichkeit für den Branch hinzugefügt. Schlussendlich wird der Branch mit der höchsten Wahrscheinlichkeit als Abgabebranch für das Team ausgewählt.

Implementierung

Der oben beschriebene Prozess wurde in der Implementierung zu einem Modul zusammengefasst. Dieses erhält die Matrix M_V und berechnet daraus den Prioritätsvektor V_P. Mittels öffentlicher Methoden kann eine Liste von Branches an das Modul übergeben werden. Das Modul berechnet dann eigenständig den wahrscheinlichsten Abgabebranch, im Vergleich zu den anderen Branches, die ebenfalls in der Liste waren. Da der Prozess modularisiert wurde, konnte dieser ebenfalls durch benutzerdefinierte Python–Skripte austauschbar gemacht werden. Dies erlaubt es den Anwendern, den Prozess beliebig anzupassen oder zu erneuern, falls dies gewünscht wird.

Projektergebnisse laden

Die Projektergebnisse müssen, wie bereits beschrieben, erst heruntergeladen werden. Diese befinden sich auf einem Gitlab–Server des Fachgebiets. Das Herunterladen kann daher auf einfache Weise mittels der REST API von Gitlab geschehen. Die API bietet dabei einen Endpunkt an, der das direkte Herunterladen beliebiger Branches erlaubt. Für die Weiterleitung an die Kunden des Softwareprojektes geschieht das Hochladen danach über die OwnCloud–Instanz des Fachgebiets. Dort können Dateien mittels eines

WebDAV Interfaces einfach hochgeladen werden. WebDAV ist dabei eine Erweiterung von HTTP und bildet das Dateisystem des Servers auf Adressen ab. Die Links, die in die E–Mails zum Teilen eingebettet werden sollen, können nach dem Hochladen über die API der OwnCloud erstellt werden.

Diese beiden Funktionalitäten wurden jeweils modularisiert und abstrahiert. Mittels der Abstraktion konnte sichergestellt werden, dass durch Python–Skripte die beiden Module durch beliebige andere Implementationen ersetzt werden können. Diese Implementationen können dann ebenfalls andere Dienste ansprechen und sind nicht an Gitlab oder OwnCloud gebunden.

E–Mailtexte generieren und senden

Für die Generierung der E–Mailtexte wurde eine Lösung ausgewählt, die nur wenig Implementationslogik benötigt, aber dennoch eine hohe Flexibilität für den Anwender bietet. Anstelle von zwei vorgeschriebenen E–Mailtexten, die in den beiden Fällen persönlich oder formell eingesetzt werden, wurde auf einen parametrisierten Ansatz zurückgegriffen. Dabei können in den E–Mailtext Parameter eingebunden werden, welche entsprechend beim Senden an die jeweiligen Softwareprojekt–Kunden automatisch angepasst werden. Mögliche Parameter umfassen dabei alle Possessiv–, Reflexiv– und Personalpronomen sowie Verben. Pronomen– und Verb–Parameter lassen sich über eine Konfigurationsdatei im Sinne der Wart– und Erweiterbarkeit vom Anwender, ohne weitere Programmierkenntnisse, hinzufügen. Im Folgenden wird ein Beispiel für eine Eingabe 5.1 sowie die danach angepasste Ausgabe 5.2 präsentiert. Es wird dabei angenommen, dass Franziska Mutig die Kundin auf formeller Ebene ist.

```
<Briefkopf>

heute möchte ich <personal_pronomen_dativ> die Projektergebnisse
aus dem Softwareprojekt übergeben. Dazu habe ich Downloadlinks
für <personal_pronomen_akkusativ> in diese E-Mail eingebunden.
Falls es noch Fragen gibt, <können> <personal_pronomen_nominativ>
<reflexiv_pronomen_akkusativ> gerne bei mir über E-Mail melden.

Hier sind die Links von <possesiv_pronomen_akkusativ> Teams:
<Downloadlinks>

<Abschlussformel>
```

Abbildung 5.1: Parametrisierter Eingabetext für den Mailtext–Parser.

```
Sehr geehrte Frau Mutig,

heute möchte ich Ihnen die Projektergebnisse aus dem
Softwareprojekt übergeben. Dazu habe ich Downloadlinks für
Sie in diese E-Mail eingebunden. Falls es noch Fragen gibt,
können Sie sich gerne bei mir über E-Mail melden.

Hier sind die Links von Ihren Teams:
Musterprojekt-1: https://../..
Musterprojekt-2: https://../..

Mit freundlichen Grüßen
Das SWP Team
```

Abbildung 5.2: Angepasster Ausgabetext für eine formelle Anrede im Singular.

Die Anpassung reagiert dabei auf vier verschiedene Fälle: *Persönliche Anrede im Singular*, *persönliche Anrede im Plural*, *formelle Anrede im Singular* und *formelle Anrede im Plural*. Im Beispiel 5.1 ist außerdem erkennbar, dass neben den angesprochenen Parametern für Pronomen und Verben auch Parameter für weitere Textblöcke existieren, beispielsweise der *Briefkopf* Parameter. Diese sogenannten Skript–Parameter lassen sich über Python–Skripte als Plugins einbinden und können beliebige Funktionen implementieren. Im Falle des Briefkopf–Parameters wird der ganze Briefkopf den Projektbedingungen entsprechend generiert.

5.4 User Interface

Im folgenden Abschnitt wird auf die implementierungsspezifischen Details des User Interfaces eingegangen und wie diese umgesetzt worden sind. Für ein einfaches Verständnis sollen diese mit Beispielen und Bildern untermauert werden. Der Abschnitt ist so strukturiert, dass zunächst die groben, oberflächlichen Konzepte erklärt werden, um dann immer tiefer in die Details einzusteigen.

Das User Interface wurde basierend auf der *Swing*–Bibliothek aufgebaut, welche in Java direkt verfügbar ist. Diese lies sich leicht an die spezifischen Designentscheidungen anpassen und konnte direkt im Code manipuliert werden. So konnten auf einfache Weise eigene Layouts und Komponenten erstellt und modularisiert werden. Dies hat dabei geholfen, den Code lesbar und entsprechend wartbar zu halten.

Umsetzung des Designs

Das Design wurde mithilfe der digitalen Papier–Prototypen und Mock–Ups erarbeitet, die bereits in Kapitel 3 beschrieben wurden. Da diese als Vision und Grundlage für den Aufbau der Software dienten, konnten manche Designkonzepte direkt übertragen werden. Dazu zählt beispielsweise die Projektleiste. Das entworfene Design und die konzeptionelle Funktionsweise, die aus den Mock–Ups heraus entwickelt wurden, haben sich als sehr geeignet erwiesen. Für die Software mussten fünf unterschiedliche Ansichten programmiert werden, um die Vision zu realisieren. Insbesondere sind dabei die *Projektübersicht*, die *Projektansicht* und die Ansicht zum *Schreiben von Mailtexten* von Bedeutung. Verbunden sind diese über eine Navigation in der Seitenleiste 5.3. Diese erlaubt eine einfache Bewegung zwischen der Projektübersicht und der Ansicht zum Schreiben von Mailtexten.

Abbildung 5.3: Die Navigations– bzw. Seitenleiste der Software.

Für eine schnelle Übersicht über alle Projekte, wurde die Projektübersicht entworfen. Sie erlaubt, alle Projekte in einer gelisteten Form zu betrachten. Von der Projektübersicht aus kann man auf die Projektansicht gelangen, indem die entsprechende Projektleiste aufgeklappt und dann mittels des Knopfes zur Projektansicht navigiert wird. Besonders macht die Projektübersicht, dass man die drei automatischen Schritte aus Kapitel 4 von hier aus auf alle Projekte gleichzeitig ausüben kann. Des Weiteren ist es dort möglich, Projektdaten und –zustände leicht zu überblicken. Damit der Fortschritt der Bearbeitung nicht verloren geht, kann auf dieser Ansicht außerdem die Projektmappe gespeichert werden.

Das Aufklappen der Projektleisten ermöglicht eine Vorschau des generierten Mailtextes, erkennbar in Abbildung 5.4. Für Informationen zum entsprechenden Projekt wird in jeder Projektleiste der ganze Zustand des Projektes angezeigt. Dies geschieht über die drei Statussymbole (1) *Branches ausgewählt*, (2) *Projektergebnisse herunter– und hochgeladen* und (3) *E–Mail gesendet*, die in der Projektleiste eingebettet sind. Die Statussymbole zeigen entsprechend der Abbildung 5.7 den Zustand des jeweiligen Schrittes an.

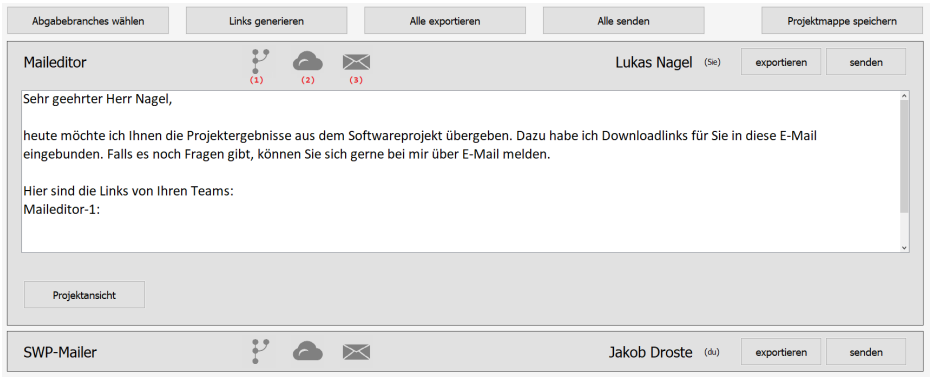


Abbildung 5.4: Übersicht über alle Projekte mithilfe der Projektübersicht.

Die Projektansicht, gezeigt in Abbildung 5.5, kann zur Kontrolle und Übersicht der automatisch gewählten Branches und generierten Links verwendet werden. Um eine bessere Fehlertoleranz und Steuerbarkeit zu gewährleisten, können die automatisch generierten Daten jedoch vom Benutzer nachträglich angepasst, gelöscht oder gar ganz manuell eingestellt werden. Sollte die Heuristik nicht die erwarteten Ergebnisse produzieren, kann an dieser Stelle manuell ein Branch gewählt werden. Diese Ansicht zeigt außerdem alle Teams des Projektes an und erlaubt, zwischen ihnen zu wechseln.

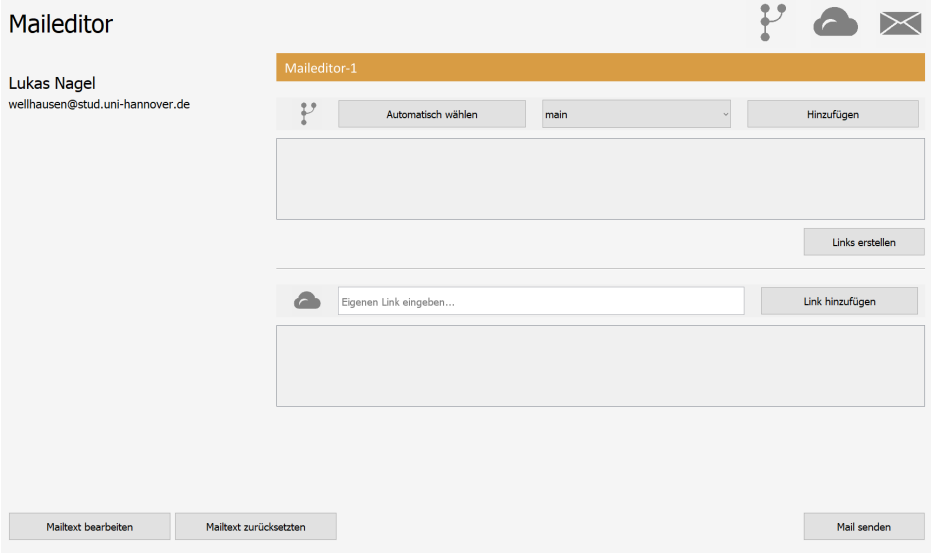


Abbildung 5.5: Detailansicht eines Projektes mit verschiedenen Steuerungsmöglichkeiten.

In der Projektansicht kann außerdem die Auswahl getroffen werden, dem jeweiligen Projekt einen speziellen Mailtext hinzuzufügen. Auch dies dient der Steuerbarkeit des Prozesses. So kann für ein oder mehrere Projekte jeweils ein spezieller Mailtext eingesetzt werden, falls ein allgemeiner Mailtext nicht ausreichend ist.

In der Ansicht zum Schreiben von Mailtexten, abgebildet in 5.6, liegt der Fokus auf dem Eingabefeld. In diesem kann der Mailtext geschrieben und bearbeitet werden. Parameter können dabei, sollte man diese auswendig kennen, per Hand geschrieben werden oder über die Leiste auf der rechten Seite eingefügt werden. Die Parameter sind für eine bessere Übersicht in zwei Klassen aufgeteilt: *Skript–Parameter* und *Pronomen–Parameter*.

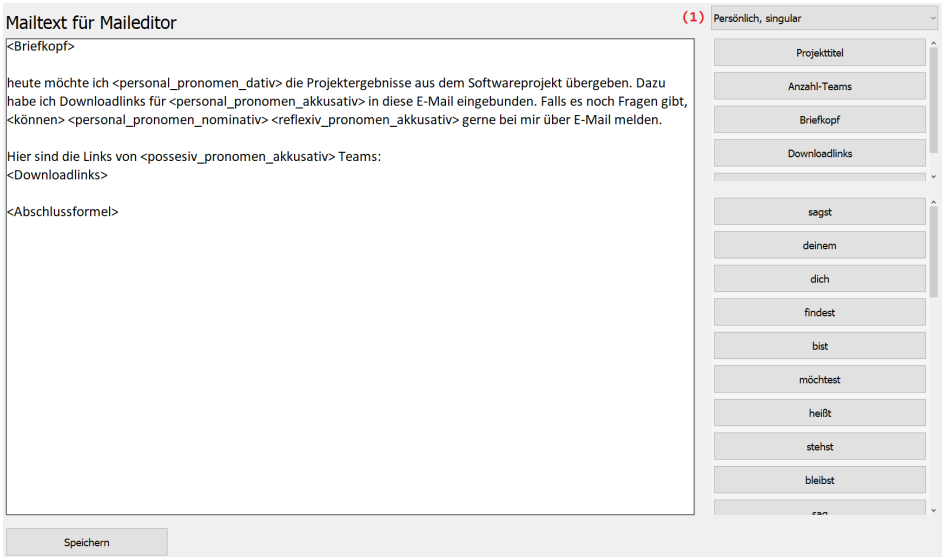


Abbildung 5.6: Ansicht zum Schreiben von Mailtexten.

Mittels eines Drop–Down Menüs (1) kann die Darstellungsart der *Pronomen–Parameter* in der rechten Parameter–Leiste angepasst werden. Dies ist eine wichtige Einstellung, da manche der Pronomen mehrfach belegt sind. So ist das Personalpronomen im Akkusativ der zweiten Person in Persönlich–Singular („*dich*") gleich dem Reflexivpronomen im Akkusativ der zweiten Person. Der Unterschied wird erst beim Wechsel der Form durch das Drop–Down Menü klar. Eine Anzeige aller Versionen der Pronomen auf den Knöpfen war nach Absprache mit den Kunden keine Option, da dies die Übersichtlichkeit zu sehr einschränkt. Die Bedienbarkeit der Software würde darunter erheblich leiden.

Statussymbole und Grafiken

Damit der Zustand der Software und der Projekte zu jeder Zeit eindeutig ablesbar ist, wurde ein spezifisches und leicht identifizierbares Schema für Grafiken und Symbole entwickelt. Jeder Schritt erhielt dabei ein eigenes Symbol, das den entsprechenden Zustand repräsentiert. Um die Unterscheidbarkeit der Symbole für Anwender mit Einschränkungen im Farbwahrnehmung zu gewährleisten, wurden neben der Farbkodierung auch verschiedene Symbolformen verwendet, damit diese weitere Unterscheidungsmerkmale erhalten. Die Symbole wurden in bis zu fünf separate Zustände unterteilt, um den aktuellen Status deutlich darzustellen. Diese Zustände sind wie folgt definiert: *Unversucht*, *In Bearbeitung*, *Erfolgreich*, *Warnung* und *Fehler*. Diese Aufteilung ist notwendig, da es in jedem der Teilschritte zu individuellen Fehlern und Problemen kommen kann. Für die Auswahl der Branches kann dies beispielsweise ein API Fehler sein. Analog ist es beim Symbol für das Übertragen der Projektdaten. In der Abbildung 5.7 sind die entsprechenden Symbole mit ihren jeweiligen Bedeutungen dargestellt. Durch die klare Visualisierung der Zustände soll die Transparenz und Nachvollziehbarkeit des gesamten Automatisierungsprozesses gewährleistet werden, um dem Ziel einer effektiven und reibungslosen Durchführung des Arbeitsprozesses gerecht zu werden.

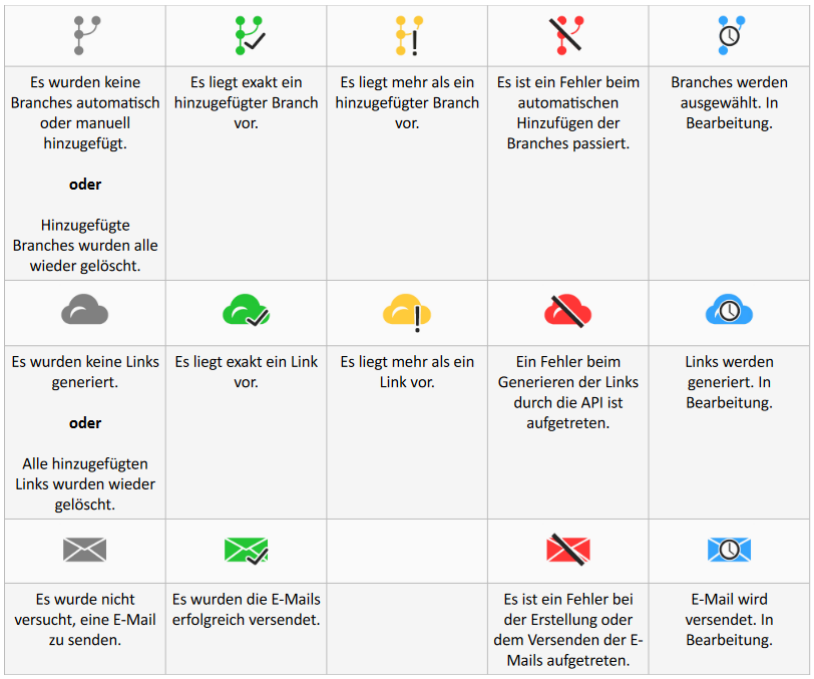


Abbildung 5.7: Statussymbole mit unterschiedlichen Formen und Farbkodierungen zu besseren Erkennbarkeit. Auszug aus dem Benutzerhandbuch.

5.5 Brücke zwischen Oberfläche und Geschäftslogik

Die Benutzeroberfläche muss auf Ergebnisse und Änderungen in der Geschäftslogik reagieren, beispielsweise nachdem die Links generiert wurden und das Statussymbol umstellen muss. Andererseits muss auch die Geschäftslogik ausgeführt werden, wenn Ereignisse, wie das Betätigen des „*Mail senden*" Knopfes, im User Interface auftreten. Zu diesem Zweck wurde ein single–threaded Event–System gebaut. Durch das Registrieren von *Listenern* können Ergebnisse und Ereignisse zentralisiert an alle Interessenten verteilt werden. Mithilfe dieser Architektur wird die Kopplung der einzelnen Module stark verringert. Dies hängt damit zusammen, dass die Module sich untereinander nicht mehr kennen und Instanzen voneinander besitzen müssen. Das Event–System besteht aus den in der Abbildung 5.7 eines UML Klassendiagramms abgebildeten Klassen. Als Beispiel wurde die Event–Klasse *MailtextChangedEvent* hinzugefügt, um die Funktionsweise und Vererbungshierarchie der Events zu verdeutlichen.

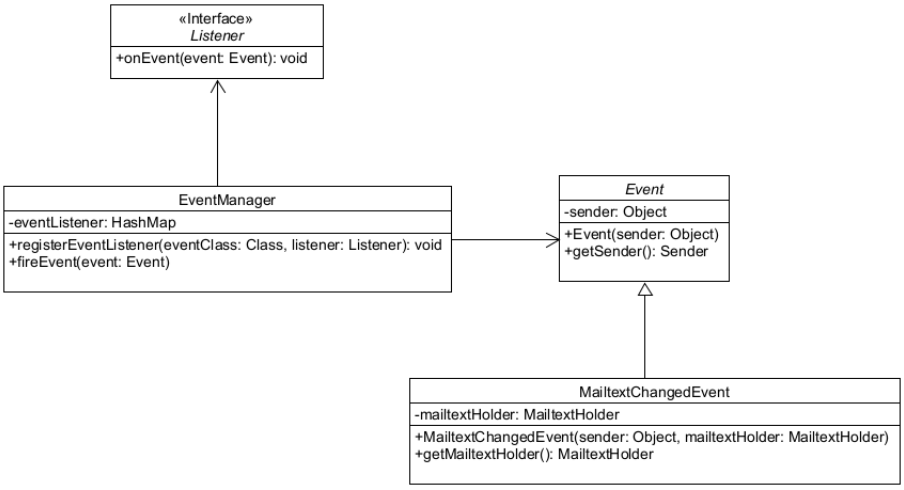


Abbildung 5.8: UML Klassendiagramm zur Veranschaulichung des Aufbaus des Eventsystems.

Events können demnach beliebig erstellt werden, indem sie von der abstrakten Klasse *Event* erben. Damit Events empfangen werden können, muss ein *Listener* beim *EventManager* registriert werden. Dazu muss außerdem die Klasse der Events, die für den Listener interessant sind, angegeben werden. Dies erlaubt eine schnelle Vorsortierung der Events und nur die interessierten Listener werden beim Auslösen der Events aufgerufen.

5.6 Konfigurationen

Die Software verfügt über 27 grundlegende Konfigurationsmöglichkeiten. Diese können vom Anwender beliebig im Rahmen ihrer jeweiligen Möglichkeiten angepasst werden. Dadurch erhält der Anwender einen hohen Grad der Freiheit in Bezug auf die Arbeitsweise der Software. Mithilfe der bereits angesprochenen Plugins können weitere Konfigurationen nach eigenem Ermessen hinzugefügt und speicherbar gemacht werden. Ein Beispiel für die Konfigurationsmöglichkeiten ist die *heuristic-undecidable-difference*. Auf Basis dieser Einstellung kann der Anwender die Heuristik steuern und ihr mitteilen, ab welchem Unterschiedlichkeitsgrad zwei Branches als gleichwertig wahrscheinlich gelten.

Die Konfigurationen können vom Anwender unkompliziert in einer Konfigurationsdatei vorgenommen werden. In dieser können, durch die Verwendung von Schlüssel–Wert–Paaren, den einzelnen Konfigurationen Werte zugewiesen werden. Diese werden zum Start des Programms geladen und entsprechend übernommen. Ausgewählte Konfigurationen können vom Anwender vor dem Start des Programms direkt manipuliert werden, da zuvor eine Konfigurationsübersicht angezeigt wird, in der grundlegende Einstellungen getätigt werden können. Darunter fällt beispielsweise das Übergeben von Passwörtern an die E–Mail Schnittstelle.

5.7 Schwierigkeiten bei der Implementierung

Während der Implementierung haben sich an manchen Stellen Schwierigkeiten gezeigt. Der erste Ansatz, die Geschäftslogik und das User Interface miteinander zu verknüpfen, basierte auf dem Model–View–Controller Pattern. Es zeigte sich jedoch schnell eine zu hohe Kopplung der einzelnen Module. Da das User Interface in einzelne, spezialisierte Module aufgeteilt ist, um einen hohen Grad der Kohäsion zu erreichen, mussten Abhängigkeiten teilweise sehr tief durchgereicht werden und die Zugehörigkeiten und Herkunft dieser wurde nach einiger Zeit sehr verworren. Deshalb wurde auf das Event–System umgerüstet, wodurch der Code des User Interfaces zu großen Teilen umstrukturiert werden musste. Die Umstellung hatte sehr gute Ergebnisse gezeigt und die Abhängigkeiten zwischen User Interface und Geschäftslogik konnten drastisch minimiert werden.

Des Weiteren traten Schwierigkeiten bei der Umsetzung der Plugins auf. Dort wurde zunächst ein auf Java basierender Ansatz gewählt. Dazu hätte von den Anwendern allerdings neuer Java–Code kompiliert werden müssen und die Einbindung dieser Plugins verlief mittels Java–Reflections. Da diese Variante zu kompliziert für den tatsächlichen Gebrauch war, wurde der Python–Interpreter in das System integriert. So konnten die Plugins stattdessen über Skripte eingebunden werden. Dazu musste allerdings das

Plugin–System vollständig neu aufgesetzt werden.

In den SWP–Projekten musste außerdem der interne Zustand der Teams bei einer Veränderung durch das User Interface in Echtzeit übernommen werden, damit der Gesamtzustand des Systems konsistent bleibt. Dazu wurde im ersten Versuch ein hierarchisches Oberserver–Pattern eingesetzt, welches ebenfalls zu einer starken Kopplung und Verworrenheit führte. Daher wurde auch hier schlussendlich das Event–System integriert, um Zustandsänderungen auf einfache Weise mittels asynchronen Events zu erlauben.

Kapitel 6

Qualitätssicherung

Um zu gewährleisten, dass das System den qualitativen Anforderungen der Kunden gerecht wird, wurden insbesondere zwei Maßnahmen durchgeführt: *Usabilitytests* und *Abnahmetests*. Diese spielen eine große Rolle bei der Bewertung und Überprüfung der Qualität des Systems und tragen maßgeblich zur Gewährleistung einer positiven Benutzererfahrung und einer funktionalen Vollständigkeit bei. Im folgenden Kapitel werden diese beiden Maßnahmen im Detail erläutert.

6.1 Usabilitytests

Usabilitytests sind ein wichtiger Bestandteil des Qualitätssicherungsprozesses, da diese darauf abzielen, die Benutzerfreundlichkeit und damit einhergehende Bedienbarkeit des Systems zu bewerten [12]. Die Bedienbarkeit der Software liegt den Kunden sehr am Herzen und ist eine wichtige Qualitätsanforderung im Entwicklungsprozess gewesen. Dies liegt daran, dass die Software im Regelfall nur ein Mal pro Jahr eingesetzt wird. Damit die Bedienung der Software daher nicht jedes Jahr erneut gelernt werden muss, ist eine intuitive und einfache Bedienbarkeit wichtig. Dazu wurden Usabilitytests durchgeführt. Die Tests wurden dabei nach dem Schema von Dumas [12] erstellt und umfassten sieben aufeinander aufbauende Szenarien. Jedes Szenario besteht demnach aus einem *Setup*, einer *Beschreibung des Szenarios* und der *auszuführenden Aufgabe*, wie in Abbildung 6.1 zu sehen ist. Während der Tests wurden die Teilnehmer gebeten, ihre Gedanken laut zu äußern, entsprechend der „*Think–Aloud*" Methode.

Szenario 1	Du möchtest für die zu sendenden E-Mails den Text generell halten, sodass du nicht für jedes Projekt eine einzelne E-Mail schreiben musst.
Aufgabe	Schreibe im Programm einen E-Mailtext, der für alle Projekte global gilt. Der Text sollte mindestens einen Briefkopf und die Abschlussformel enthalten.
Setup	Auf einem laufenden PC ist die SWP-Mailer Software installiert. Dieser PC ist mit dem Internet verbunden und es liegt die Projekt-XML Datei im Projektordner des Installationsverzeichnisses. Die darin vermerkten Gitlab-Projekte sind angelegt.

Abbildung 6.1: Auszug eines Szenarios der Usabilitytests.

Die Tests wurden dabei in zwei Phasen aufgeteilt. Die erste Phase diente der Findung von problematischen Designentscheidungen, welche die Bedienbarkeit einschränken oder verschlechtern. Die gefunden Designentscheidungen wurden vor dem Beginn der zweiten Phase entsprechend ausgebessert. In der zweiten Phase wurden die Verbesserungen dann auf die Probe gestellt. Dabei wurde geprüft, ob die vorgenommenen Änderungen tatsächlich zur Verbesserung der Bedienbarkeit beigetragen haben. Für die Versuchsdurchführung wurde der „Between–Group" Ansatz gewählt. Mit diesem Ansatz konnte für beide Phasen eine gleiche Grundlage gelegt werden, indem die Teilnehmer die Software nicht zuvor bedient haben. Die beiden Phasen mit den selben Teilnehmern durchzuführen, würde die Ergebnisse aus der zweiten Phase weniger zuverlässig machen, da die Anwender dann bereits den Arbeitsablauf kennen würden.

Ergebnisse

In der ersten Phase sind Bedienungsmuster bei den Teilnehmenden aufgefallen, welche nicht gefordert oder gar falsch waren. Dies umfasst beispielsweise das Navigieren auf die falsche Ansicht oder das Betätigen der falschen Knöpfe. Insbesondere aufgefallen ist, dass alle Teilnehmenden die Bearbeitungsseite des projektspezifischen Mailtextes mit der Bearbeitungsseite des globalen Mailtextes verwechselt haben. Des Weiteren konnte kein Teilnehmender die automatische Branchauswahl ohne Hilfe starten. Diese verbarg sich unter der Schaltfläche „Verarbeitung beginnen", was die Teilnehmenden verwirrt hat und eine Verknüpfung mit der Aufgabe nicht ermöglichte.

Maßnahmen und Nachprüfung

Für die zweite Phase wurde die angepasste Software verwendet. Die Ergebnisse vielen dabei deutlich besser aus. Als Maßnahme gegen die Verwechslung der Bearbeitungsseiten wurde eine Überschrift auf dieser Seite eingesetzt. Diese hat den Teilnehmenden den aktuellen Zustand mitgeteilt und es war besser nachvollziehbar, auf welchen Mailtext Einfluss genommen wird. Außerdem konnten die Teilnehmenden die Schaltfläche für die automatische

Branchauswahl besser finden, da diese zu „Abgabebranches auswählen" umbenannt wurde.

6.2 Abnahmetests

Nach Abschluss der Implementierungsphase wurden umfangreiche Abnahmetests durchgeführt, um den Kunden einen klaren Nachweis für die funktionelle Vollständigkeit der entwickelten Software zu liefern. Die Abnahmetests waren von großer Bedeutung, da sie sicherstellten, dass die Software alle wesentlichen Funktionen erfüllt, die von den Kunden erwartet wurden. Sie konzentrierten sich insbesondere auf die wichtigsten und grundlegendsten Funktionen der Software. Ein Beispiel für eine solche Funktion war die Generierung der Mailtexte und dass die automatische Anpassung der E–Mail an die jeweiligen Projekte reibungslos und korrekt ablief. Die Abbildung 6.2 zeigt einen der durchgeführten Abnahmetests. Diese wurden aus vorher angelegten Use Cases abgeleitet.

Setup	Auf dem Dateisystem existiert im aktuellen Arbeitsverzeichnis die Datei „demo.xml" und die Datei „demo.json". Das Programm wurde gestartet und die Konfigurationen sind vorgenommen worden. Aktuell befindet sich das Programm auf der Startsteite.
Eingaben	**Ausgaben**
Der Benutzer betätigt den Knopf „Importiere externe XML"	Das System zeigt einen Dateiauswahldialog an.
Der Benutzer wählt im Dialog die Datei „demo.xml" aus. Und betätigt „Öffnen".	Das System öffnet die Projektübericht mit den neuen, geladenen Projekten.
Der Benutzer betätigt den Knopf „Importiere externe XML"	Das System zeigt einen Dateiauswahldialog an.
Der Benutzer wählt im Dialog die Datei „demo.json" aus. Und betätigt „Öffnen".	Das System zeigt die Fehlermeldung „Dateityp nicht unterstütz oder fehlerhafte Datei"

Abbildung 6.2: Auszug eines Abnahmetestfalles.

Nach der Durchführung der Abnahmetests zeigten die Kunden sich zufrieden mit dem Entwicklungsstand der Software. Die Abnahmetests deckten allerdings nicht alle Funktionen ab. Insbesondere die zusätzlichen Funktionen, die nicht zur Hauptaufgabe gehörten, wurden in den Tests aus Zeitgründen vernachlässigt.

Kapitel 7

Fazit und Ausblick

7.1 Fazit

Das händische Umschreiben und Versenden von E–Mails an mehrere Adressaten ist ein langwieriger und fehleranfälliger Prozess. Durch die hohe Anzahl verschiedener Kunden wird dies im Softwareprojekt verstärkt. Der manuelle Abschlussprozess wurde in Hinsicht auf potentielle Automatisierungsmöglichkeiten untersucht. Aus der Untersuchung traten einige Stellen im Prozess hervor, die sich leicht durch Software automatisieren lassen. Beispielsweise lässt sich der Übertragungsmechanismus der Abgabebranches aus dem Gitlab in die OwnCloud leicht verselbstständigen. Durch die Implementierung einer unterstützenden Software lässt sich die Effizienz des Fachgebietes steigern. Im Rahmen der Entwicklung wurden gängige Praktiken des Software– und Requirements Engineering verwendet. Dazu zählen beispielsweise die Anforderungserhebung, die mithilfe von Interviews durchgeführt wurde, sowie der Einsatz agiler Praktiken. Die Interviews wurden mit den Auftraggebern der Software durchgeführt, da diese das System am besten verstehen. Es existieren neben den Auftraggebern noch weitere Stakeholder, wie die Kunden der Softwareprojekte. Allerdings interagieren diese nicht mit dem System. Die Durchführung eines gründlichen Requirements Engineering Prozesses hat im Entwicklungsprozess geholfen, die Anforderungen beherrschbar zu halten und schlussendlich einen Teil dazu beigetragen, dass die Software den Erwartungen und Wünschen der Auftraggeber entspricht.

Während der Implementierung wurde den Kunden regelmäßig eine funktionierende Version der Software zur Verfügung gestellt. In den wöchentlichen Scrum–Meetings war es ihnen möglich, dieses System im Beisein der Entwickler zu testen und gegebenenfalls in den Entwicklungsprozess einzulenken. In dieser Arbeit war das Ziel, die Auftraggeber in den Entwicklungsprozess bestmöglich mit einzubinden. Die Vorgehensweise wurde dazu im Kapitel 3 genau erläutert. Im Fokus standen dabei die eingesetzten Konzeptpapie-

re. Diese dienten der Interpretation der erhobenen Anforderungen und boten die Möglichkeit, Missverständnisse frühzeitig zu identifizieren. Aus diesen Konzeptpapieren wurden anschließend Mock–Ups und Prototypen entwickelt, welche bei der Validierung der verhandelten und dokumentierten Anforderungen halfen. Der manuelle Prozess wurde mit den Auftraggebern gemeinsam ausgearbeitet. In Kapitel 4 wurde erläutert, wie dieser genau abläuft und welche Chancen es gab, diesen zu gestalten. Es konnten dabei Aspekte, wie die Generierung und Versendung von E–Mails, mit hohem Potential für Automatisierung erkannt werden. In der Implementierung, die in Kapitel 5 beschrieben wird, waren die Ergebnisse des Requirements Engineering Prozesses, darunter vor allem die Konzeptpapiere, eine gute Unterstützung. An ihnen konnte sich die Implementierung entlang bewegen, ohne von den Wünschen der Auftraggeber abzuweichen. Zum Ende der Implementierung wurden qualitätssichernde Maßnahmen durchgeführt. In Kapitel 6 wurde gezeigt, dass mithilfe der Usabilitytests Mängel in der Bedienbarkeit behoben werden konnten. Wichtig ist dies für die Auftraggeber, da diese die Software nur ein Mal in jedem Jahr verwenden werden. Das gewählte Verfahren hat in dieser Arbeit zu einem hohen Maße zum Erfolg der Softwareentwicklung beigetragen. Es konnte eine umfangreiche Software entwickelt werden, die den Auftraggebern einen großen Teil der manuellen Arbeit abnimmt und die Dauer des Prozesses auf wenige Minuten herabsenkt.

7.2 Ausblick

Zum Zeitpunkt dieser Arbeit hat das entwickelte Tool genau den einen, beschriebenen Anwendungsfall. Allerdings ist es möglich, die Software auch auf andere Weisen zu verwenden. Wenn die Funktion für das Übertragen der Branches ignoriert wird, können auch andere E–Mails mit dem Programm versendet werden. Dies kann dabei helfen, Nachrichten an eine große Menge von verschiedenen Adressaten zu versenden. Dies ist auch ohne den Kontext Softwareprojekt eine interessante Möglichkeit für das Fachgebiet, die Software weiter zu verwenden und auszubauen. Außerdem wurde die Software mit einer Schnittstelle für Plugins ausgestattet. Diese erlaubt dem Fachgebiet, die Software beliebig zu erweitern.

Anhang A

Requirements Engineering

A.1 Anforderungen und User Stories

[R01]	Das Programm muss eine bestehende XML–Datei lesen können, um daraus Projekt– und Kundeninformationen abzuleiten.
[R02]	Ein SE–Mitarbeiter soll der Anwendung einen Gitlab–Server übergeben können.
[R03]	Das Programm muss aus dem konfiguriertem Gitlab–Server Projektergebnisse herunterladen können.
[R04]	Das Programm muss Downloadlinks zu den heruntergeladenen Projektergebnissen bereitstellen können.
[R05]	Ein SE–Mitarbeiter soll dem Programm einen vorgeschriebenen, parametrisierten E-Mail Text übergeben können.
[R06]	Das Programm muss aus einem parametrisiertem E–Mail Text für jedes Projekt einen angepassten, personalisierten E–Mail Text erstellen können.
[R07]	Das Programm muss alle erstellten Downloadlinks, dem SWP–Kunden zugeordnet, exportieren können.
[R08]	Das Programm sollte SWP–Kunden nur ihr eigenes Projekt als Link zusenden.
[R09]	Ein SE–Mitarbeiter soll dem Programm einen SMTP-Server übergeben können.
[R10]	Das Programm erlaubt das Versenden der E–Mails über einen eingestellten SMTP–Server.
[R11]	Das Programm muss selbstständig den vollständigsten und aktuellsten Branch zu jedem Projekt auswählen können.
[R12]	Das Programm soll Fortschritt im Rahmen einer Projektdatei speichern können.
[R13]	Ein SE–Mitarbeiter soll die Möglichkeit haben, den Programmablauf in Stages einzeln durchzuführen.

[R14]	Das Programm soll in der Projektübersicht eine Vorschau des Mailtextes anbieten.
[R15]	Das Programm soll eine Liste der nutzbaren Parameter für die Erstellung des Mailtextes anzeigen können.
[R16]	Das Programm soll leicht wartbar sein.
[R17]	Das Programm soll leicht bedienbar sein.
[R18]	Das Programm soll fehlertolerant sein.
[R19]	Ein SE–Mitarbeiter soll die Möglichkeit haben, selbständig einen oder mehrere Branches eines Teams auszuwählen.
[R20]	Ein SE–Mitarbeiter soll die Möglichkeit haben, selbstständig einen oder mehrere selbst erstellte Downloadlinks für ein Team hinzuzufügen.
[R21]	Das Programm soll dem Benutzer die Möglichkeit bieten, den angepassten Mailtext zu begutachten. bevor dieser abgesendet wird.
[R22]	Die Heuristik soll einstellbar sein.
[R23]	Wenn ein Warnung auftritt, soll man eine Meldung bekommen, woher die Warnung stammt.
[R24]	Built–in Parameter sollen leicht änderbar und hinzufügbar sein.

Tabelle A.1: Übersicht der erhobenen Anforderungen.

[US01]	Als SE–Mitarbeiter möchte ich eine bestehende XML–Datei laden können, damit ich Projekt– und Kundeninformationen importieren kann.
[US02]	Als SE–Mitarbeiter möchte ich Konfigurationen vornehmen und einen Account einstellen können, damit die Projektdaten von dort heruntergeladen werden können.
[US03]	Als SE–Mitarbeiter möchte ich eine Liste der geladenen Projekte einsehen können, damit ich eine Übersicht aller Projekte habe und einzelne Projekte ggf. bearbeiten kann.
[US04]	Als SE–Mitarbeiter möchte ich einen E–Mailtext global vorschreiben können, der für alle Projekte benutzt wird, damit dieser dann verwendet werden kann, um die E–Mails zu generieren.
[US05]	Als SE–Mitarbeiter möchte ich einzelne E–Mailtexte exportieren können, damit ich diese manuell absenden oder speichern kann.
[US06]	Als SE–Mitarbeiter möchte ich E-Mails zu Projekten einzeln versenden können, damit bei spezifischen Updates nicht alle Kunden von allen Projekten benachrichtigt werden.
[US07]	Als SE–Mitarbeiter möchte ich alle E–Mailtexte gleichzeitig exportieren können, damit ich eine gesicherte Kopie aller Texte habe.
[US08]	Als SE–Mitarbeiter möchte ich alle E-Mails gleichzeitig versenden können, damit ich für den generellen Fall nicht jedem einzeln eine E-Mail senden muss.
[US09]	Als SE–Mitarbeiter möchte ich parametrisierte E–Mailtexte vorschreiben können, indem ich die Parameter aus einer Liste auswählen kann, damit ich mir nicht alle Parameternamen einzeln merken muss.
[US10]	Als SE–Mitarbeiter möchte ich die Liste der Parameter nach einer beliebigen Anredeart (persönlich singular, persönlich plural, formell singular, formell plural) filtern können, damit es einfacher für mich ist, die richtigen Parameter auszuwählen.
[US11]	Als SE–Mitarbeiter möchte ich einen Tooltipp angezeigt bekommen, der mir eine beispielhafte Nutzung des Parameters anzeigt, damit ich nicht ausversehen eine falsche Pronomengruppe auswähle.
[US12]	Als SE–Mitarbeiter möchte ich den Fortschritt als Projekt speichern können, damit ich später weiter machen kann, wo ich aufgehört habe.
[US13]	Als SE–Mitarbeiter möchte ich eine vorhandene Projektdatei laden können, damit ich dort weiter machen kann, wo ich zuletzt aufgehört habe.

Tabelle A.2: Übersicht der erstellten User Stories.

A.2 Mock–Ups

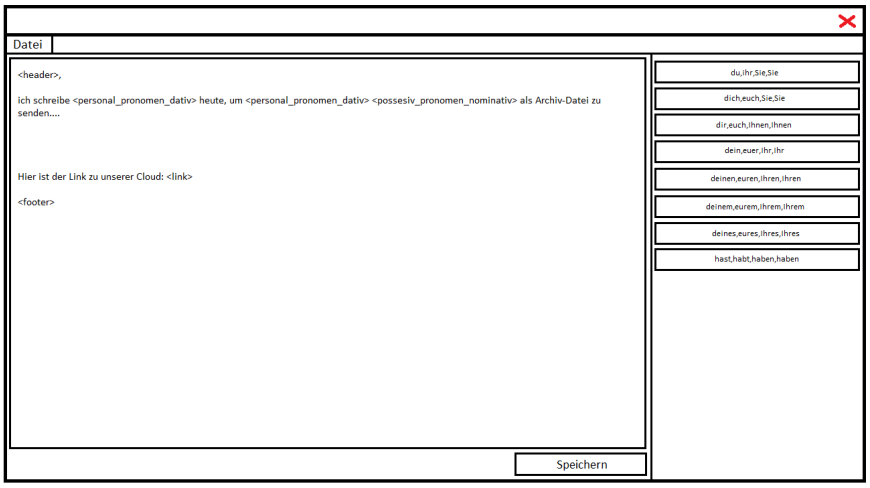


Abbildung A.1: Mock–Up des Mailtext–Editors.

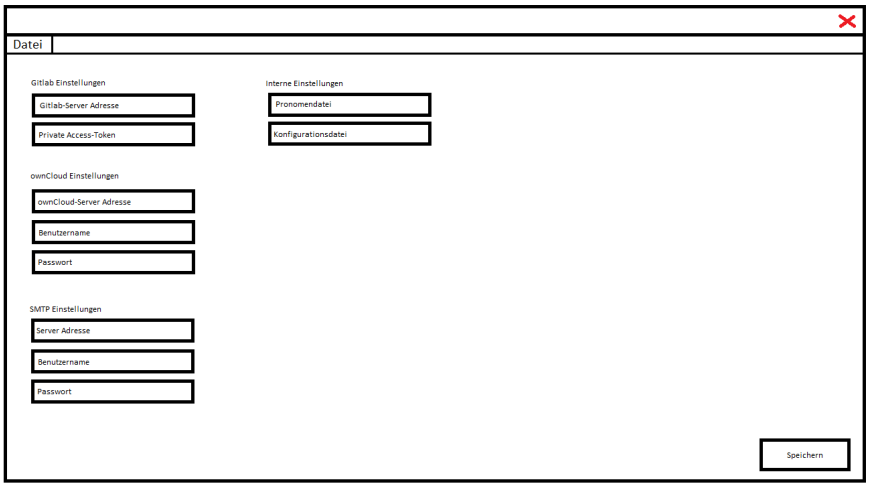


Abbildung A.2: Mock–Up des Konfigurationsfensters.

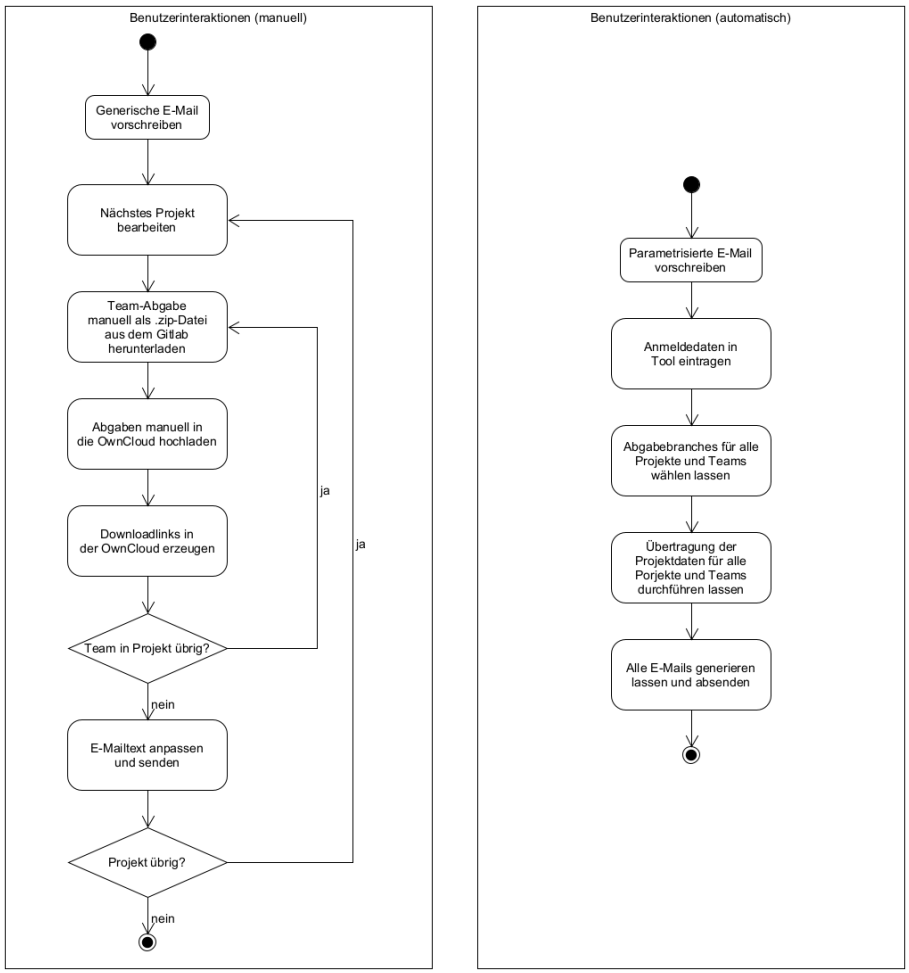


Abbildung A.3: Flussdiagramm des manuellen und des automatischen Prozesses.

```
<swp-solution name="">
    <description>
    </description>
    <customers count="3">
        <customer id="a9039ffc-24ca-4bd3-8b72-a3b13944a98f">
            <firstname>Franziska</firstname>
            <lastname>Mutig</lastname>
            <email>franziska.mutig@example.com</email>
            <formal>TRUE</formal>
            <gender>FEMALE</gender>
        </customer>
        <customer id="6703e691-cc43-4d5e-906b-507cf672cbbb">
            <firstname>Cedric</firstname>
            <lastname>Wellhausen</lastname>
            <email>whsn@stud.uni-hannover.de</email>
            <formal>FALSE</formal>
            <gender>MALE</gender>
        </customer>
    </customers>
    <projects count="3">
        <project id="138ed5e5-bac3-4be2-9c95-c6519060bb94">
            <name>SWP-Mailer</name>
            <unique-name>SWP-Mailer-1</unique-name>
            <customers count="2">
                <id>a9039ffc-24ca-4bd3-8b72-a3b13944a98f</id>
                <id>6703e691-cc43-4d5e-906b-507cf672cbbb</id>
            </customers>
        </project>
        <project id="9c77e141-7f01-42a2-8ca3-2521dcd2e06f">
            <name>SWP-Mailer</name>
            <unique-name>TMR-Optimizer-2</unique-name>
            <customers count="2">
                <id>a9039ffc-24ca-4bd3-8b72-a3b13944a98f</id>
                <id>6703e691-cc43-4d5e-906b-507cf672cbbb</id>
            </customers>
        </project>
        <project id="4ca510fb-8ca7-4459-9670-819c3265585d">
            <name>Mailtext-Editor</name>
            <unique-name>Mailtext-Editor-1</unique-name>
            <customers count="1">
                <id>6703e691-cc43-4d5e-906b-507cf672cbbb</id>
            </customers>
        </project>
    </projects>
</swp-solution>
```

Abbildung A.4: Beispiel einer möglichen Eingabedatei.

Anhang B

Digitale Daten

Für diese Arbeit sind weitere Anhänge interessant, die nicht textuell oder grafisch in dieses Dokument eingebunden werden können. Daher wird der Arbeit eine CD mitgeliefert, welche die folgenden zusätzlichen Anhänge umfasst:

- Die SWP–Mailer Software mit JRE.
- Eine Demo zum Testen der Software, mit simplen vorab durchgeführten Konfigurationen.
- Das Benutzerhandbuch für den SWP–Mailer.
- Digitale Papier–Prototypen.
- Eine PDF mit erstellten Use Cases.
- Alle angefertigten Konzeptpapiere in der aktuellsten Version.
- Die Ergebnisse aus den Usabilitytests.

Literaturverzeichnis

[1] Softwareprojekt. `https://www.pi.uni-hannover.de/de/se/lehre/swp/`, Sep 2022. Aufgerufen am: 19. Juni 2023.

[2] Agile Alliance. What is Agile? `https://www.agilealliance.org/agile101/`. Aufgerufen am: 24. Juni 2023.

[3] Alsaqqa, S. and Sawalha, S. and Abdel-Nabi, H. Agile Software Development: Methodologies and Trends. *International Journal of Interactive Mobile Technologies (iJIM)*, 14(11):246–270, Jul 2020.

[4] Aurum, A. and Wohlin, C. *Engineering and Managing Software Requirements*. Springer Science Business Media, 7 2005.

[5] M. Bruhn and K. Hadwich. *Dienstleistungen 4.0*. Springer Gabler, Jul 2017.

[6] Cohn, M. *User Stories*. Hüthig Jehle Rehm, Apr 2010.

[7] S. A. Conger. *The New Software Engineering*. Course Technology Press, Boston, MA, USA, 1st edition, 1993.

[8] M. Coronado and C. A. Iglesias. Task Automation Services: Automation for the Masses. *IEEE Internet Computing*, 20(1):52–58, 2016.

[9] Dadi, H. and Al-Haidous, M. and Radwi, N. and Ismail, L. Service Robots in Hospitals To Reduce Spreading of COVID-19. In *2021 Fifth World Conference on Smart Trends in Systems Security and Sustainability (WorldS4)*, pages 212–217, 2021.

[10] Davis, C. What If the Web Were Not RESTful? In *Proceedings of the Third International Workshop on RESTful Design*, WS-REST '12, pages 3—10, New York, NY, USA, 2012. Association for Computing Machinery.

[11] Dipsis, N. and Stathis, K. A RESTful middleware for AI controlled sensors, actuators and smart devices. *Journal of Ambient Intelligence and Humanized Computing*, 11(7):2963–2986, Jul 2020.

[12] J. S. Dumas and J. Redish. *A Practical Guide to Usability Testing.* Intellect Books, revised edition, 1999.

[13] Dumas, M. and La Rosa, M. and Mendling, J. and Reijers, H. A. *Fundamentals of Business Process Management.* Springer Berlin, Heidelberg, Jan 2018.

[14] Epping, T. *Kanban für die Softwareentwicklung.* Springer-Verlag, Aug 2011.

[15] Herrmann, M. and Klünder, J. From textual to verbal communication: towards applying sentiment analysis to a software project meeting. In *2021 IEEE 29th International Requirements Engineering Conference Workshops (REW)*, pages 371–376. IEEE, 2021.

[16] R. Ho and Y. Yen. Design and evaluation of an XML–based platform-independent computerized adaptive testing system. *IEEE Transactions on Education*, 48(2):230–237, 2005.

[17] Jha, S. and Ahmad, M. and Siddiqui, S. and Ahmed, J. and Agarwal, P. Implementation of IoT Frameworks in Business Process Automation. In *Proceedings of the 2nd International Conference on ICT for Digital, Smart, and Sustainable Development, ICIDSSD 2020, 27-28 February 2020, Jamia Hamdard, New Delhi, India*, 2021.

[18] Jørstad, I. and Bakken, E. and Johansen T. A. Performance evaluation of JSON and XML for data exchange in mobile services. In *Proceedings of the International Conference on Wireless Information Networks and Systems (ICETE 2008) - WINSYS*, pages 237–240. INSTICC, SciTePress, 2008.

[19] Koch, A. *Agile Software Development.* Artech House, 2004.

[20] Li, J. Analytic Hierarchy Process Automatically. In *2021 2nd International Conference on Artificial Intelligence and Information Systems*, ICAIIS 2021, New York, NY, USA, 2021. Association for Computing Machinery.

[21] Liskin, O. and Schneider, K. and Fagerholm, F. and Münch, J. Understanding the Role of Requirements Artifacts in Kanban. In *Proceedings of the 7th International Workshop on Cooperative and Human Aspects of Software Engineering*, CHASE 2014, pages 56—63, New York, NY, USA, 2014. Association for Computing Machinery.

[22] Lv, T. and Yan, P. and He, W. Survey on JSON Data Modelling. *Journal of Physics: Conference Series*, 1069(1):012101, Aug 2018.

[23] S. Roser. Modellgetriebene Geschäftsprozessautomatisierung. Technical Report 2005-09, Fakultät für Angewandte Informatik, 2005.

[24] Rupp, C. and Simon, M. and Hocker, F. Requirements Engineering und Management. *HMD Praxis der Wirtschaftsinformatik*, 46(3):94–103, Jun 2009.

[25] Saaty, T. L. and Vargas, L. *Models, Methods, Concepts & Applications of the Analytic Hierarchy Process*. Springer, Oct 2001.

[26] R. Schmelzer and T. VanDersypen. *XML and Web Services Unleashed*. Sams Publishing, 2002.

[27] C. G. Schuetz, B. Neumayr, and M. Schrefl. Multilevel modeling for business process automation. In *2015 IEEE 19th International Enterprise Distributed Object Computing Workshop*, pages 51–60, 2015.

[28] C. Severance. Discovering JavaScript Object Notation. *IEEE Computer*, 45(4):6–8, 2012.

[29] H. Sharp, A. Finkelstein, and G. Galal. Stakeholder identification in the requirements engineering process. In *Proceedings. Tenth International Workshop on Database and Expert Systems Applications. DEXA 99*, pages 387–391, 1999.

[30] Sjøberg, D. I. K. An Empirical Study of WIP in Kanban Teams. In *Proceedings of the 12th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement*, ESEM '18, New York, NY, USA, 2018. Association for Computing Machinery.

[31] B. Smith. *Beginning JSON*. Apress, 2015.

[32] J. W. Tukey. The Teaching of Concrete Mathematics. *The American Mathematical Monthly*, 65(1):1–9, 1958.

[33] A. van Lamsweerde. Requirements engineering: From craft to discipline. SIGSOFT '08/FSE-16, pages 238–249, New York, NY, USA, 2008. Association for Computing Machinery.

[34] Weidlich, M. and Grosskopf, A. and Lübke, D. and Schneider, K. and Knauss, E. and Singer, L. Verzahnung von Requirements Engineering und Geschäftsprozessdesign. In *Software Engineering 2009 - Workshopband*, pages 229–236. Gesellschaft für Informatik e.V., Bonn, 2009.

[35] Zhao, H. and Doshi, P. Towards Automated RESTful Web Service Composition. In *2009 IEEE International Conference on Web Services*, pages 189–196, 2009.