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

Spezifikation von Interaktionen mit graphischen Benutzerschnittstellen als Videos

Specification of GUI Interactions as Videos

Bachelorarbeit

im Studiengang Informatik

von

Lennart Glauer

Prüfer: Prof. Dr. Kurt Schneider
Zweitprüfer: Prof. Dr. Michael Rohs
Betreuer: M.Sc. Oliver Karras

Hannover, 22.03.2017

Erklärung der Selbstständigkeit

Hiermit versichere ich, dass ich die vorliegende Bachelorarbeit selbstständig 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 22.03.2017

Lennart Glauer

Zusammenfassung

Graphische Benutzeroberflächen bilden die Schnittstelle zwischen Benutzern und Softwaresystemen. Zu Beginn des Entwicklungsprozesses werden Skizzen der Benutzeroberfläche, sogenannte Mockups, erstellt. Sie dienen dem Zweck, die Wünsche und Bedürfnisse der Benutzer durch eine konkrete Veranschaulichung abzubilden und zu validieren. Auf diese Weise sollen die Nutzer bzw. Kunden bereits frühzeitig eine gute Vorstellung der geplanten Software erhalten, ohne dass die Entwickler dafür Quellcode schreiben müssen. In diesem Zusammenhang ist es sinnvoll, nicht nur das Aussehen der Benutzeroberfläche, sondern auch die Benutzung im Sinne eines *Usage-Centered Designs* zu spezifizieren. Eine herkömmliche Methode für die Dokumentation von Interaktionsabläufen (Szenarien) ist die Beschreibung des Ablaufs in Textform mit der Veranschaulichung der Benutzeroberfläche durch Mockups. Für die Dokumentation der Szenarien bietet sich zudem das Medium Video an, welches komplexe und interaktive Abläufe auf eine verständliche und nachvollziehbare Weise präsentieren kann.

In dieser Arbeit wird ein Softwareprototyp entwickelt, der es Requirements Engineers ermöglicht, gemeinsam mit den Endnutzern Mockups zu erstellen und mit diesen Interaktionsabläufe aufzunehmen, abzuspielen und im Verlauf des Entwicklungsprozesses mit geringem Aufwand zu bearbeiten. Die erstellten Szenarien sollen als Video dokumentiert und somit für die Anforderungsspezifikation der Software genutzt werden können. Das Ziel ist, die Erhebung, Dokumentation und Validierung von Anforderungen zu unterstützen und die Kommunikation zwischen Kunden, Requirements Engineers und Entwicklern zu verbessern. Die Dokumentation als Video soll es den Entwicklern ermöglichen, auch ohne direkten Kundenkontakt deren Ziele und Vorstellungen genau zu verstehen, damit sie in der Lage sind, eine Software zu implementieren, die den Wünschen und Bedürfnissen der Kunden entspricht.

Zusätzlich soll in dieser Arbeit mit einer Evaluation die Frage geklärt werden, ob Videos besser als Mockups zur Unterstützung der Beschreibung von Interaktionsabläufen in Textform geeignet sind.

Abstract

Graphical user interfaces provide the basis for interactions between users and software systems. At the beginning of the development process sketches of user interfaces (mockups) are created to describe and validate the goals and perceptions of end-users. In this way it is possible to present a vision of the planned software at an early stage of development without writing a single line of sourcecode. Not only the look of graphical user interfaces but also what the user experiences while using the software should be captured and validated following the *usage-centered design* approach. In particular the flow of user interactions (scenarios) can provide a better understanding of the software requirements for developers. A common form to document scenarios utilizes text and mockups to describe the flow of interactions. Videos can be a more appropriate way to document complex and interactive scenarios in an understandable manner.

The goal of this work is to develop a software prototype, that supports requirements engineers in creating mockups and capturing scenarios in close cooperation with end-users. Captured scenarios can be exported as videos to enhance the software requirements specification. The software prototype aims to support the elicitation, documentation and validation phases of the requirements engineering process and to improve the communication between stakeholders, requirements engineers and developers by using videos to document scenarios. Easily understandable video scenarios enable developers to implement software exactly like the user wants it even without direct exchange.

An evaluation tries to clarify the question if videos support the description of scenarios even better than mockups in addition to text.

Inhaltsverzeichnis

Kapitel 1

Einleitung

1.1 Motivation

In der heutigen Zeit sind Software-Systeme durch die Verbreitung von Smartphones, (Tablet-)PCs und vielen weiteren Produkten allgegenwärtig in unserem Leben vertreten. Mit der zunehmenden Präsenz steigen auch die Ansprüche und Erwartungen an die verwendete Software. Zudem muss diese vor dem Hintergrund der wachsenden Konkurrenz möglichst schnell und kostengünstig erstellt werden. Allerdings scheitert ein Großteil der Software-Projekte. Dem Chaos Report der Standish Group International zufolge ist dies zu einem hohem Anteil auf unzureichend erhobene Anforderungen zurückzuführen [22].

Im Bereich des Software Engineerings nimmt die Kommunikation zwischen den Beteiligten eine entscheidende Rolle ein [37], [19]. Hierzu gehören der Kunde, Requirements Engineers und das Entwicklerteam, welches sich unter anderem aus den Programmierern und den Userinterface Designern zusammensetzt. Die Requirements Engineers übernehmen dabei eine Vermittlerrolle zwischen dem Kunden und dem Entwicklerteam. Der Kunde hat spezifische Ziele und Vorstellungen für eine zu entwickelnde Software. Ihm fehlt es jedoch an der Sachkenntnis, um diese eigenständig umzusetzen. Er sucht sich deshalb einen Hersteller für Software mit der Absicht, ihn bei dieser Aufgabe professionell zu unterstützen. Für die erfolgreiche Umsetzung besteht die Grundvoraussetzung darin, dass der Requirements Engineer als ersten Schritt im Rahmen der Anforderungsanalyse die Vorstellungen des Kunden möglichst eindeutig erfassen und verstehen muss. Dies erfordert einen intensiven Austausch mit dem Kunden. Zu Beginn des Entwicklungsprozesses muss zunächst ein gemeinsames Verständnis aufgebaut werden, um Anforderungen an eine Software aufzustellen, die den Bedürfnissen und Wünschen des Kunden entspricht. Dieser Prozess ist nicht nur anspruchsvoll, sondern auch zeit- und kostenintensiv. Zudem ist dieser wichtige Teil der Softwareentwicklung anfällig für Fehler, alleine die Erhebung verursacht ca. 13% der Anforderungsfehler [3]. Rupp et al. [37] sehen die Hauptprobleme der Analyse in den unklaren Zielvorstellungen, der hohen Komplexität, sowie den Sprachbarrieren und damit bestehenden Kommunikationsproblemen zwischen den Beteiligten. Zusätzlich verkompliziert die Situation, dass es dem Kunden aufgrund fehlender Sachkenntnis schwer fällt, seine Vorstellungen

eindeutig auszudrücken [16]. Als Entwickler kann man nicht erwarten, dass der Kunde eine genaue Vorstellung des späteren Soll-Zustands der teils sehr komplexen zu entwickelnden Software hat. Insbesondere, da es sich bei Software um ein immaterielles Gut handelt, das mental nur schwer zu erfassen ist. Des Weiteren sind die technischen Lösungsmöglichkeiten, welche den Entwicklern zur Verfügung stehen, für den Kunden oftmals zu abstrakt und schwierig nachvollziehbar [14]. Auf der anderen Seite haben die Entwickler keine Kenntnis vom Arbeitsablauf und den Problemen des Kunden und verstehen daher nicht, was der Kunde sich wünscht bzw. benötigt [16]. Während der Entwicklung kann es zudem durch unterschiedliche Ansprechpartner auf allen Seiten zu Informationsverlusten kommen, so dass zum Beispiel bereits kommunizierte Wünsche des Kunden nicht umgesetzt werden. Außerdem ist es möglich, dass der Kunde bestimmte Wünsche erst in der Anwendungsphase des Programmes entwickelt. Hieraus ergibt sich die Forderung an den Requirements Engineer, möglichst frühzeitig im Entwicklungsprozess die Vorstellungen und Ziele des Anwenders zu erfassen. Dies bezieht sich insbesondere auch auf die geplante Verwendung der Software durch den Benutzer (vgl. *Usage-Centered Design* [11] und *User-Centered Design* [5]), um daraus konkrete Anforderungen an die Benutzeroberflächen und deren Interaktionsabläufe zu formulieren. Gleichermaßen ist es erforderlich, diese Anforderungen dem späteren Benutzer möglichst anschaulich zu präsentieren, um mögliche Missverständnisse aufzudecken und Änderungswünsche zu ermöglichen. Des Weiteren sollte es möglich sein, diese Änderungswünsche für alle Beteiligten gut verständlich dokumentieren zu können. Eine zuverlässige Dokumentation bzw. Spezifikation ist zudem erforderlich, da der Kunde häufig nur eine begrenzte Zeit für die Arbeit mit dem Requirements Engineer zur Verfügung hat und das Entwicklerteam diese als Grundlage für ihren Entwurf und die Implementierung der Software nutzt.

In der klassischen Softwareentwicklung erfolgt der Prozess der Anforderungsspezifikation größtenteils textbasiert unter Zuhilfenahme von sogenannten Mockups, welche die gewünschte Benutzeroberfläche skizzieren. Interaktionsabläufe werden hierbei als eine Abfolge von Mockups in Kombination mit textueller Beschreibung (z.B. in abstrakter Form als Use-Case) dokumentiert. Die Arbeit mit Text ist bei komplexen Inhalten und Interaktionen aufgrund möglicher Interpretationsspielräume verschiedener Personen fehleranfällig [31]. Dies ist insbesondere deshalb ein Problem, da viele Beteiligte aus dem Entwicklerteam nie direkten Kundenkontakt haben und somit auf die schriftliche Dokumentation angewiesen sind. Im späteren Prozess der Validierung von erhobenen Anforderungen ist ein Prototyp zur konkreten Präsentation für den Kunden sehr hilfreich [4]. Hierbei bieten sich Videos als Dokumentationsmedium an, da sie konkreter, einfacher und schneller für den Kunden zu verstehen sind [5], [12]. Zudem fordern Carter und Karatsolis [9] für eine robuste Dokumentation eine Ergänzung der Beschreibung als Text um andere Ausdrucksformen wie Video oder Audio. Auch Karras [25] sieht einen Vorteil darin, die Anforderungsspezifikation durch die Verwendung von Videos mit zusätzlichen Informationen anzureichern. Die Entwickler können nur eine zufriedenstellende Software implementieren, wenn die in der Spezifikation enthaltenen Anforderungen

von ihnen korrekt erfasst und verstanden werden [19]. Videos bieten sich an, um komplexe interaktive Inhalte – wie etwa Interaktionsabläufe mit graphischen Benutzerschnittstellen – auf eine verständliche Weise zu präsentieren [20]. Außerdem kann die Kommunikation von Anforderungen durch die Verwendung von Videos profitieren, da sie die enthaltenen Informationen auf nachvollziehbare Weise darstellen und dadurch ein besseres Verständnis ermöglichen [25], [4]. Somit bieten Videos zwar ein großes Potential, jedoch stellt der Aufwand zum Erstellen und zum nachträglichen Bearbeiten einen entscheidenden Nachteil für ihre Verwendung dar. Diesen gilt es zu minimieren, um den von Videos gebotenen Mehrwert bei gleichzeitig möglichst geringem Aufwand auszunutzen.

1.2 Ziel der Arbeit

Nachdem in der Motivation die Probleme der Anforderungsanalyse diskutiert wurden, werden im folgenden Abschnitt die Hauptziele der Arbeit definiert. Der Kunde bzw. der Benutzer der Software soll im Sinne des *Codevelompent and Coviewing* Konzeptes [5] stärker in den Entwicklungsprozess eingebunden werden. Insbesondere für den Entwurf der Benutzeroberfläche mit Hilfe von Mockups und das Spezifizieren der Interaktionsabläufe ist es sinnvoll, eng mit dem Endnutzer zusammenzuarbeiten, um die Software entsprechend seiner Bedürfnisse und Wünsche auszurichten [11]. Über die Interaktion mit Mockups kann mit ihm gemeinsam ein klarer Ablauf der geplanten Software erstellt werden. Hauptziel dieser Arbeit ist daher der Entwurf und die Entwicklung eines Softwareprototypen zum Aufzeichnen, Abspielen und Editieren von Interaktionsabläufen inklusive der Interaktionseingaben. Dieser soll den Prozess der Anforderungsanalyse und die Kommunikation mit dem Kunden unterstützen. Der Fokus liegt dabei auf Interaktionsabläufen (Szenarien), welche auf herkömmliche Weise in Form von Use-Cases dokumentiert und durch Mockups veranschaulicht werden. Der Prototyp soll die aufgezeichneten Szenarien zur Dokumentation und Verwendung in der Anforderungspezifikation als Video exportieren können. Ein nachträgliches Editieren der Szenarien – z.B. durch Anpassung der Mockups oder erneutes Aufzeichnen von Interaktion – muss mit geringem Aufwand möglich sein, damit diese während des Entwicklungsprozesses an die sich ändernden Wünsche der Benutzer angepasst werden können. Der Softwareprototyp soll zum einen die Kommunikation zwischen Kunden und Entwicklern erleichtern, zum anderen die Möglichkeit bieten, mittels realistischer Mockups bereits in einer frühen Phase der Entwicklung eine Vorstellung der Bedienung der geplanten Software zu erhalten. Mit dieser verbesserten Kommunikation können die Anforderungen leichter verstanden und nachvollzogen werden. Mittels des Softwareprototypen soll es daher möglich sein, verschiedene Arbeitsschritte der Anforderungsanalyse und auch der Entwicklungsarbeit zu vereinfachen:

1. Es sollen gemeinsam mit dem Kunden Mockups und Szenarien erstellt werden, welche im Verlauf des Entwicklungsprozess mit geringem Aufwand bearbeitet werden können. Die Szenarien sollen als Video dokumentiert und somit für die

Anforderungsspezifikation der Software genutzt werden können.

2. Zudem soll es möglich sein, das erstellte Video den der Anforderungsanalyse nachgelagerten Bereichen ohne direkten Kundenkontakt (Programmierer, Designer, etc.) zur Verfügung zu stellen, um die Kundenwünsche zu verdeutlichen und die Umsetzung zu vereinfachen.

Des Weiteren soll in dieser Arbeit die dem Prototypen zugrundeliegende Idee, zur Spezifikation von Interaktionen mit graphischen Benutzerschnittstellen das Medium Video zu verwenden, evaluiert werden. Die Evaluation soll untersuchen, ob die Hypothese zutrifft, dass Videos zu einem leichteren und schnelleren Verständnis von Szenarios beitragen. Es soll geklärt werden, ob Videos besser als Mockups zur Unterstützung der Beschreibung von Interaktionsabläufen in Textform geeignet sind.

1.3 Struktur der Arbeit

Diese Arbeit gliedert sich in sechs Kapitel, die im Folgenden kurz vorgestellt werden. Im ersten Kapitel soll zum Einstieg in die Fragestellung die Motivation und das Ziel der Arbeit erläutert werden. Hierbei geht es insbesondere um die Spezifikation von Interaktionen mit graphischen Benutzerschnittstellen und deren Dokumentation in Form von Videos. Ziel ist, den Prozess der Anforderungsanalyse und die Kommunikation zwischen Kunde, Requirements Engineer und Entwicklerteam zu unterstützen. Im zweiten Kapitel werden die Grundlagen der Anforderungsanalyse, des Anforderungsmanagements und der Anforderungskommunikation erläutert, sowie wichtige Begriffe und Definitionen eingeführt. Im Weiteren geht das Kapitel auf die Erstellung der Anforderungen ein und schließt mit dem Veranschaulichen in der Softwareentwicklung, dem Prototyping, ab. Im dritten Kapitel werden die Anforderungsspezifikationen für den Softwareprototypen erarbeitet. Ziel ist es, eine Software zu erstellen, mit der die in dieser Arbeit definierten Ziele erreicht werden können. Es werden die Anwendungsfälle, Rahmenbedingungen sowie funktionale und qualitative Anforderungen definiert. Im vierten Kapitel werden die Konzepte des Softwareprototyps erläutert. Hier werden die technischen Grundlagen und die Schnittstellen aufgezeigt. Außerdem wird die Graphische Benutzerschnittstelle und die Architektur hinter der Software vorgestellt. Der Abschnitt 4.5 „Verwandte Arbeiten“ gibt einen Einblick in die bestehende Literatur. Nachdem die Grundlagen, Anforderungsspezifikationen und Konzepte des Prototyps erläutert wurden, fokussiert sich das fünfte Kapitel auf die Evaluation von Videos im Vergleich zu Mockups als Unterstützung der textuellen Beschreibung von Szenarien. Dabei wird die Vorbereitung, Planung und Durchführung eines Experiments beschrieben. Am Ende dieses Kapitels werden Auswertung und finale Bewertung der erhobenen Daten betrachtet. Das sechste und letzte Kapitel führt über ein Fazit und einen Ausblick zum Abschluss der Arbeit. Hier werden im Rahmen einer kurzen Zusammenfassung der gesamten Arbeit die Ergebnisse diskutiert und abschließend Ideen für weiterführende Arbeiten präsentiert.

Kapitel 2

Grundlagen

Zu Beginn der Arbeit werden einige Grundlagen definiert, damit die späteren Aussagen eindeutig zu verstehen sind. Zunächst erfolgt die Definition des Begriffs des Requirements Engineering sowie die Erläuterung des zugehörigen Referenzmodells. Dabei wird auch auf den Begriff der Anforderungskommunikation eingegangen. Danach folgt die in dieser Arbeit verwendete Definition für Anforderungen, eine Übersicht von Qualitätskriterien sowie eine Erläuterung der Ziele und verwendeten Techniken zum Prototyping. Letztlich wird auf Szenarien eingegangen.

2.1 Requirements Engineering

Requirements Engineering (RE) beschreibt systematische Vorgehensweisen und Techniken zum Definieren, Dokumentieren und Verwalten von Anforderungen an Systeme [1], [2], [26]. In der Software-Entwicklung nimmt die exakte Erfassung der Anforderungen von Requirement Engineers einen äußerst wichtigen Stellenwert ein, da diese Bedürfnisse und Wünsche der Kunden dokumentieren und folglich entscheidend für den Erfolg der Software sind. Ein wichtiges Maß hierfür ist, wie gut die Software den angedachten Zweck ihrer Entwicklung erfüllt [30]. Schließlich verfolgt der Kunde mit der Entwicklung konkrete Ziele und Vorstellungen, wie z.B. das Vereinfachen eines Arbeitsablaufs. Die Requirement Engineers müssen diese Ziele erkennen, verstehen und analysieren, um daraus konkrete Anforderungen an z.B. die Benutzeroberfläche oder die Verwendung abzuleiten. Eine große Herausforderung stellt hierbei die Kommunikation der Kundenanforderungen und deren möglichst eindeutige Dokumentation dar. Das Erheben, Diskutieren und Verfeinern der Anforderungen kann als iterativer Prozess betrachtet werden, welcher schließlich in einer Anforderungsspezifikation mündet. Die gesammelten Anforderungen dienen den Entwicklern später als Grundlage für den Entwurf und die Implementierung. Darüber hinaus werden die spezifizierten Anforderungen auch nach Fertigstellung der Software weiter genutzt. Sie bilden über den gesamten Lebenszyklus des Produktes eine wichtige Grundlage für Wartung und Weiterentwicklung [37].

In dieser Arbeit wird die Definition für *Requirements Engineering* nach IEEE-24765 verwendet:

Definition 1. Requirements Engineering [1]

The science and discipline concerned with analyzing and documenting requirements

Im Folgenden wird der Prozess des Requirements Engineering im Kontext der Softwareentwicklung erläutert. Dafür wird der Prozess nach dem Requirements Engineering Referenzmodell (siehe Abbildung 2.1) in Abschnitte und einzelne Phasen gegliedert. Der in dieser Arbeit erstellte Prototyp soll primär drei Phasen der Anforderungsanalyse unterstützen (*Elicitation*, *Documentation* und *Verification & Validation*). Die Erläuterung erfolgt im Kapitel 3 „Anforderungsspezifikation".

Nach dem Referenzmodell wird der Prozess zunächst in zwei Abschnitte unterteilt, die **Anforderungsanalyse** und das **Anforderungsmanagement**.

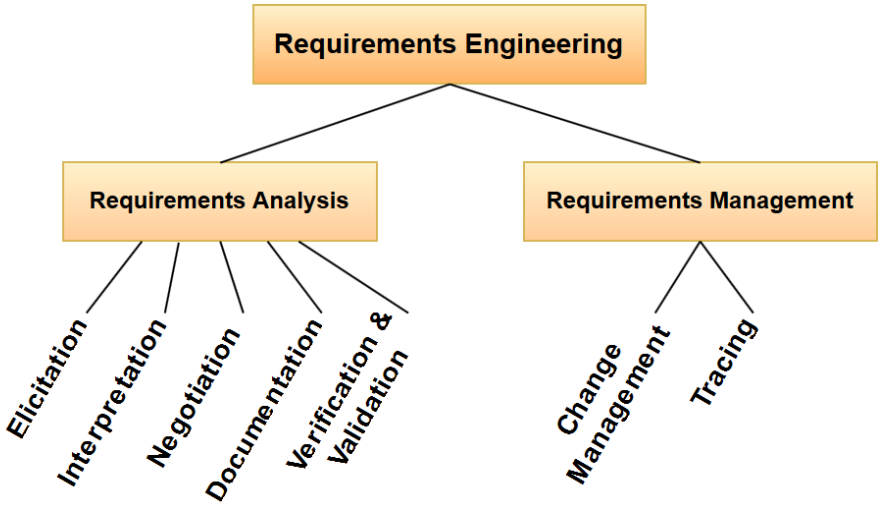

Abbildung 2.1: Requirements Engineering Referenz Modell [7]

2.1.1 Anforderungsanalyse

Die *Anforderungsanalyse* soll dazu dienen, Anforderungen zu spezifizieren, zu analysieren und ein gemeinsames Verständnis zwischen Kunden und Entwicklern zu bilden. Es ist wichtig, dass die Anforderungen für alle Beteiligten eindeutig zu verstehen sind. Auch der Kunde muss in der Lage sein, sich schnell einen Überblick über den aktuellen Stand der dokumentierten Anforderungen zu verschaffen. Dies ist sinnvoll, damit das Entwicklungsteam frühzeitig qualitatives Feedback vom Kunden erhalten und somit unvollständig oder fehlerhaft dokumentierte Anforderungen

erkennen und korrigieren kann. Die Anforderungsanalyse findet am Anfang des Entwicklungsprozesses statt und mündet in einer Anforderungsspezifikation, welche die Grundlage für die Konzeption, Implementierung und spätere Wartung der Software bildet. Sie hat einen erheblichen Wert für den Hersteller des Systems, da sie dokumentiert, *welche Funktionalität das System in welcher Qualität nach der Fertigstellung vorweisen muss* [38].

Definition 2. Requirements Analysis (dt. Anforderungsanalyse) [1]

(i) The process of studying user needs to arrive at a definition of system, hardware, or software requirements

(ii) The process of studying and refining system, hardware, or software requirements

(iii) A systematic investigation of user requirements to arrive at a definition of a system

Die Anforderungsanalyse lässt sich nach dem *Requirements Engineering Referenz Modell* (siehe Abbildung 2.1) in Phasen unterteilen. Diese bilden eine logische Abfolge von Schritten, welche für die Analyse und Dokumentation von Anforderungen notwendig sind. In der Praxis können diese Phasen – je nach verwendetem Entwicklungsmodell – sequenziell (linear), iterativ oder parallel ablaufen. Ein linearer Ablauf ist nur unter idealen Bedingungen möglich, da es hierbei zu keinen Verständnis- oder sonstigen Problemen in der Anforderungsanalyse kommen darf.

Die folgenden Erläuterungen der einzelnen Phasen des Requirements Engineering Referenz Modells basieren auf ISO/IEC/IEEE 29148:2011 [2], Schneider [39] und Rupp et al. [37].

Elicitation

Als *Elicitation* (dt. Erhebung) bezeichnet man die erste Phase der Anforderungsanalyse, in der es darum geht, Informationen vom Kunden zu sammeln und daraus *Rohanforderungen* zu bilden [39]. Der Begriff Rohanforderungen steht hierbei sinnbildlich für die ersten erhobenen Anforderungen, welche unstrukturiert sind und noch nicht den Qualitätskriterien für gute Anforderungen genügen (siehe Abschnitt 2.2.2 „Qualitätskriterien für Anforderungen“). Im ersten Schritt werden dafür die *Stakeholder* ermittelt, also die Personen bzw. Rollen, welche vom System in irgendeiner Weise betroffen sind und somit Einfluss auf das System haben. Interessant ist hierbei, zu ermitteln, in welcher Beziehung die Stakeholder zum System stehen und für wen daraus u.a. Vorteile oder eventuell auch Nachteile resultieren. Die unterschiedlichen Sichtweisen helfen den Requirements Engineers und Entwicklern dabei, das System besser durchdringen zu können. Außerdem sollen in dieser Phase weitere Quellen für Informationen identifiziert werden, wie z.B. bereits vorhandene Dokumente oder verwandte Systeme.
Das generelle Ziel ist es, möglichst viel Wissen über das System zu sammeln und erste Rohanforderungen zu erfassen. Hierfür muss das Gespräch mit den Stakeholdern gesucht werden, wozu als systematische Ansätze z.B. *Interviews* oder *Workshops* genutzt werden können [37], [39], [34]. Die Dokumentation der erhobenen Anforderungen kann auf unterschiedliche Arten erfolgen, oft verwendete Techniken sind unter anderem *Use-Cases*, *Storycards* oder *Fließtext* [37], [39].

Interpretation

Die unstrukturierten Anforderungen aus der Elicitation werden von den Requirements Engineers in der *Interpretationphase* in Beziehung zueinander gesetzt, klassifiziert und strukturiert. Kriterien hierfür sind beispielsweise Abhängigkeiten zu anderen notwendigen Systemkomponenten, logische Zusammengehörigkeiten oder die rollenbasierte Ansicht der Stakeholder [37]. Dieses Vorgehen soll die Menge der erhobenen Anforderungen übersichtlicher machen und das Verständnis für Abhängigkeiten oder andere Beziehungen zwischen den Anforderungen verbessern. Dadurch können doppelte und ähnliche Anforderungen erkannt und miteinander verschmolzen werden. Schwer verständliche oder unkonkrete Anforderungen müssen genau erläutert werden, um eindeutig verständliche Anforderungen zu erhalten. Das Ausformulieren unterstützt außerdem das Finden von Unklarheiten, welche durch Rücksprache mit den Stakeholdern beseitigt werden müssen. Das Ziel dieser Phase ist es, aus den unstrukturierten und abstrakten Anforderungen der Elicitation in der Interpretation geordnete und konkrete Anforderungen zu machen, um letztlich den Qualitätskriterien für gute Anforderungen zu genügen (siehe Abschnitt 2.2.2 „Qualitätskriterien für Anforderungen“). Stehen Anforderungen im Widerspruch zueinander, müssen diese mit den Stakeholdern besprochen und der entstandene Konflikt in der *Negotiationphase* aufgelöst werden [7], [39].

Negotiation

In der *Negotiation* (dt. Aushandlungsphase) sollen Anforderungskonflikte, wie etwa widersprüchliche Anforderungen verschiedener Stakeholder, identifiziert und aufgelöst werden [7], [39]. Dafür werden die gefundenen Konflikte den Stakeholdern präsentiert und im Gespräch eine für alle Beteiligten möglichst zufriedenstellende Lösung ausgehandelt. Das Ziel dieser Phase ist es, eine einheitliche Basis von Anforderungen zu bilden und bei der Zusammenführung auftretende Konflikte zu lösen.

Documentation

Die *Documentation* von Informationen und Anforderungen findet während des gesamten RE-Prozesses und insbesondere im Abschnitt der Anforderungsanalyse statt. In jeder Phase werden neue Informationen oder Erkenntnisse gesammelt, die sorgfältig dokumentiert werden müssen, um sie im weiteren Verlauf der Entwicklung wiederverwenden zu können. Das Ziel ist das Erstellen einer Spezifikation, welche das gesamte Wissen über das System dokumentiert. Dabei handelt es sich um einen iterativen Prozess, in dem die Spezifikation stetig erweitert und verfeinert wird. Ändern sich die Anforderungen, müssen diese Änderungen auch in die Spezifikation integriert werden.

Verification & Validation

In der *Verifikations- und Validierungsphase* werden die dokumentierten Anforderungen inhaltlich überprüft. Die Verifikation vergleicht dabei die Anforderungen mit den ursprünglich in der Elicitation erhobenen Rohanforderungen und zugehörigen Informationen. Eventuelle Abweichungen müssen mit den Stakeholdern besprochen, geklärt und anschließend nachvollziehbar dokumentiert werden. Diese Überprüfung soll unter anderem sicherstellen, dass alle Informationen aus den Rohanforderungen in die dokumentierten Anforderungen eingeflossen sind und nichts vergessen wurde. Die Validierung der Anforderungen soll sicherstellen, dass die abschließend dokumentierten Anforderungen die Ziele und Vorstellungen der Stakeholder auch korrekt erfassen. Dies ist wichtig, da die bisherige Spezifikation durch Probleme bei der Anforderungsanalyse fehlerhaft sein kann und die tatsächlichen Kundenwünsche nicht vollständig wiedergibt. Ein Fehler steht hierbei für eine Abweichung der dokumentierten Spezifikation (*Ist-Zustand*) von den Wünschen und Bedürfnissen der Stakeholder (*Soll-Zustand*). Da die Kosten für Änderungen an der Software im Verlauf des Entwicklungsprozesses stetig steigen (siehe Abbildung 2.2), ist es wichtig, dass fehlerhafte bzw. ungenaue Anforderungen frühzeitig vor dem Entwurf und der Entwicklung durch das Überprüfen der dokumentierten Anforderungen entdeckt werden. Dies soll die Anzahl der notwendigen Änderungen nach der Anforderungsanalyse möglichst gering halten. Man möchte – mit anderen Worten ausgedrückt – eine belastbare Spezifikation erhalten.

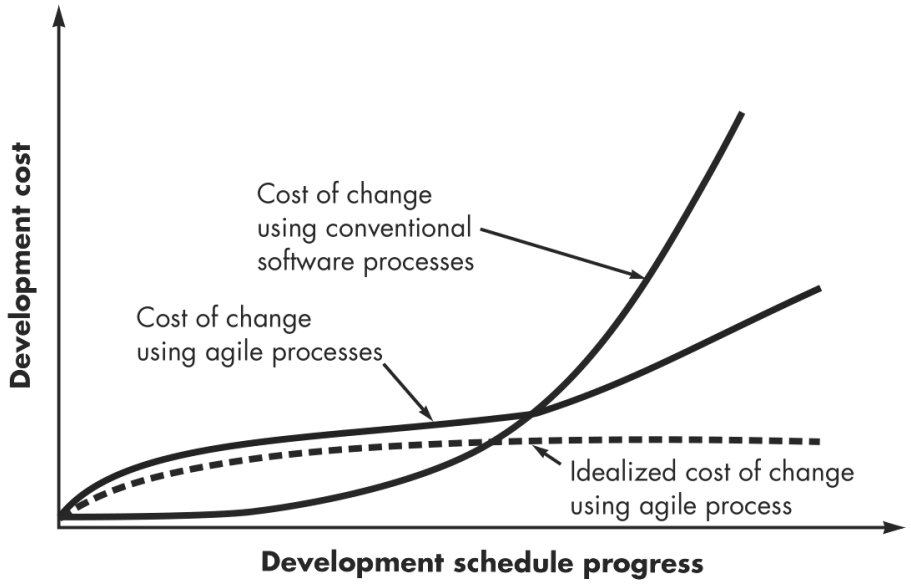

Abbildung 2.2: Kosten für Änderungen in der Softwareentwicklung als eine Funktion über die Zeit [32]

2.1.2 Anforderungsmanagement

Nach dem Referenzmodell (siehe Abbildung 2.1) folgt auf die Erstellung der Anforderungsspezifikation das *Anforderungsmanagement*. Da Software ein immaterielles Gut darstellt, können sich Kunden zu Beginn des Entwicklungsprozesses oft keinen konkreten *Soll-Zustand* vorstellen. Aus diesem Grund muss man davon ausgehen, dass sich Anforderungen auch nach der Spezifikation noch ändern können. Nach Rupp et al. [37] ist dies eine wesentliche Annahme, die auf jedes Projekt zutrifft. Somit ist es wichtig, Anforderungen während des gesamten Lebenszyklus einer Software zu verwalten und Änderungen zu dokumentieren, damit der aktuelle Stand zu jedem Zeitpunkt klar ersichtlich und nachvollziehbar bleibt. Da die Anforderungen auch nach Auslieferung der Software weiter verwendet werden – entweder zur Fehlerbehebung oder Weiterentwicklung – müssen die spezifizierten Anforderungen und deren Dokumentation bzw. die der Software zugrundeliegenden Konzepte in diesen Prozess miteinbezogen und beachtet werden.

Definition 3. Requirements Management (dt. Anforderungsverwaltung) [2]

Activities that ensure requirements are identified, documented, maintained, communicated and traced throughout the life cycle of a system, product, or service

Change Management

Das *Change Management* (dt. Änderungsverwaltung) beschreibt einen Prozess, welcher dazu dienen soll, die aus der Requirements Analysis resultierende Spezifikation während des gesamten Lebenszyklus der Software zu verwalten. Spätere Änderungen an der Spezifikation müssen sorgfältig geprüft, dokumentiert und durchgeführt werden. Der Prozess beginnt mit einem *Change Request*, welcher die gewünschte Änderung beschreibt und alle notwendigen Informationen bereitstellt. Von der Anfrage betroffene Abhängigkeiten müssen dabei beachtet werden. Diese Änderungsanfrage wird einem Gremium, dem sogenannten *Change Control Board* vorgelegt, welches die Anfrage bewertet und schließlich die Entscheidung trifft, ob die Änderung umgesetzt werden soll. Zum Change Management gehört anschließend die Planung und Durchführung der autorisierten Änderungen, damit der laufende Betrieb nur so kurz wie möglich unterbrochen wird.

Tracing

Das *Tracing* (dt. Rückverfolgung) soll dazu dienen, während des Projektverlaufs dokumentierte Anforderungen und weitere Informationen mittels *Trace Links* miteinander zu verknüpfen, um so den Erstell- und Änderungsprozess nachvollziehbar zu machen. Auf diese Weise entsteht ein Graph, welcher mit Methoden der Graphentheorie analysiert werden kann, um die Qualität des Entwicklungsprojektes zu bewerten [33]. Damit soll jeder Beteiligte in der Lage sein, die Anforderungen und Informationen bis zu ihrer Quelle in der Elicitation zurückverfolgen zu können (*Pre Tracing*). Außerdem sollen Abhängigkeiten zwischen den Systemkomponenten gefunden werden, damit ihre Einflüsse auf das zu entwickelnde System beurteilt werden können (*Post Tracing*). Dies ist wichtig, um ein besseres Verständnis zu erhalten und fundierte Entscheidungen beim Entwurf und der Implementierung treffen zu können.

2.1.3 Anforderungskommunikation

Ein wichtiger Bereich des Requirements Engineerings ist die notwendige Kommunikation der Beteiligten zur Erhebung und Vermittlung von Anforderungen. An komplexen Softwareprojekten arbeitet oft eine große Gruppe von u.a. Requirements Engineers, Entwicklern, Usability-Experten und Designern. Die nachgelagerten Bereiche für die Entwicklung, also z.B. die Entwickler und Designer, können häufig nicht direkt mit den Stakeholdern kommunizieren. Stattdessen dienen ein oder mehrere Requirements Engineers als Vermittler zwischen den Stakeholdern und den nachgelagerten Bereichen, welche die von den Requirements Engineers erhobenen Anforderungen als Grundlage für ihren Entwurf und die Implementierung nehmen. Damit die Anforderungen der Stakeholder für alle verständlich sind, müssen sie von den Requirements Engineers exakt dokumentiert werden. Die Dokumentation muss den beteiligten Teams möglichst einfach und direkt das notwendige Verständnis vermitteln, um zu gewährleisten, dass auch ohne direkten Kundenkontakt deren

Wünsche und Bedürfnisse unmissverständlich nachvollziehbar sind. Die Anforderungskommunikation beschäftigt sich mit dem Vermitteln von Anforderungen der Stakeholder an die Requirements Engineers und das Entwicklerteam, mit dem Ziel eine Software zu entwerfen und zu entwickeln, die den Funktionalitäts- und Qualitätsanspruchen der Stakeholder genügt. Die folgende Definition nach Fricker [18] lässt sich auf den Bereich des Software Engineering übertragen, wenn bei der Formulierung die Begriffe *Customer* durch *Stakeholder* und *Supplier* durch *Entwicklungsteam* ersetzt werden [19].

Definition 4. Requirements Communication (dt. Anforderungskommunikation) [18]

The process of conveying needs from a given customer to a given supplier who enables the latter to implement a solution that is accepted by the former

Um eine möglichst einfache und unmissverständliche Kommunikation zwischen Stakeholder und dem Entwicklungsteam zu ermöglichen, nimmt die Wahl des Mediums eine besondere Bedeutung ein. Die Kommunikation der beiden wird insbesondere erschwert, da sie nicht über einen einheitlichen Kenntnisstand verfügen. Auf der einen Seite befindet sich der Stakeholder, der konkrete Ziele verfolgt, ohne jedoch über fundierte Kenntnisse der möglichen Umsetzung zu verfügen. Auf der anderen Seite steht der Entwickler, der die professionellen Kenntnisse besitzt, jedoch nicht den Arbeitsablauf und Probleme des Kunden kennt und daher nicht versteht, was der Stakeholder sich wünscht bzw. benötigt (vgl. *Symmetry of Ignorance* [16]). Das für die Kommunikation und Dokumentation von Anforderungen gewählte Medium muss für beide Seiten verständlich sein, um zwischen ihnen zu vermitteln und eine gemeinsame Ebene der Kommunikation zu finden. Nur mit einem gemeinsamen Verständnis kann eine Software entwickelt werden, welche den Ansprüchen der Stakeholder genügt. Die These der vorliegenden Arbeit ist, dass für die Verdeutlichung von Interaktionsabläufen mit graphischen Benutzerschnittstellen das Medium Video gut geeignet ist (siehe Abschnitt 1.1 „Motivation“).

2.2 Anforderungen

Nachdem der Prozess des Requirement Engineerings betrachtet wurde, soll in diesem Abschnitt der Begriff der Anforderungen definiert werden, wobei hier die Unterteilung in die zwei Arten der funktionalen und nicht-funktionalen Anforderungen genauer erläutert wird. Um im späteren Verlauf die Qualität von dokumentierten Anforderungen miteinander vergleichen zu können, werden anschließend Qualitätskriterien für Anforderungen betrachtet.

2.2.1 Definition

Anforderungen bilden den Kern des Requirement Engineering und somit auch jedes Softwareentwicklungprozesses, da sie die Basis für den Entwurf und die Implementierung darstellen. Die Ziele und Vorstellungen der Stakeholder werden von den Requirements Engineers erhoben und in Form von Anforderungen dokumentiert. Allgemein stellen diese dabei eine Aussage der Stakeholder über die zum Erreichen eines Systemziels benötigte Funktionalität bzw. qualitative Merkmale der Software dar. Eine genauere Beschreibung bietet die Definition nach IEEE-24765, welche in dieser Arbeit verwendet wird.

Definition 5. Requirement (dt. Anforderung) [1]

(i) A condition or capability needed by a user to solve a problem or achieve an objective

(ii) A condition or capability that must be met or possessed by a system, system component, product, or service to satisfy an agreement, standard, specification, or other formally imposed documents

(iii) A documented representation of a condition or capability as in (i) or (ii)

Anforderungen lassen sich auf unterschiedliche Arten klassifizieren. Die wohl am häufigsten verwendete Unterteilung ist die in *funktionale* und *nicht-funktionale* Anforderungen.

Funktionale Anforderungen legen dabei fest, welche Fähigkeiten das System haben muss, um die Bedürfnisse der Benutzer erfüllen zu können, d.h. sie legen fest, was das Produkt tun soll [34].

Nicht-funktionale Anforderungen dagegen stellen Ansprüche an die Qualität, also die Art und Weise der Umsetzung, d.h. sie legen fest, wie gut ein System eine Leistung erbringen soll. Sie können aber auch Einschränkungen (Constraints) beinhalten, welche ein spezielles Vorgehen fordern, bestimmte Lösungsansätze ausschließen oder andere Randbedingungen ausdrücken.

Nach Fricker [19], Schneider [39] und auch Rupp et al. [37] ist die Förderung der Kommunikation zwischen Stakeholdern, Requirements Engineers und Entwicklerteam ein wichtiges Ziel von Anforderungen. Sie sollen zur Diskussion anregen und so helfen, zu einem besseren Verständnis aller Beteiligten beizutragen.

2.2.2 Qualitätskriterien für Anforderungen

Um die Qualität von Anforderungen und der daraus resultierenden Spezifikation beurteilen zu können, werden als nächstes Qualitätskriterien für Anforderungen betrachtet. Diese Erläuterungen basieren auf ISO/IEC/IEEE 29148:2011 [2], Schneider [39] und Rupp et al. [37]. Für ausgezeichnete Anforderungen müssen alle folgenden Kriterien erfüllt sein.

- **Vollständigkeit**
  Anforderungen müssen alle nötigen Informationen enthalten, um die gewünschten Funktionen und Eigenschaften des Kunden vollständig zu beschreiben. Es darf keine impliziten Annahmen seitens des Entwicklerteams über das System geben.

- **Eindeutigkeit**
  Anforderungen müssen präzise formuliert sein, damit sie eindeutig zu verstehen sind und somit das Risiko für Missverständnisse minimiert wird.

- **Notwendigkeit**
  Anforderungen beschreiben Funktionalität, Eigenschaften oder Qualitätsmerkmale, welche vom Kunden gefordert werden. Eine Anforderung ist nur dann notwendig, wenn sie der Erfüllung mindestens eines dieser Ziele dient.

- **Korrektheit**
  Anforderungen müssen den Kundenwunsch korrekt wiedergeben.

- **Prüfbarkeit**
  Der durch die Anforderung geforderte Soll-Zustand muss messbar sein und somit überprüft werden können.

- **Verständlichkeit für Stakeholder**
  Der Stakeholder muss Anforderungen mit geringem Aufwand verstehen und sich einen Überblick verschaffen können.

- **Verfolgbarkeit**
  Jede Funktionalität des Systems muss über die Anforderung bis zu ihrer Quelle (Stakeholder) zurückverfolgt werden können. Des Weiteren muss überprüfbar sein, ob eine Anforderung vollständig erfüllt wird.

- **Aktualität**
  Anforderungen müssen stets aktuelle Informationen enthalten. Veraltete Informationen müssen als solche markiert oder entfernt werden.

- **Juristisch Verbindlichkeit**
  Die juristische Verbindlichkeit muss für jede Anforderung geklärt werden. Daraus folgen die minimalen Anforderungen, welche ein System nach der Fertigstellung aufgrund des Kaufvertrags erfüllen muss.

2.3 Prototyping

Im Allgemeinen bezeichnet *Prototyping* eine Methode, die verwendet wird, um mit geringem Zeit- und Geld-Aufwand anhand eines konkreten Beispiels einen Teilbereich eines Systems zu veranschaulichen [35]. Prototyping eignet sich besonders, um Kommunikation zu fördern, ein gemeinsames Verständnis zu erzeugen und frühzeitig Feedback von Stakeholdern für ein zu entwickelndes System zu erhalten. Im Bereich der Softwareenwicklung werden Prototyping-Methoden oft für den Entwurf von graphischen Benutzerschnittstellen verwendet [12]. Die Benutzeroberfläche kann dadurch bereits zu Beginn des Entwicklungsprozesses entworfen und den Stakeholdern präsentiert werden. Im Allgemeinen kann Prototyping als iterativer Prozess betrachtet werden, der aus den Design, Implement und Analyze Phasen (DIA-Cycle) [35] besteht.

Houde und Hill [24] haben Prototypen nach den Gründen für ihre Erstellung klassifiziert. Demnach gibt es im Wesentlichen drei Kategorien: *Role*, *Implementation* und *Look and feel* Prototypen. Role bezieht sich hierbei auf die Fragestellung, welche Aufgabe bzw. Funktion für den späteren Benutzer erfüllt werden soll. Implementation Prototypen beziehen sich auf die genaue Umsetzung einer Problemstellung und dienen somit als ein *Proof-of-Concept*. Look and Feel Prototypen sind für diese Arbeit am interessantesten, da sie auf die Frage eingehen, welche Erfahrungen und Eindrücke ein Benutzer während der Verwendung des finalen Systems sammelt.

Es gibt verschiedene Prototyping-Methoden, welche jeweils bestimmte Vor- und Nachteile haben. Am Anfang des Prozesses eignet sich besonders gut das *Paper-Prototyping*, welches einfache Skizzen auf Papier nutzt, um schnell und mit geringen Kosten erste Ergebnisse zu erhalten. Ein Nachteil von Paper-Prototyping ist jedoch, dass nur schwer Interaktionen beschrieben werden können. Hierfür kann die Methode des *Storyboard-Prototyping* verwendet werden, welche Abläufe durch eine Folge von Bildern mit kurzer Beschreibung darstellt. Besser eignet sich jedoch die Methode des *Video-Prototyping*, die das Aufzeichnen und Abspielen von Interaktionen erlaubt und somit schnell ein gemeinsames Verständnis aufbauen kann. Allerdings besteht hierbei der generelle Nachteil von Videos, dass die Bearbeitung nur mit relativ hohen Aufwand möglich ist. Zum vermitteln des "Look and Feel" einer Benutzeroberfläche können *Facade-Prototypen* eingesetzt werden, welche durch die Verwendung von üblichen Bedienelementen bereits wie fertige Anwendungen aussehen. In digitalen Versionen lassen sich die Bedienelemente bereits wie gewohnt verwenden. Damit können einfache Abläufe durchgespielt werden, ohne Quellcode für die Anwendungslogik schreiben zu müssen. Eine konkrete Form der Facade-Prototypen sind *User Interface Builder*, die meist Bestandteil einer *Integrated Development Environment* (IDE) sind. Beispiele hierfür sind *Apple's Interface Builder*[1] oder der *Gluon SceneBuilder*[2]. Sie bieten den zusätztlichen Vorteil, dass die erstellten Benutzeroberflächen zur Veranschaulichung dienen, jedoch auch für die finale Anwendung weiterverwendet werden können.

[1] `https://developer.apple.com/xcode/interface-builder/`
[2] `https://gluonhq.com/products/scene-builder`

Mockups

Für diese Arbeit sind speziell *Mockups* von Bedeutung. Nach der IEEE Definition [1] beschreibt der Begriff Mockup im Allgemeinen ein Wegwerfprodukt. Bezogen auf die Softwareentwicklung bezeichnen Mockups einen Wegwerfprototypen, der die graphische Benutzerschnittstelle einer zu entwickelnden Software anhand üblicher Bedienelemente veranschaulichen soll. Sie werden insbesondere in frühen Entwicklungsphasen eingesetzt, um Anforderungen an Benutzeroberflächen in enger Zusammenarbeit mit Stakeholdern zu ermitteln oder Feedback zu erhalten. Oftmals gehen Mockups über einfache Skizzen hinaus und bieten zum Beispiel durch die Verknüpfung von mehreren Mockups die Möglichkeit Abläufe durchzuspielen (vgl. Facade-Prototyp).

2.4 Szenarien

Szenarien fassen mehrere einzelne Interaktionen zwischen dem Benutzer und einem Softwaresystem zu einem konkreten Ablauf zusammen. Nach Carroll [8] verfolgt jeder Ablauf ein bestimmtes Ziel und darf keine Entscheidungsmöglichkeiten (Verzweigungen) enthalten. Für optionale bzw. alternative Interaktionen werden jeweils eigene Szenarien erstellt, damit der einzelne Ablauf linear und deterministisch bleibt. Dies sorgt dafür, dass keine unnötigen Informationen enthalten sind, die das korrekte Verstehen erschweren würden. Das Ziel ist eine Minimierung der Komplexität des Ablaufs. Ein Szenario soll die einzelnen Interaktionen in einen Kontext einbetten und dabei helfen, die Einzelschritte auf dem Weg zum Ziel zu verdeutlichen und nachvollziehbar zu machen.

Definition 6. Szenario [8]

A narrative description of what people do and experience as they try to make use of computer systems and applications

Szenarien können auf verschieden Arten genutzt werden. Zum Beispiel können sie das gegenwärte Verhalten eines Softwaresystems dokumentieren (*Ist-Zustand*) oder aber eine Vision von gewünschten Interaktionen (*Soll-Zustand*) beschreiben. Der letztere Fall ist besonders für die Anforderungsanalyse interessant, da Szenarien zum Erheben, Dokumentieren und Validieren von Interaktionen zwischen Benutzer und System genutzt werden können. Nach Stangl [41] beschreiben Szenarien die funktionalen Anforderungen an das zu entwickelnde System. Auf diese Weise ist leicht ersichtlich, welche Funktionalität bzw. Bedienelemente für die Software benötigt werden, um den vorgegebenen Interaktionsablauf mit dem realen System durchführen zu können.

Kapitel 3

Anforderungsspezifikation

In diesem Kapitel werden Anforderungen formuliert und definiert, damit die mit dieser Arbeit verbundenen Ziele (siehe Abschnitt 1.2 „Ziel der Arbeit“) erreicht werden können. Zu Beginn werden die dazu notwendigen Konzepte erläutert und daraus die entsprechenden Anforderungen abgeleitet, welche die Grundlage für den Entwurf und die Entwicklung eines Softwareprototypen bilden. Zuerst wird dem zu entwickelnden Prototypen ein Name gegeben. Da seine Hauptaufgabe das Aufzeichnen von Interaktionsabläufen mit Mockups ist, wird er im Folgenden als **Mockup Recorder** bezeichnet.
Wie im Abschnitt 1.1 „Motivation“ beschrieben, soll der Mockup Recorder den Prozess der Anforderungsanalyse unterstützen. Der Fokus liegt hierbei auf der Erhebung, Dokumentation und Validierung von Anforderungen an graphische Benutzerschnittstellen und im Besonderen bei deren Verwendung in Form von Szenarien. Dazu werden im ersten Schritt durch den Requirements Engineer in Zusammenarbeit mit den einzelnen Stakeholdern Mockups für die graphische Benutzerschnittstelle erstellt. Anschließend führen die Stakeholder oder auch Requirements Engineers nach ihren Vorstellungen Interaktionsabläufe mit den Mockups durch. Es ist jedoch sinnvoll, die Stakeholder die Interaktionen selbst durchführen zu lassen, da das interaktive Vorgehen dabei hilft, die Wünsche und Bedürfnisse der Stakeholder im Kontext ihres täglichen Arbeitsablaufs exakter erfassen und verstehen zu können. Die erstellten Szenarien sollen als Video exportierbar sein, um die erhobenen Informationen bzw. Anforderungen in einer leicht verständlichen und nachvollziehbaren Weise zu dokumentieren. Die Vorstellungen der Stakeholder können so den nachgelagerten Bereichen der Softwareentwicklung konkreter vermittelt werden (siehe Abschnitt 2.1.3 „Anforderungskommunikation“), wobei auch sichergestellt werden kann, dass die erhobenen Anforderungen die Kundenwünsche vollständig wiedergeben und von den Entwicklern eindeutig zu verstehen sind.

Zunächst werden verschiedene Anwendungsfälle erläutert, um ein besseres Verständnis für die zur Entwicklung des Mockup Recorders erforderlichen Konzepte zu erhalten. Anschließend werden unterschiedliche Konzepte zur Unterstützung

des angestrebten Arbeitsablaufs ausgearbeitet, erläutert und daraus konkrete Anforderungen abgeleitet.

3.1 Anwendungsfälle

Der Hauptanwendungsfall des Mockup Recorders ist das interaktive Erstellen von Mockups und im Besonderen das Spezifizieren von Interaktionsabläufen in enger Zusammenarbeit mit den Stakeholdern. Die Anwendung wird dabei von einem Requirements Engineer zusammen mit einem Stakeholder verwendet. Es soll herausgefunden werden, welche Wünsche und Bedürfnisse die einzelnen Stakeholder haben und wie ihre täglichen Arbeitsabläufe durch die Benutzeroberfläche verbessert werden können.
Der Requirements Engineer führt mit jedem Stakeholder zunächst ein Interview durch, um Informationen über dessen spezifische Wünsche und Bedürfnisse zu sammeln. Während des Interviews stellt der Requirements Engineer Fragen zu den Zielen und Vorstellungen des Stakeholders für die Funktionen der Benutzeroberfläche. Der Requirements Engineer erstellt interaktiv mit dem Stakeholder einen digitalen Mockup. Die Bedienung des Mockup Editors sollte durch Drag & Drop der Bedienelemente an die gewünschte Position geschehen. Es ist auch möglich, dass der Stakeholder seine Vorstellungen auf Papier aufmalt und das Resultat als Bild in die Anwendung importiert wird. Daher kann ein beliebiger Editor verwendet werden, solange die erstellten Mockups als Bild exportiert werden können. Auf diese Weise wird die Verwendung des Mockup Recorders von der Funktion des Erstellens der Mockups entkoppelt. Mit den erstellten Mockups kann der Stakeholder die Interaktionsabläufe selbst durchführen, dabei sollen die markanten Interaktionen aufgenommen werden. Zu diesen markanten Interaktionen gehören unter anderem Mausklicks und Tastatureingaben (siehe Abbildung 4.2), aber keine Mausbewegungen. Diese werden zwischen den markanten Interaktionen interpoliert. Die aufgenommenen Interaktionen können abschließend zur leicht verständlichen und nachvollziehbaren Dokumentation als Video exportiert werden. Dies ermöglicht die Integration der mit dem Mockup-Recorder erstellten Interaktionsabläufe in die Spezifikation, um die textuell beschriebenen Use-Cases auf konkrete Weise visuell zu veranschaulichen. Weiterhin bietet die Anwendung die Möglichkeit, die Interaktionen des Szenarios als Text und die verwendeten Mockups als Bilder zu exportieren.

Ein alternativer Anwendungsfall kann wie folgt skizziert werden: In einer ersten Diskussion mit dem Stakeholder sammelt der Requirements Engineer zunächst Rohanforderungen an die Benutzeroberfläche und die Bedienung. Daraus werden von den Userinterface-Experten Mockupvorschläge entworfen. Daraufhin zeichnet der Requirements Engineer die mit dem Stakeholder bereits besprochene Interaktion mit den Mockups auf und exportiert das Szenario als Video. Hierfür ist es nicht nötig, dass der Stakeholder anwesend ist. Dieses Video sendet der Requirements Engineer

an den jeweiligen Stakeholder, um Feedback in Form von Kritik oder Bestätigung zu erhalten. Auf diese Weise können die Mockups und Interaktionsabläufe leicht validiert werden, um herauszufinden, ob sie den Kundenwünschen entsprechen. Der Vorteil hierbei ist, dass dafür kein Treffen des Stakeholders mit dem Requirements Engineer nötig ist, da der Kunde das Video leicht eigenständig abspielen kann. Der Stakeholder kann das Anschauen und Bewerten der Szenariovideos so optimal in seinen Terminplan integrieren und schnell Feedback geben. Dieses sogenannte *Rapid Feedback* ist sehr wichtig für einen agilen Entwicklungsprozess, da so große Verzögerungen vermieden und Ungewissheiten seitens der Entwickler schnell ausgeräumt werden können [42].

Der alternative Anwendungsfall kann auch wie folgt abgewandelt werden: Nachdem die Mockupvorschläge entworfen worden sind, werden in einer zweiten Sitzung mit dem Stakeholder die Mockups validiert. Der Requirements Engineer schließt hierfür eine zweite Tastatur an den Präsentationscomputer an und fordert den Stakeholder auf, die gewünschten Interaktionen durchzuführen. Der Requirements Engineer führt dabei "versteckt" die Aktionen des Systems aus. Er übernimmt also dessen Rolle und wechselt zwischen den Ansichten, indem er Tastenkombinationen drückt. Während der Stakeholder das Szenario nach seinen Vorstellungen ausführt, nimmt das System die Interaktionen auf. Dieser Versuchsaufbau folgt dabei dem Ansatz der *Wizard-of-Oz* Prototyping Methode [35], welche die Reaktion eines komplexen Systems durch einen für den Benutzer vorborgenen Menschen simuliert.

Die jeweils aufgezeichneten Interaktionen können auf vielfältige Art und Weise aufbereitet und genutzt werden. Ein Beispiel hierfür ist, die Interaktionen als Grundlage für manuelle oder automatisierte Benutzerinterface (GUI) Tests zu verwenden. Dies ist speziell im Fall von JavaFX Anwendungen leicht durchzuführen, da für jede Interaktion das Zielelement in Form eines eindeutigen CSS Selektors[1] gespeichert wird. Damit können die erstellten Mockups sogar bearbeitet werden, ohne die Interaktion erneut aufzeichnen bzw. die Testfälle ändern zu müssen. Die GUI Elemente können beispielsweise verschoben werden, da jedes Element durch den Selektor eindeutig identifiziert werden kann und Interaktionen wie etwa Mausklicks nicht von der absoluten Position innerhalb des Mockups abhängen. Dies ist ein Vorteil gegenüber der direkten Aufnahme von Interaktionen als Video.

[1] Ein Cascading Style Sheet (CSS) Selektor ist eine Zeichenkette zum Auswählen von Elementen (z.B. JavaFX Bedienelemente)

3.2 Rahmenbedingungen und Umfeld

Nach dem Hauptanwendungsfall (siehe Abschnitt 3.1 „Anwendungsfälle“) wird der Mockup Recorder vom Requirements Engineer in enger Zusammenarbeit mit den Stakeholdern im Rahmen eines Gesprächs bedient. Es ist davon auszugehen, dass dies direkt beim Kunden oder beim Requirements Engineer geschieht. Da dies jedoch nicht garantiert werden kann, sollte die Planung für das Umfeld der Verwendung des Mockups Recorders variabel bleiben. Um die Software unabhängig vom Ort benutzen zu können, ist es erforderlich, dass das gesamte System einfach zu transportieren und schnell einsatzbereit ist. Dafür eignen sich insbesondere mobile Endgeräte wie zum Beispiel Notebooks oder Tablets sehr gut. Da heutzutage eine große Auswahl an mobilen Endgeräten vorhanden ist bzw. diese weit verbreitet sind, bleiben die Kosten für die benötigte Hardware in einem akzeptablen Rahmen. Somit kann die erste Anforderung an das System formuliert werden:

[R101] Das System muss unabhängig vom Ort benutzen werden können und einfach zu transportieren sein.

[R102] Das System muss auf einem Notebook oder Tablet ausführbar sein.

Beispielsweise soll der Mockup Recorder mit dem *Microsoft Surface* Tablet benutzt werden können, welches ein vollwertiges Windows Betriebssystem auf Basis der x86/x64 Prozessorarchitektur bereitstellt. Andere weit verbreitete Architekturen – wie unter anderem ARM (*Advanced RISC Machines*) – sollen gegebenenfalls später unterstützt werden. Durch die unterschiedliche Architektur und somit verschiedene Befehlssätze der Prozessoren müssen native Anwendungen angepasst und neu kompiliert werden. Damit der Mockup Recorder auch auf vielen Android Tablets ausgeführt werden kann, soll die Möglichkeit bestehen, den Mockup Recorder um eine Unterstützung für ARM Prozessoren zu erweitern.

[R103] Das System soll um eine ARM Unterstützung erweitert werden können.

Einschränkungen und Vorgaben

Vom Software Engineering Institut der Leibniz Universität Hannover wurden Vorgaben über die zu verwendende Programmiersprache sowie dem mindestens zu unterstützenden Betriebssystem gemacht. Der Mockup Recorder muss in der Programmiersprache Java programmiert sein und mindestens mit den Betriebssystemen Windows 7 und Windows 10 kompatibel sein.

[R201] Das System muss mit den Betriebssystemen Microsoft Windows 7 und Windows 10 kompatibel sein.

Damit der Mockup Recorder von einer Vielzahl von Benutzern mit unterschiedlicher Betriebssystemen ausgeführt werden kann, sollen außerdem MacOS und Linux-Distributionen unterstützt werden.

[R202] Das System soll auch mit anderen Betriebssystemen wie MacOS und Linux-Distributionen kompatibel sein.

Da die Programmiersprache vorgegeben wurde und der Prototyp auf möglichst vielen verschiedenen Betriebssystemen und Architekturen lauffähig sein soll, wird die plattformunabhängige Programmiersprache Java in Version 8 verwendet. Java bietet durch die Nutzung einer virtuellen Maschine zum Ausführen des JavaVM Bytecodes eine gute Abstraktionsschicht, um die Software vom verwendetem Betriebssystem bzw. der Architektur zu entkoppeln. Die Anwendung wird somit ohne Anpassungen bzw. mit nur geringem Aufwand auf den verschiedenen Systemen nutzbar.

[R203] Das System muss in der Programmiersprache Java programmiert sein.

3.3 Funktionale Anforderungen

Als nächstes gilt es, Anforderungen an die Funktionalität des Mockup Recorders aufzustellen. Diese funktionalen Anforderungen sollen als Minimalanforderungen verstanden werden, welche von dem im Zusammenhang mit dieser Arbeit zu entwickelnden Prototypen unterstützt werden müssen, um die Ziele der Arbeit erreichen zu können.

3.3.1 Hauptfunktionen der Anwendung

Nach den zuvor erläuterten Anwendungsfällen müssen zu Beginn des Arbeitsablaufs mit Hilfe des Prototypen Mockups erstellt werden können. Es gibt bereits eine Vielzahl unterschiedlicher Mockup Editoren, von Open-Source Tools wie *Pencil*[2] über kommerzielle wie *balsamiq*[3] bis zu Web-basierten Mockup Editoren wie *placeit.net*[4] oder *draw.io*[5]. Viele Tools bieten nicht nur die Möglichkeit des Erstellens von Mockups, sondern unterstützen auch bereits das Anlegen von Verknüpfungen. So kann beispielsweise einem Button die Funktion zugewiesen werden, bei einem Klick den angezeigten Mockup zu wechseln. Es existieren also bereits Möglichkeiten zum interaktiven Durchspielen von Szenarien, allerdings sind hierbei nicht alle Interaktionsformen möglich. Die Eingabe von Text oder das Interagieren mit komplexen Bedienelementen werden z.B. nur unzureichend oder gar nicht unterstützt. Außerdem bietet keines der zuvor genannten Tools die Möglichkeit, Interaktionsabläufe durch die Stakeholder ausführen zu lassen und diese gleichzeitig aufzunehmen und als Video zu exportieren. Diese Funktionalität ist jedoch sinnvoll, um den Stakeholder stärker in den Entwicklungsprozess einzubinden und gleichzeitig den gewünschten Ablauf unverfälscht auf geeignete Weise zu dokumentieren. Nicht nur statische Ansichten der Bedienoberfläche in Form von Mockups, sondern auch

[2] `https://pencil.evolus.vn`
[3] `https://balsamiq.com`
[4] `https://placeit.net`
[5] `https://www.draw.io`

die Abläufe der Bedienung sollen erfasst und in der Anforderungspezifikation dokumentiert werden. Besonders bei sehr wichtigen Szenarien, welche exakt auf die Wünsche und Bedürfnisse der Stakeholder abgestimmt sein müssen, kann das Spezifizieren der Abläufe zu einem eindeutigen Verständnis der Entwickler und somit einem besseren Produkt beitragen. Denn wenn die Entwickler den Ablauf genau verstehen, kann die Bedienoberfläche entsprechend ausgerichtet werden. Ein wichtiger Punkt ist nicht nur das Verständnis der Entwickler an sich, sondern auch die benötigte Zeit, um ein gutes Verständnis für die Anforderungen der Stakeholder zu erreichen. Je verständlicher bzw. nachvollziehbarer das Dokumentationsmedium die gewünschten Anforderungen der Stakeholder beschreibt, desto kürzer ist der Entwicklungsprozess und um so schneller können die Anforderungen in eine reale Software übersetzt werden. Der Zeit- und Kostenaufwand wird somit reduziert und bleibt auf einem akzeptablen Niveau.

[R301] Das System muss die Möglichkeit bieten, Mockups in digitaler Form zu erstellen.

Die Mockups sollen nicht nur eine statische Darstellung bieten, sondern den jeweiligen Zustand von realen Bedienelementen wiedergeben können. Die Bedienelemente müssen daher auf Interaktion des Benutzers reagieren. Ein Beispiel hierfür ist die Möglichkeit der Eingabe und anschließenden Darstellung von Texteingaben. Des Weiteren sollen z.B. Checkboxen durch Benutzerinteraktion ihren Zustand wechseln und diesen korrekt wiedergeben können. Ein wichtiger Vorteil von digitalen Mockups – im Vergleich zu herkömmlichen analogen Mockups auf Papier – ist eine unkomplizierte und mit geringem Aufwand durchführbare Bearbeitung der erstellten Mockups. Das Bearbeiten ist notwendig, um den iterativen Prozess z.B. zum Erstellen von Prototypen so gut wie möglich zu unterstützen. Dieser Prozess wiederholt dabei die Design-, Implementations- und Analysephasen und wird deshalb auch *DIA-Cycle* genannt [35]. Das Konzept des iterativen Ablaufs findet sich insbesondere auch in der agilen Softwareentwicklung wieder, also dem iterativen und flexiblen Vorgehen bei Projekten. Daraus folgt eine weitere funktionale Anforderung:

[R302] Das System muss die Möglichkeit bieten, Mockups nachträglich zu bearbeiten.

Es soll nicht unerwähnt bleiben, dass analoge Mockups auch Vorteile bieten. So ist es für Stakeholder ohne Kenntnisse der Bedienung eines Mockup Editors möglich, ihre Vorstellungen von der Gestaltung der Benutzeroberfläche auf Papier aufzumalen. Auf diese Weise kann die Vorstellung des Stakeholder auch unverfälscht erfasst werden. Des Weiteren können Mockups auf Papier sehr schnell erstellt werden. Aus diesen Gründen muss der Prototyp die Möglichkeit bieten, analog erstellte Mockups zu importieren. Dafür werden die Mockups zuerst digitalisiert. Die einfachste Lösung hierfür ist ein Einscannen oder Abfotografieren der gezeichneten Mockups. Die daraus entstehenden Bilder können anschließend vom Mockup Recorder importiert und zu einem digitalen Mockup im .fxml Format konvertiert

werden.

[R303] Das System muss Bilder im .png oder .jpeg Format zu einem digitalen Mockup konvertieren können.

Ein weiterer essentieller Arbeitschritt ist das Aufzeichnen von Interaktionsabläufen mit den zuvor erstellten Mockups. Wie im Abschnitt 1.2 „Ziel der Arbeit“ erläutert, bildet das Spezifizieren von Interaktionen den Kern dieser Arbeit. Es muss möglich sein, Interaktionen mit den Mockups durch die weitverbreiteten Eingabegeräte wie Tastatur, Maus und berührungsempfindlichen Bildschirmen (Touch) aufzuzeichnen. Diese stellen heutzutage die am häufigsten verwendeten Eingabegeräte dar, welche für Tablets, Notebooks und stationäre Computer verwendet werden.
Auf den ersten Blick ähnelt dieses Vorgehen der Funktionsweise eines Screenrecorders – wie zum Beispiel dem weit verbreitetem Programm *Camtasia Studio*[6]. Allerdings gibt es einen wesentlichen Unterschied. Camtasia Studio zeichnet im Normalfall den gesamten Bildschirm als einen Screencast in Echtzeit auf. Damit ist gemeint, dass z.B. eventuelle Unterbrechungen des Interaktionsablaufs durch ein Gespräch zwischen Stakeholder und Requirements Engineer das entstehende Video unnötig verlängern. Im Gegensatz dazu soll der Mockup Recorder nur die markanten Interaktionen mit den Mockups erfassen und so einen kurzen durchgängigen Ablauf produzieren. Das Ziel ist ein möglichst kurzes Video des Szenarios zu erstellen, welches keine Unterbrechungen bzw. unnötigen Informationen enthält. Dies wird erreicht, indem nur die markanten Interaktionen wie etwa Mausklicks oder Texteingaben aufgezeichnet und die Bewegungen des (Maus-)Zeigers dazwischen interpoliert werden. Durch die Interpolation wird das Video auf die wesentlichen Interaktionsaspekte beschränkt. Daraus folgt jedoch, dass eine Aufnahme von Ton nicht unterstützt wird, da dies die Videoaufzeichnung in Echtzeit erfordert, wobei letzteres der Anforderung nicht genügen würde, die Szenarien mit geringem Aufwand bearbeiten zu können (siehe [R305]) und Videos somit nur als Exportmedium dienen sollen. Weitere Informationen zum Kontext sollten als Annotationen in das Video eingebunden werden können. Der Mockup Recorder unterstützt dies zum aktuellen Zeitpunkt noch nicht, da die Funktion für den Hauptanwendungsfall nicht benötigt wird. Es wäre jedoch sinnvoll, diese Funktionalität später zu integrieren.

[R304] Das System muss die Möglichkeit bieten, vom Benutzer getätigte Interaktionen mit Mockups aufzuzeichnen. Dies beinhaltet Maus-, Tastatur- und Touch-Eingaben.

Die aufgezeichneten Szenarien sollen, wie bereits bei den Mockups, nachträglich bearbeitet werden können. Zum einen soll dadurch das iterative Erstellen von Mockups und Szenarien unterstützt werden. Zum anderen soll das nachträgliche Editieren von exportierten Videos vermieden werden, da dies nur mit großem

[6] `https://www.techsmith.de/camtasia.html`

Aufwand durchgeführt werden kann. Um ein Video nachträglich zu bearbeiten, wäre es nämlich nötig, das Video mit einem Schnittprogramm zu öffnen, Teile zu entfernen oder neue hinzuzufügen und das Video anschließend wieder zu speichern. Werden Teile entfernt, ist es in der Regel nicht möglich, eine durchgehende Bewegung für den (Maus-)Zeiger beizubehalten. Der Mockup Recorder hingegen unterstützt das Bearbeiten von Interaktionen mit einzelnen Mockups, ohne dass der gesamte Interaktionsablauf neu aufgenommen bzw. bearbeitet werden muss. Insbesondere können Bedienelemente verschoben werden, ohne den Interaktionsablauf bearbeiten zu müssen, da sie durch einen Identifier (CSS Selektor) wiedergefunden werden können. Nach einem erneuten Export erhält man mit geringem Aufwand ein Video des neuen Interaktionsablaufs.

[R305] Das System muss die Möglichkeit bieten, Szenarien mit geringem Aufwand nachträglich zu bearbeiten.

Um unnötigen Aufwand zur Bearbeitung zu vermeiden, wurde festgelegt, dass Videos nur ein Exportmedium darstellen und die Szenarien innerhalb des Systems in einem leicht bearbeitbaren Format gespeichert werden müssen.

[R306] Das System muss Szenarien in einem mit geringem Aufwand bearbeitbaren Format speichern.

[R307] Das System muss fähig sein, Szenarien als Video im MP4 Format zu exportieren.

[R308] Der Name eines Videos muss aus dem Szenarionamen sowie dem Zeitpunkt des Exports bestehen.

Der Mockup Recorder muss außerdem in der Lage sein, die aufgezeichneten Szenarien innerhalb des Programms wiedergeben zu können. Ein vorheriger Export und die Wiedergabe in einem standardmäßigen Videoplayer (z.B. *VLC Media Player*[7]) sollen explizit vermieden werden. Dadurch soll ein angenehmeres Arbeiten mit dem Mockup Recorder, ohne ständige Wechsel zwischen verschiedenen Programmen, gewährleistet werden.

[R309] Das System muss Szenarien auch ohne Videoexport wiedergeben können.

Bisher werden zur Spezifikation von Interaktionen mit graphischen Benutzerschnittstellen oftmals Use-Cases eingesetzt, die den Ablauf abstrakt beschreiben. Diese werden mit Fließtext in Tabellenform in die Spezifikation eingebunden. Es ist wichtig, zu verdeutlichen, dass die Möglichkeit der Aufzeichnung von Szenarien und des Exports als Video nicht die herkömmliche Beschreibung in Textform ersetzen soll. Vielmehr unterstützen die Videos das eindeutige Verstehen und

[7] `https://www.videolan.org`

Nachvollziehen der dokumentierten Anforderungen durch eine konkrete Darstellung. Da der Prototyp aber auch zum Erheben von Anforderungen gedacht ist, kann es je nach Anwendungsfall vorkommen, dass die Szenarien bzw. Use-Cases zuerst in Videoform vorliegen und anschließend mittels Fließtext beschrieben werden. Um diesen Ablauf besser zu unterstützen, muss der Mockup Recoder die Möglichkeit bieten, die markanten Interaktionen in einen einfachen Ablauf in Textform zu konvertieren, welcher als Vorlage für eine ausformulierte Version dienen kann.

[R310] Das System muss den Interaktionsablauf als Text (mittels Textbausteinen) exportieren können, damit dieser in *Microsoft Word* verwendet werden kann.

In vielen Softwareprojekten gibt es mehrere Stakeholder, welche jeweils ihre eigene Vorstellung von dem zu entwickelnden System haben. Um die einzelnen Vorstellungen, Wünsche und Bedürfnisse herausarbeiten und verstehen zu können, sollen die Szenarien mit jedem Stakeholder aufgezeichnet werden. Um die Übersicht innerhalb eines Softwareprojekts zu behalten und bereits erstellte Mockups für weitere Szenarien wiederverwenden zu können, müssen die Mockups und Szenarien zur einfachen Handhabung in einem gemeinsamen Ordner gespeichert werden.

[R311] Das System muss Mockups und Szenarien in einem gemeinsamen Projektordner speichern.

Um so mehr Beteiligte es gibt und je mehr Änderungen an den Mockups und Szenarien gemacht werden, desto sinnvoller ist die Integration einer Versionsverwaltung (z.B. Subversion[8], Git[9] oder Mercurial[10]). Eine Versionsverwaltung ist sinnvoll, um Änderungen leicht nachvollziehen und wenn nötig rückgängig machen zu können. Außerdem unterstützt sie das verteilte Arbeiten an einer einzigen (zentralen) Datenbasis, dem *Repository*. Dabei sollte es nur eine *Single source of truth*[11] geben, welche die Verwaltung der Daten stark vereinfacht und eine Vielzahl von möglichen Problemen verhindert. So stellt sich niemals die Frage, welche Version die aktuellste ist, da diese stets im zentralen Repository zu finden ist. Zusätzlich treten durch die Nutzung seltener Konflikte zwischen verschiedenen Dateiversionen auf.
Eine direkte Integration einer Versionsverwaltung in den Mockup Recorder würde Vorteile hinsichtlich der Usability bieten. Allerdings entstehen dadurch auch Nachteile, welche die Vorteile überwiegen. So ist es zum Beispiel mit großem Aufwand verbunden, verschiedene Versionsverwaltungen zu integrieren und fortlaufend zu warten. Die eventuell einfachere Benutzung durch die direkte Integration einer Versionsverwaltung in den Mockup Recorder rechtfertigt nicht den hohen Aufwand und die entstehenden Kosten, da sie auch über die Kommandozeile bedient

[8] `https://subversion.apache.org`
[9] `https://git-scm.com`
[10] `https://www.mercurial-scm.org`
[11] Single source of truth (SSOT) ist ein Prinzip aus dem Datenmanagement und besagt, dass es im Falle redundanter Datenbestände genau einen geben sollte, auf dessen Inhalt Verlass ist.

werden kann und somit kein entscheidender Mehrwert entsteht. Insgesamt ist die generelle Nutzung einer Versionsverwaltung aber sinnvoll und zu empfehlen. Um die Versionsverwaltung effektiv nutzen zu können, müssen die Mockups und Szenarien in einem für Menschen lesbaren Format vorliegen. Durch die Nutzung von Textformaten können weitere Vorteile einer Versionsverwaltung, wie zum Beispiel das verständliche Anzeigen von Unterschieden (Diffs), genutzt werden. Binäre Dateiformate bieten diesen Vorteil nicht.

[R312] Die Mockups und Szenarien müssen in einem von Menschen lesbaren Format gespeichert werden, sodass sie mit einer Versionsverwaltung (z.B. Subversion, Git oder Mercurial) verwaltet werden können.

3.3.2 Graphische Benutzeroberfläche

Nachdem die primäre Funktionalität der Anwendung spezifiziert wurde, wird in diesem Abschnitt genauer auf das Layout und die Bedienung des Mockup Recorders eingegangen. In der Softwareentwicklung bilden graphische Benutzeroberflächen die Schnittstelle zwischen Benutzer und System. Das Layout und die Bedienung der Oberfläche sind von erheblicher Bedeutung für die Usability und Zufriedenheit der Benutzer. Die Verwendung muss daher auf die Wünsche und Bedürfnisse der Benutzer ausgerichtet sein, damit sie sich gut in den täglichen Arbeitsablauf integriert und den Benutzer beim Erreichen seiner Ziele sinnvoll unterstützt. Zudem kann die Effizienz der Bedienung positiv beeinflusst werden, indem beispielsweise für professionelle Nutzer Tastenkombinationen vorhanden sind.

Für den Entwurf der Benutzeroberfläche wurde ein Wireframe Mockup erstellt (siehe Abbildung 3.1). Die Abbildung soll zunächst einen Überblick über die Bedienoberfläche des Mockup Recorders geben, bevor auf die Funktionalität und das Zusammenspiel der einzelnen Bedienelemente eingegangen wird.

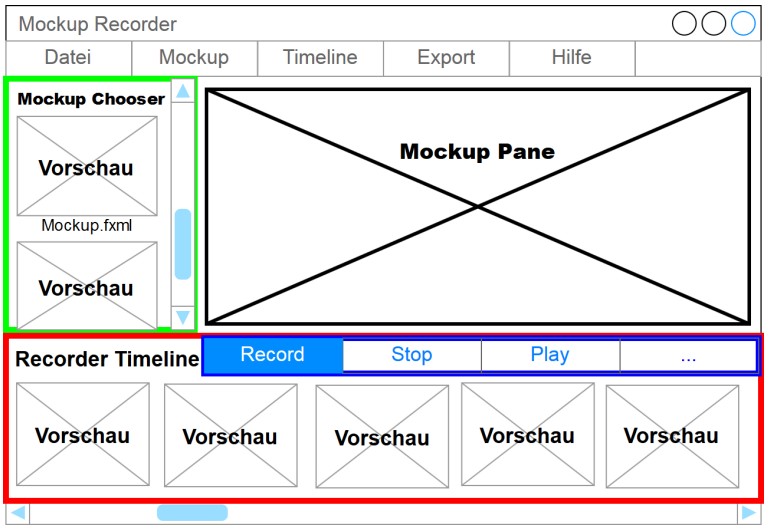

Abbildung 3.1: Wireframe Darstellung des Mockup Recorders

Die Funktionalität der Software kann im Wesentlichen zwei Aufgabenbereichen zugeordnet werden. Der erste Bereich umfasst das Erstellen und Bearbeiten von Mockups. Dafür wird der *Gluon SceneBuilder* verwendet, welcher bereits eine intuitive Bedienung mittels Drag & Drop ermöglicht. An dieser Stelle wird darauf jedoch nicht weiter eingegangen, da der SceneBuilder ein eigenständiges Programm und somit nicht Teil der Benutzeroberfläche des Mockup Recorders ist. Auf den SceneBuilder wird später im Abschnitt 4.2 „Schnittstellen und angrenzende Systeme“ genauer eingegangen. Der zweite Bereich nutzt die erstellten Mockups zum Aufzeichnen von Szenarien. Die Bedienoberfläche des Mockup Recorders ist auf diesen Aufgabenbereich ausgerichtet. Dafür wird zunächst eine Liste aller für ein Projekt erstellter Mockups benötigt, um eine Übersicht zu erhalten. Zur leichteren Orientierung werden die einzelnen Mockups jeweils mit einem Vorschaubild und ihrem Dateinamen angezeigt. Dieses Bedienelement wird im folgendem als *Mockup Chooser* (grün in Abbildung 3.1) bezeichnet.

Mockup Chooser

[R401] Der Mockup Chooser muss alle im aktuellen Projektordner vorhandenen Mockups mit einem Vorschaubild darstellen.

[R402] Falls der Benutzer einen Doppelklick ausführt, muss das System den gewählten Mockup im Mockup Pane darstellen.

[R403] Mockups müssen per *Drag & Drop* zur Recorder Timeline hinzugefügt werden können.

Für das Aufzeichnen von Szenarien muss als nächstes ein für den Ablauf benötigter Mockup ausgewählt und chronologisch angeordnet werden, bevor Interaktionen mit diesem Mockup aufgezeichnet werden können. Natürlich können die Mockups auch zu einem späteren Zeitpunkt mittels *Drag & Drop* verschoben oder die aufgenommenen Interaktionen bearbeitet werden. Dieser Teil der Oberfläche erhält den Namen *Recorder Timeline* (rot in Abbildung 3.1). Der Begriff Timeline lässt sich als “Zeitstrahl“ übersetzen und bildet so eine Metapher für die chronologische Anordnung der Mockups.

Recorder Timeline

[R501] Die Timeline muss alle Mockups gemäß des zeitlichen Ablaufs mit einem Vorschaubild darstellen.

[R502] Mockups müssen per *Drag & Drop* sortiert werden können.

[R503] Die Timeline muss den ausgewählten Mockup im Mockup Pane darstellen.

Es ist sinnvoll, für das Anzeigen der verfügbaren Mockups und das Anordnen zwei unterschiedliche Bedienelemente zu verwenden. Zum einen sollen nicht immer alle für ein Projekt erstellten Mockups in einem Szenario verwendet werden. Zum anderen kann der Fall auftreten, dass ein einzelner Mockup mehrmals innerhalb eines Szenarios vorkommmt und jeweils verschiedene Interaktionen aufgezeichnet werden sollen. Für das Hinzufügen und Anordnen von Mockups in der Recorder Timeline wird das Konzept des Drag & Drop verwendet. Ein Mockup in der Recorder Timeline kann durch einen Klick ausgewählt werden und wird daraufhin automatisch im sogenannten *Mockup Pane* (schwarz in Abbildung 3.1) dargestellt. Dieser Bereich dient sowohl dem Aufzeichnen als auch Abspielen von Interaktionen. In den exportierten Videos ist nur der Mockup Pane zu sehen.

Mockup Pane

[R601] Der gewählte Mockup muss inklusive der Bedienelemente dargestellt werden.

[R602] Der Mockup Pane muss fähig sein, Benutzereingaben für den aktuellen Mockup aufzeichnen und wiedergeben zu können.

[R603] Der Mockup Pane muss die Möglichkeit bieten, die Größe des Aufnahmebereichs festzulegen.

Um die Aufahme, Wiedergabe und andere wichtige Funktion zu steuern, wird die *Recorder ToolBar* (blau in Abbildung 3.1) verwendet. Sie besteht aus Buttons zum Starten des Aufnahme-, Abspiel und Exportmodus und enthält weitere nützliche Funktionen. Es kann sowohl die Größe des Mockup Panes und somit des Videos, als auch die Geschwindigkeit der (Maus-)Zeigerbewegung für die Wiedergabe eingestellt werden. Damit der Benutzer leicht erkennen kann, welche Aktionen zum aktuellen Zeitpunkt verfügbar sind, werden nicht erlaubte Buttons deaktiviert (ausgegraut). Dieses Konzept verbessert die Usability, da der Benutzer einen Überblick über die aktuellen Aktionsmöglichkeiten erhält und leichter eine Entscheidung über die als nächstes auszuführende Aktion treffen kann.

Recorder ToolBar

[R701] Die ToolBar zum Aufzeichnen, Wiedergeben und Exportieren von Interaktionen mit Mockups muss mindestens über die Funktionen Record-Start, Record-Stop, Play, Pause, Skip-To-Begin, Skip-To-End, Resize sowie Export verfügen.

[R702] Die ToolBar muss eine Möglichkeit bieten, die Wiedergabegeschwindigkeit anzupassen.

[R703] Bei Start des Mockup Recorders sind alle ToolBar Buttons aktiviert.

[R704] Während der Aufnahme sind Record-Start, Play, Pause und Export Button deaktiviert.

[R705] Während der Wiedergabe sind Record-Start, Record-Stop, Play, Skip-To-Begin, Skip-To-End und Export Button deaktiviert.

[R706] Während des Videoexports sind Record-Start, Record-Stop, Play, Skip-To-Begin, Skip-To-End und Export Button deaktiviert.

Für die Verwendung des Mockup Recorders nach dem *Wizard-of-Oz* Prototyping Prinzip [35] und um eine schnelle Bedienung für professionelle Benutzer zu ermöglichen, sollen zudem die wichtigsten Funktionen über Tastenkombinationen steuerbar sein.

[R707] Die ToolBar muss mittels Tastenkombinationen bedient werden können. Dafür müssen mindestens folgende Funktionen unterstützt werden:

Strg + R: Die Aufnahme wird gestartet oder beendet.

Strg + Leertaste: Die Wiedergabe wird gestartet oder beendet.

Strg + ←: Der vorherige Mockup wird in der Timeline ausgewählt.

Strg + →: Der nächste Mockup wird in der Timeline ausgewählt.

Strg + Q: Die Timeline scrollt zum ersten Mockup.

Strg + E: Die Timeline scrollt zum letzten Mockup.

Um die Steuerung der Anwendung mittels der Recorder ToolBar zu verifizieren und eine korrekte Funktionsweise zu garantieren, wurde ein UML Zustandsdiagramm (siehe Abbildung 3.2) erstellt.

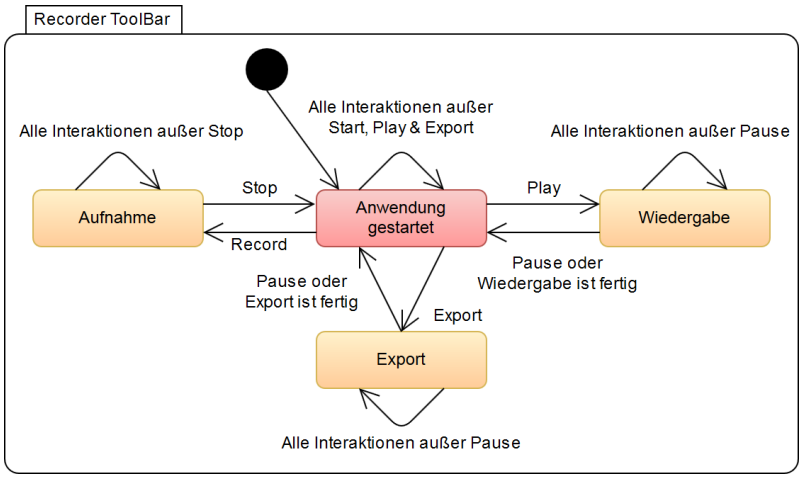

Abbildung 3.2: UML Zustandsautomat der Recorder ToolBar

3.4 Qualitative Anforderungen

In diesem Abschnitt werden Anforderungen bezüglich der Qualität der Umsetzung einzelner Aspekte des Mockup Recorders aufgestellt.

Der Mockup Recorder soll, entsprechend des Hauptanwendungsfalls, während des Gesprächs mit den Stakeholdern verwendet werden. Hierfür ist es erforderlich, dass dieser schnell einsatzbereit ist und schnell auf Benutzereingaben reagiert. Zum Beispiel darf das Starten der Anwendung bzw. das Laden gespeicherter Szenarien zu keiner erheblichen Verzögerung führen. Es wurde festgelegt, dass alle Funktionen maximal drei Sekunden benötigen dürfen bis eine Reaktion des Systems erfolgt.

[R801] Die maximale Reaktionszeit des Systems darf drei Sekunden nicht überschreiten.

Weiterhin müssen Szenarien genau so wiedergegeben werden, wie sie aufgezeichnet wurden. Dieses Qualitätsmerkmal ist entscheident, um den Kundenwunsch exakt repräsentieren zu können. Im Allgemeinen müssen die durchgeführten Interaktionen in chronologischer Reihenfolge aufgenommen und abgespielt werden. Außerdem müssen Details, wie etwa die Position von Mausklicks und die in Textfelder eingegebenen Zeichen, korrekt erfasst werden.

[R802] Das System muss Szenarien exakt aufzeichnen und wiedergeben können.

Auch der Export von Videos muss bestimmten Mindestqualitätsstandards folgen. Die Breite und Höhe des generierten Videos muss den gewählten Mockups bzw. dem Mockup Pane entsprechen. Auf diese Weise kann der Benutzer die Größe des generierten Videos selbst festlegen.

[R803] Die Breite und Höhe generierter Videos muss den gewählten Mockups bzw. dem Mockup Pane entsprechen.

Zudem müssen die exportierten Videos den Ablauf flüssig wiedergeben, damit dem Ablauf gut gefolgt werden kann. Hierfür wurde entschieden, dass mindestens 30 Bilder pro Sekunde aufgenommen werden müssen.

[R804] Das System muss Videos mit mindestens 30 Bilder pro Sekunde aufnehmen.

Kapitel 4

Konzepte des Prototyps

Im vierten Kaptiel wird auf die genaue Umsetzung der im vorherigen Kapitel spezifizierten Anforderungen in Form eines Softwareprototypen eingegangen. Zu Beginn wird die Basis der technischen Umsetzung erläutert, danach die Implementierungsdetails für die Benutzeroberfläche und die Softwarearchitektur. Anschließend folgt ein Ausblick auf sinnvolle Erweiterungen des Prototypen. Zum Schluss wird auf verwandte Arbeiten eingegangen, um Unterschiede zu diesen aufzuzeigen und deren Ziele und Umsetzung von dieser Arbeit abzugrenzen.

4.1 Technische Grundlagen

Die Basis für die Umsetzung der Konzepte des Softwareprototypen bildet die gewählte Programmiersprache. Wie aus der Anforderungsspezifikation hervorgeht, soll der Prototyp von einer Vielzahl Benutzer mit unterschiedlichen Endgeräten wie etwa Notebooks und Tablets verwendet werden [R102] und muss daher auf den weit verbreiteten Betriebssystemen Windows [R201], MacOS, Android und anderen Linux-Distributionen [R202] lauffähig sein. Dies gewährleistet die Verwendung der plattformunabhängigen Programmiersprache Java in Version 8 [R203]. Zur einfachen Distribution und Installation bietet Java den Export der Anwendung als Jar-Archiv an, welches die gesamte Anwendung inklusive Abhängigkeiten wie etwa Bilder oder Drittanbieter Libraries enthält. Zur Distribution der Anwendung genügt es, die Jar Datei zur Verfügung zu stellen. Zur Installation wird die Jar Datei auf ein Speichermedium des Endgerätes kopiert und die *Java Runtime Environment*[1] installiert. Es sind keine weiteren Schritte zur Einrichtung erforderlich.

Ab Version 8 von Java ist das Application Framework *JavaFX*[2] in das Java Runtime Environment aufgenommen worden und löst damit das zuvor verwendete *Swing* Framework zum Erstellen von graphischen Benutzerschnittstellen ab. JavaFX

[1] `https://www.oracle.com/technetwork/java/javase/downloads/jre8-downloads-2133155.html`

[2] `https://docs.oracle.com/javase/8/javafx/get-started-tutorial/jfx-overview.htm`

bietet eine Vielzahl weiterer Funktionen, die im Vergleich zum veralteten Swing Framework zeitgemäße Konzepte verfolgen. Als wichtiges Beispiel ist die direkte Integration des *Model-View-Controller* Design Patterns (MVC) zu nennen, welches die Grundlage für alle JavaFX Anwendungen bildet. Das MVC Design Pattern erlaubt die strikte Trennung von Anwendungslogik, verwendeter Datengrundlage und ihrer graphischen Darstellung, was zu einer geringen Kopplung und somit einer guten Wiederverwendbarkeit der einzelnen Komponenten führt. Jede einzelne Komponente kann unabhängig von den anderen bearbeitet bzw. ausgetauscht werden. In den meisten Fällen sind durch die Trennung der Komponenten keine weiteren Anpassungen nötig, somit wird der Arbeitsaufwand für Änderungen verringert. Für die Modelle werden normale Java Klassen verwendet, die Attribute für alle benötigten Daten enthalten und eine Schnittstelle nach dem Prinzip des *Information Hiding* über Getter- und Settermethoden bereitstellen. Für die Darstellung der Benutzeroberfläche werden sogenannte *Views* verwendet, welche in der auf XML aufbauenden Sprache *FXML*[3] erstellt werden und durch den Einsatz von *Cascading Style Sheets* (CSS) ähnlich wie HTML gestaltet werden können. Die Views sind dadurch komplett von der Datengrundlage der Anwendung entkoppelt und können leicht ausgetauscht werden. Mittels des *Dependency Injection* Konzeptes [17] werden die in der .fxml Datei enthaltenen Bedienelemente zur Laufzeit in den Controller injiziert, damit auf die Bedienelemente zugegriffen werden kann.

Für den Videoexport von Szenarien wird die Open-Source Library *JavaCV* von Bytedeco[4] verwendet. JavaCV besteht aus einer Reihe von Schnittstellen zum Einbinden von nativen Libraries zur Aufnahme, Bearbeitung und Analyse von Videos in Java. Für den Mockup Recorder wird ausschließlich die Schnittstelle für das weitverbreitete Programm *FFmpeg*[5] verwendet, welches eine effiziente Möglichkeit zur Generierung von Videos aus Einzelbildern bietet. Der größte Vorteil bei der Verwendung von nativen in C++ geschriebenen Libraries ist die Ausnutzung von Hardware Beschleunigung und somit die Steigerung der Performance der Videogenerierung. Außerdem fällt der Overhead der *Java Virtual Machine* (JVM) durch Verwendung von nativ ausführbaren Programmcodes weg. Unter Verwendung von FFmpeg in Verbindung mit der Hardware Beschleunigung *Quick Sync Video*[6] von Intel Prozessoren, die seit der *Haswell* Generation aus dem Jahr 2013 auch das *MPEG-4 AVC/H.264* Format unterstützt, kann die benötigte Zeit zur Videogenerierung im Vergleich zu einem reinem Softwareenkodierer stark gesenkt werden. Dies bedeutet, dass Videos mit einer höheren Bildrate (Frames-per-second) bei gleichbleibender CPU Auslastung enkodiert werden können. Im Jahr 2013 wurde eine quelloffene Implementierung von H.264 von

[3] `https://docs.oracle.com/javase/8/javafx/api/javafx/fxml/doc-files/introduction_to_fxml.html`

[4] `https://bytedeco.org`

[5] `https://ffmpeg.org`

[6] `https://trac.ffmpeg.org/wiki/HWAccelIntro`

Cisco unter der freien BSD Lizenz veröffentlicht[7] und hat sich seither als primärer Videocodec für das Streaming von Videos in HTML5 durchgesetzt[8]. H.264 bietet sich hierfür besonders an, da es eine gute Qualität trotz starker Kompression ermöglicht und somit auch für hoch aufgelöste Bilddateien hervorragend geeignet ist. Im Vergleich zu dem Vorgänger *MPEG-2 H.262* benötigt H.264 nur rund ein Drittel des Speicherplatzes bei gleichbleibender Bildqualität. Auf Grundlage der guten Qualität und der Verfügbarkeit von Hardware Beschleunigung in den weitverbreiteten Intel Prozessoren (z.B. im Microsoft Surface Tablet) wird das MPEG-4 AVC/H.264 Format zur Videogenerierung für den Export von Szenarien verwendet [R307].

Zum Speichern und Laden von Szenarien wird eine weitere Open-Source Library verwendet. Die Library *Jackson*[9] dient zum (De-)Serialisieren von Daten im *JavaScript Object Notation*[10] (JSON) Format. Dabei handelt es sich um ein menschenlesbares Textformat, das sich durch seine Einfachheit und – trotz des Namens – Unabhängigkeit zur verwendeten Programmiersprache auszeichnet. Die Einfachheit bezieht sich hierbei auf den Aufbau bzw. die Struktur, die durch eine $LL(1)$ Grammatik ausgedrückt werden kann. Damit kann JSON von einem Parser mit nur einem einzelnen sogenannten Vorschausymbol eindeutig und zugleich sehr effizient geparsed werden. Jackson verfolgt das Konzept des *Object Bindings*, welches es ermöglicht, Java Objekte direkt in JSON oder umgekehrt umzuwandeln. Dies vereinfacht die Handhabung der Library und fördert zugleich die Verwendung von Modellklassen, die bereits aufgrund des MVC Design Patterns genutzt werden.

4.2 Schnittstellen und angrenzende Systeme

In diesem Abschnitt soll auf die Schnittstellen zu angrenzenden Systemen eingegangen werden. Grundsätzlich ist zunächst zwischen Eingabe- und Ausgabeschnittstellen zu unterscheiden. Die Eingabeschnittstellen dienen dazu, Daten von anderen Systemen entgegenzunehmen und eine Verwendung mit dem Mockup Recorder zu ermöglichen. Da sich der Mockup Recorder auf die Aufnahme von Interaktionen mit Mockups konzentriert, muss es eine Schnittstelle für den Import von Mockups geben. Der Mockup Recorder unterstützt dabei zwei verschiedene Eingabeformate. Zum einen das von JavaFX genutzte Format .fxml für graphische Benutzeroberflächen und zum anderen Bilder in den Formaten .png und .jpeg, um Mockups von beliebigen Editoren importieren zu können. Das Erstellen und Pflegen von Mockups ist zeitaufwendig, daher wird diese Prototypingmethode in der Praxis nur selten bzw. nicht konsequent verwendet. Mockups im .fxml Format bieten den Vorteil, dass sie für die reale Umsetzung der Anwendung weiterverwendet

[7] `https://blogs.cisco.com/collaboration/open-source-h-264-removes-barriers-webrtc`

[8] `https://developer.mozilla.org/en-US/Apps/Fundamentals/Audio_and_video_delivery/H.264_support_in_Firefox`

[9] `http://wiki.fasterxml.com/JacksonHome`

[10] `http://www.json.org/json-de.html`

werden können, was den Arbeitsaufwand reduziert und somit den genannten Nachteil minimiert. Außerdem kann mit den Bedienelementen eines FXML Mockups genau so interagiert werden wie mit einer bereits fertiggestellten Anwendung. Dies bietet Vorteile für das Aufzeichnen von Szenarien, da Reaktionen der Bedienelemente auf Benutzereingaben wie z.B. die Darstellung von eingegebenem Text ermöglicht werden.

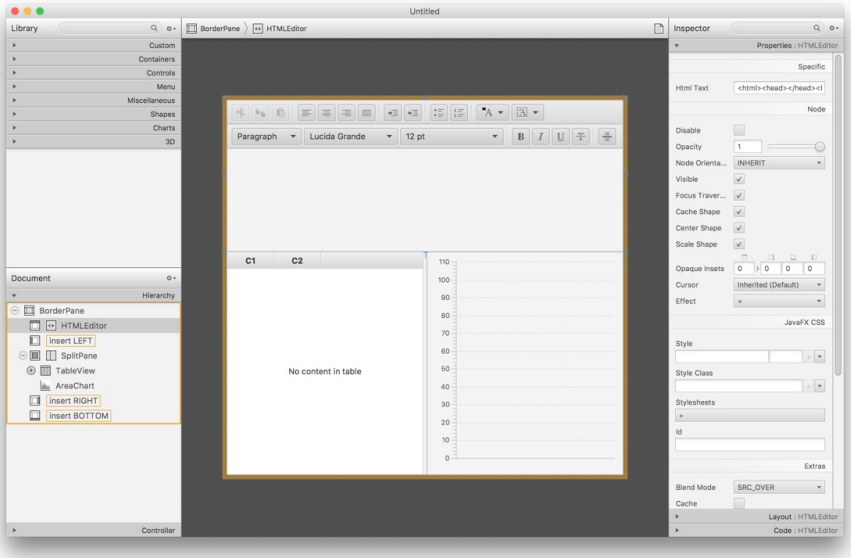

Abbildung 4.1: Hauptansicht des Gluon SceneBuilders

Für das Erstellen von Mockups im .fxml Format wird der *Gluon SceneBuilder*[11] (ehemals *Oracle JavaFX SceneBuilder*) verwendet. Dieser *„What You See Is What You Get“* (WYSIWYG) Editor (siehe Abbildung 4.1) ermöglicht das Erstellen von Mockups ohne größeren Aufwand. Die benötigten Bedienelemente werden dafür mittels *Drag & Drop* wie gewünscht angeordnet. Da der SceneBuilder der Standardeditor für JavaFX Anwendungsoberflächen ist und somit bereits ein großes Spektrum von Bedienelementen und Funktionen abdeckt, wurde für diese Arbeit entschieden, diesen zu verwenden und keinen eigenen Editor zu programmieren.

Die Möglichkeit des Imports von Bildern soll den Mockup Recorder vom verwendeten Editor entkoppeln und es so ermöglichen, ohne zusätzlichen Aufwand einen Editor der Wahl zu verwenden [R303]. Die Importfunktion erzeugt einen Mockup im .fxml Format mit dem ausgewählten Bild als Hintergrund. Auf diese Weise können leicht JavaFX Bedienelemente eingefügt und über dem Bild platziert werden. Es entstehen also keine Nachteile durch die Verwendung von Bildern im Vergleich zu Mockups aus purem FXML. Der Mockup Recorder bietet mehrere Ausgabeschnittstellen, um die aufgenommenen Szenarien für andere Systeme zugänglich zu machen. Primär wird hierfür der Export als Video unterstützt. Videos

[11] `https://gluonhq.com/products/scene-builder`

als visuelles Medium eignen sich besonders gut zum verständlichen Darstellen von Interaktionsabläufen. Sie sollen dabei die bisher oft verwendeten Use-Cases in Form einer Tabelle mit textueller Beschreibung nicht ersetzen, sondern sind dafür gedacht, den konkreten Ablauf auf verständliche und nachvollziehbare Weise zu dokumentieren. Videos sind auch von Personen ohne tieferes technisches Verständnis abspielbar, da die Nutzung heutzutage – z.B. durch das Abspielen von Filmen auf DVD – weit verbreitet ist. Darüber hinaus unterstützt der Mockup Recorder einen einfachen Export von Szenarien in Form einer textuellen Beschreibung des Ablaufs mittels Textbausteinen. Diese Funktion ist nützlich, um aus den Szenarien eine Vorlage für das Erstellen von Use-Cases zu generieren und zum Beispiel in Microsoft Word einzufügen. Als zusätzliche Ausgabeschnittstelle können die im Szenario verwendeten Mockups als .png Bilder gespeichert werden. Zusammen mit dem generierten Text können Szenarios somit auch analog auf Papier – z.B. als Teil einer ausgedruckten Spezifikation – nachvollzogen werden. Als letzte Ausgabeschnittstelle sind die im JSON Format vorliegenden gespeicherten Szenarien zu nennen. Die weitere Verwendung der JSON Dateien setzt zwar die Kenntnis über die Struktur des verwendeten Modells voraus, bietet dafür aber die einfache Möglichkeit, durch eine Software die Inhalte auszulesen. Auf diese Weise können die Szenarien auch als Grundlage für die Erzeugung von automatisierten GUI Tests für JavaFX Anwendungen verwendet werden.

Die mit dem Gluon SceneBuilder erstellten Mockups liegen auch im .fxml Format [R301] vor und bieten somit alle Möglichkeiten von JavaFX zur Gestaltung von Anwendungen. Mittels des SceneBuilders können Mockups schnell erstellt und bearbeitet werden [R302]. Außerdem bietet sich FXML an, damit die Mockups für die Programmierung der realen Anwendung direkt weiterverwendet werden können. Der Mehraufwand zum Erstellen von Mockups relativiert sich, da die .fxml Dateien ohne größere Änderung für die Views der Anwendung benutzt werden können.

Szenario Modell

Das Szenario Modell (siehe Abbildung 4.2) beschreibt die verwendete Struktur für die Speicherung von Szenarien. Es besteht aus einer Hierarchie von Modellklassen im Sinne des Model-View-Controller Patterns, die in einer Baumstruktur angeordnet werden können. Die einzelnen Ebenen bestehen aus Klassen desselben Typs. Jeder Klassentyp wird dabei zum Speichern von unterschiedlichen Daten genutzt. Die Wurzel bildet eine Klasse mit dem Namen *ScenarioModel*. Sie enthält globale Einstellungen für das Szenario und eine Liste von *InteractionModels*. Das *InteractionModel* wird zum Serialisieren von Timeline Items verwendet. Es enthält den Pfad zum jeweiligen Mockup relativ zum Projektordner und eine Liste von Objekten, die das *IAction* Interface implementieren. Diese Objekte dienen dem Speichern von einzelnen Benutzerinteraktionen mit dem zugehörigen Mockup. Das umfasst u.a. Texteingaben, Mausklicks und -bewegungen.

Durch die Verwendung des Object Binding Konzepts können die Modellklassen ohne größeren Aufwand in JSON umgewandelt bzw. geparsed werden. Außerdem

erhalten die erzeugten JSON Dateien die gleiche Struktur. Ein Beispiel für das Aussehen eines Szenarios im JSON Format ist im Anhang enthalten (siehe Abschnitt A.2 „Auszug eines Szenarios im JSON Format“).

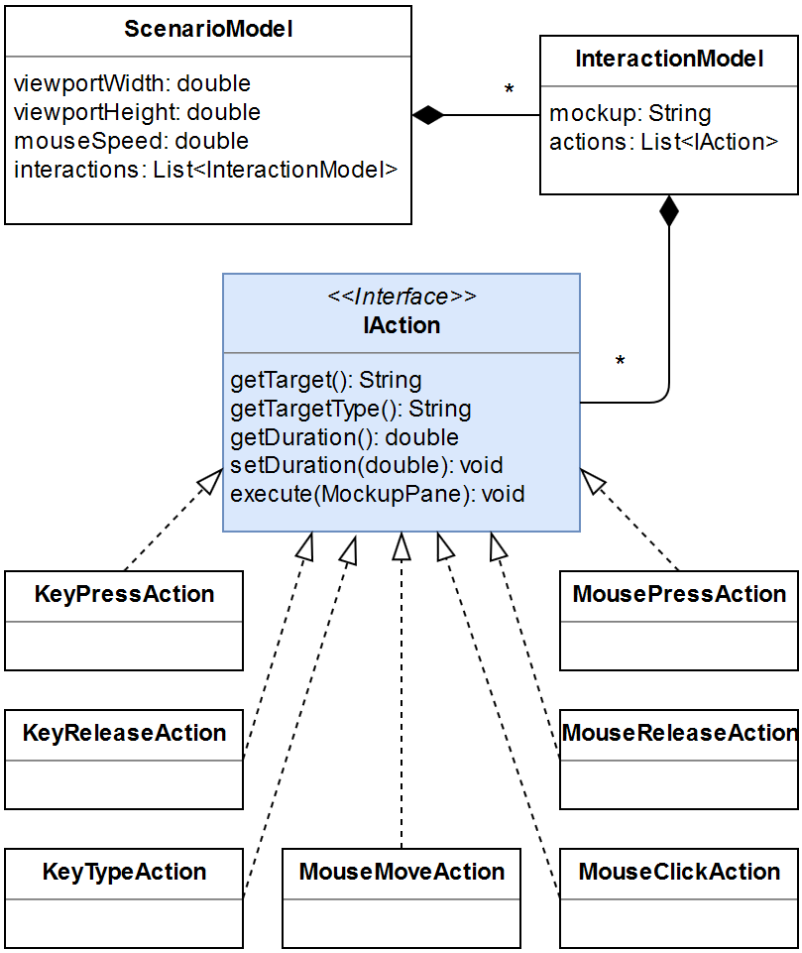

Abbildung 4.2: UML Diagramm des Szenariomodells

4.3 Graphische Benutzerschnittstelle

Die Hauptansicht des Mockup Recorders besteht aus einem Anwendungsmenü und mehreren Bedienelementen, welche durch zwei JavaFX Split Panes in ihrer Größe angepasst werden können (siehe Abbildung 4.3). Das Anwendungsmenü ermöglicht eine schnelle Übersicht der vom Mockup Recorder unterstützten Funktionen, wobei die meisten über Tastenkombationen angesteuert werden können. Die Split Panes sorgen dafür, dass nicht benötigte Elemente keinen Platz auf dem Bildschirm beanspruchen, da der sichtbare Bereich jeweils ensprechend den aktuellen Bedürfnissen angepasst werden kann. Das erste Split Pane teilt die Ansicht in eine obere und eine untere Hälfte auf. Die obere enthält ein weiteres Split Pane, welches auf der linken

Seite den Mockup Chooser und auf der rechten Seite den Mockup Pane anzeigt. Die untere Hälfte zeigt die Recorder Timeline und die zur Bedienung benötigte Recorder ToolBar.

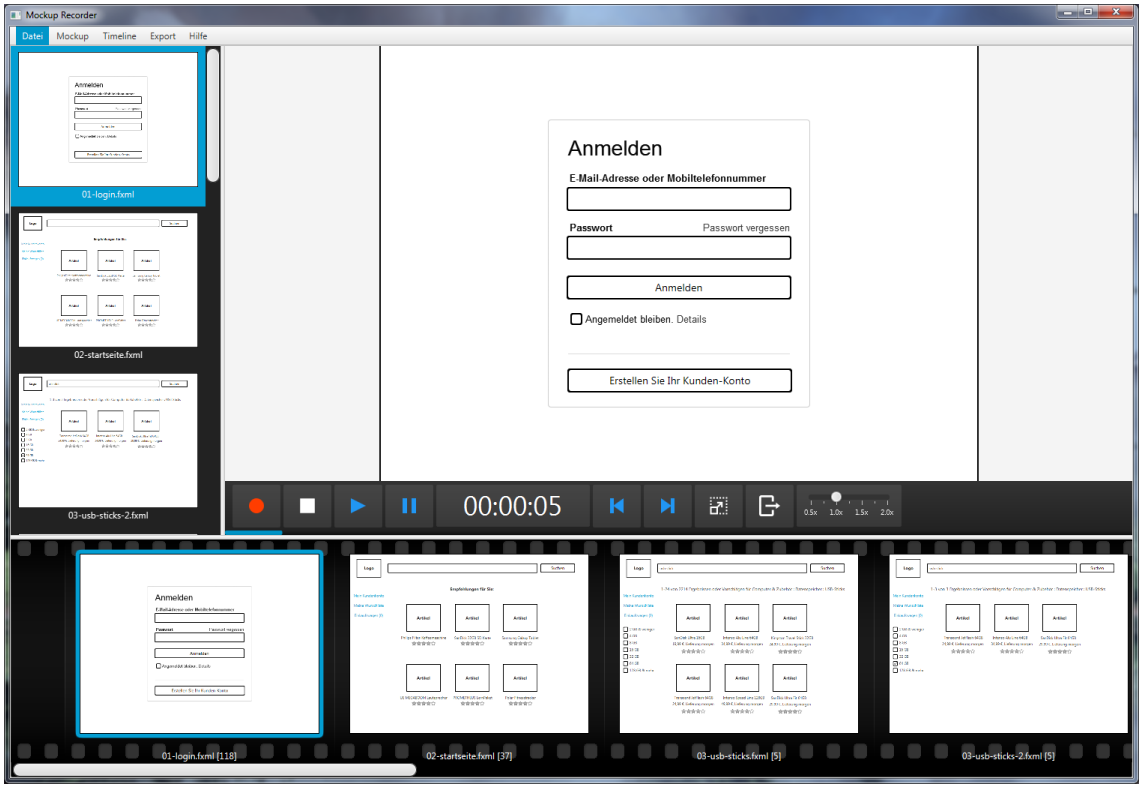

Abbildung 4.3: Hauptansicht des Mockup Recorders

Mockup Chooser

Der Mockup Chooser besteht aus einer Liste aller im Projektordner verfügbaren Mockups im .fxml Format [R401]. Für eine einfache Übersicht werden die Mockups mit Dateinamen und einem Vorschaubild angezeigt. Mit einem Doppelklick wird der ausgewählte Mockup in Originalgröße im Mockup Pane angezeigt [R402]. Ein Rechtsklick öffnet ein Kontextmenü, welches die Möglichkeit zum erneuten Laden der Mockups oder zum Bearbeiten des aktuell ausgewählten Mockups bietet. Mittels *Drag & Drop* können Mockups zur Recorder Timeline hinzugefügt werden [R403].

RecorderTimeline

Die Recorder Timeline stellt die im Szenario enthaltenen Mockups dem zeitlichen Ablauf entsprechend dar [R501]. Um die chronologische Darstellung zu verdeutlichen, wird als Metapher ein Filmstreifen als Hintergrundbild verwendet (siehe Abbildung 4.3). Wie auch beim Mockup Chooser wird der Dateiname und ein Vorschaubild angezeigt, um einen schnellen Überblick zu ermöglichen. Die

Vorschaubilder stellen dabei sinnbildlich die Einzelbilder (Frames) eines Films dar. Mockups können vom Mockup Chooser per *Drag & Drop* zur Timeline hinzugefügt und auch zu einem späteren Zeitpunkt sortiert werden [R502]. Mit einem Klick kann ein Mockup Item ausgewählt und im Mockup Pane dargestellt werden [R503]. Der ausgewählte Mockup wird durch einen blauen Rahmen farblich hervorgehoben.

Mockup Pane

Der Mockup Pane dient zur Darstellung von Mockups und zum Aufzeichnen und Abspielen von Interaktionen. Mockups werden genau wie Views einer fertigen JavaFX Anwendung per *FXMLLoader* geladen und anschließend als Kindelement zum Mockup Pane hinzugefügt [R601]. Auf diese Weise stehen alle JavaFX und sogar selbsterstellte Bedienelemente zur Verfügung. JavaFX enthält ein Eventsystem für Benutzereingaben bzw. Interaktionen ähnlich wie etwa HTML. Durch das Hinzufügen eines Filters zum Mockup Pane können alle Events aufgezeichnet werden [R602]. Das Abspielen bzw. Simulieren von Benutzereingaben funktioniert dementsprechend durch die Erzeugung der aufgezeichneten Events [R602].

Recorder ToolBar

Die Recorder ToolBar ist das zentrale Bedienelement der Hauptansicht. Sie besteht aus JavaFX Buttons (siehe Abbildung 4.4), welche die Aufnahme, Wiedergabe und den Export von Interaktionen steuern [R701]. Als Vorlage für die Buttons dienen die Bedienelemente eines Videorecorders, die durch die typischen Symbole für Play, Pause, Stop, etc. visualisiert werden. Diese Metapher soll dem Benutzer dabei helfen, die Verwendung der ToolBar anhand bereits bekannter Konzepte leichter verstehen zu können. Zusätzlich enthält sie Buttons zum an den Anfang bzw. das Ende der Timeline Springen, Anpassen der Größe des Mockup Panes, eine Stoppuhr zum Anzeigen des Wiedergabefortschritts und einen Schieberegler zum Anpassen der Wiedergabegeschwindigkeit [R702].

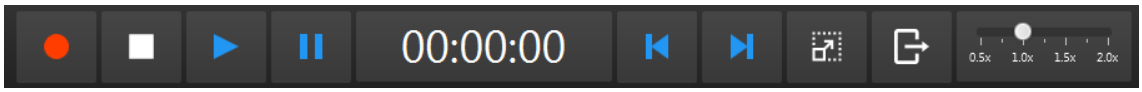

Abbildung 4.4: Recorder ToolBar

Die ToolBar als zentrales Bedienelement ist für die Verwendung des Mockup Recorders von essentieller Bedeutung. Daher wird im Folgenden ein formales Modell in Form von *aussagenlogischen Formeln* erstellt. Daraus werden anschließend Bedingungen für das korrekte Verhalten des Systems abgeleitet.
Mit Hilfe des Zustandsautomaten für die ToolBar (siehe Abbildung 3.2) können leicht Vorbedingungen für alle ToolBar Buttons aufgestellt werden. Ein Zustand kann nur angenommen werden, wenn seine Vorbedingung zu *wahr* evaluiert. Wir definieren für jede Funktionalität eine boolesche Variable und weisen jedem Zustand eine Belegung zu.

Anwendung gestartet:	$\overline{isRecording}$, $\overline{isPlaying}$, $\overline{isExporting}$
Aufnahme:	$isRecording$, $\overline{isPlaying}$, $\overline{isExporting}$
Wiedergabe:	$\overline{isRecording}$, $isPlaying$, $\overline{isExporting}$
Export:	$\overline{isRecording}$, $isPlaying$, $isExporting$

Da das Exportieren eine Erweiterung der Wiedergabe darstellt, können wir daraus eine Randbedingung ableiten. Immer wenn *isExporting wahr* ist, muss auch *isPlaying wahr* sein. Daraus ergibt sich die Implikation:

$$(isExporting \implies isPlaying) \iff (\overline{isExporting} \vee isPlaying)$$

Nun leiten wir für jeden Button eine Vorbedingung in Form einer *aussagenlogischen Formel* ab, indem wir für jede Transition alle erlaubten Ausgangszustände mit einem logischen Oder verknüpfen. Die erzeugten Formeln in *Disjunktiver Normalform* können nun nach der Booleschen Algebra und mit Hilfe der Randbedingung vereinfacht werden.

Record-Start Button:	$\overline{isRecording} \wedge \overline{isPlaying}$
Record-Stop Button:	$\overline{isPlaying}$
Play Button:	$\overline{isRecording} \wedge \overline{isPlaying}$
Pause Button:	$\overline{isRecording}$
Skip-To-Begin & Skip-To-End Button:	$\overline{isPlaying}$
Export Button:	$\overline{isRecording} \wedge \overline{isPlaying}$

Dieses formale Modell kann leicht mit Hilfe des JavaFX Frameworks implementiert werden. Dafür wird die JavaFX Klasse *BooleanProperty* um ein Predicate[12] erweitert, welches ausgeführt wird, bevor der Wert gesetzt wird. Gibt das Predicate *wahr* zurück, darf der Wert gesetzt werden, andernfalls wird eine Exception geworfen (siehe Beispiel A.4.1). Auf diese Weise kann gewährleistet werden, dass nur erlaubte Zustandsänderungen ausgeführt werden können und das System immer in einem definierten Zustand bleibt. Außerdem bleibt, nach dem Prinzip des Information Hidings, die Komplexität des Systems vor dem Aufrufer verborgen. Jede Variable wird als eine *BooleanPropertyWithPredicate* deklariert. Die zuvor aufgestellte Randbedingung kann durch zwei *ChangeListener* ausgedrückt werden. Das Predicate für eine Variable ergibt sich aus den zuvor gebildeten Vorbedingungen (siehe Beispiel A.4.2).

[12] Ein Predicate repräsentiert eine Funktion mit einem Argument, welche einen booleschen Wert zurückgibt. Beispiel in Java: *boolean test(T arg)*.

Ist eine Vorbedingung nicht erfüllt, muss der jeweilige Button deaktiviert sein [R703], [R704], [R705], [R706]. Die Disable-Property der Buttons wird dafür mit der invertierten Vorbedingung mittels JavaFX Binding verknüpft. Ändert sich eine Variable der Vorbedingung, wird der Disable Zustand entsprechend automatisch angepasst. Somit sind die Buttons nur aktiviert und benutzbar, wenn ihre Vorbedingung erfüllt ist.

4.4 Software Architektur

In diesem Abschnitt soll auf die Software Architektur, also die strukturelle Umsetzung des Prototypen, eingegangen werden. Die Hauptziele sind dabei, eine niedrige *Kopplung* und hohe *Kohäsion* zu erreichen, damit einzelne Komponenten leicht erweitert oder für zukünftige Softwareprojekte wiederverwendet werden können [21], [39]. Der modulare Aufbau von Programmen ist ein wichtiges Ziel der professionellen Softwareentwicklung, damit regelmäßig auftretende Problemstellungen durch das Wiederverwenden von Quellcode schneller auf eine geeignete Weise gelöst werden können. Somit kann der generelle Aufwand zur Umsetzung von komplexen Systemen reduziert werden. Außerdem wird durch die häufige Nutzung desselben Quellcodes eine gute Abdeckung durch reale "Testfälle" erreicht, welche die Wahrscheinlichkeit auf ein Fehlverhalten senken und somit auch zu einer höheren Qualität beitragen.

Kopplung beschreibt in der Softwareentwicklung ein Maß für die Ausprägung der Verknüpfung bzw. der Abhängigkeit von Systembestandteilen untereinander. Ziel ist es, diese möglichst niedrig zu halten, um mit geringem Aufwand Änderungen an der Software durchführen zu können, ohne dass dadurch Probleme in anderen Teilen der Software auftreten. Des Weiteren können die Komponenten leichter wiederverwendet werden, wenn sie nur wenige Abhängigkeiten zu anderen Systembestandteilen enthalten. Um dieses Verhalten zu erreichen, werden Schnittstellen (Interfaces) verwendet, welche die einzelnen Bestandteile voneinander trennen, aber weiterhin ein Zusammenarbeiten ermöglichen.

Kohäsion ist ein Maß dafür, wie gut eine Programmkomponente eine logische Aufgabe abbildet. Eine hohe Kohäsion bedeutet also, dass Funktionen, die miteinander nichts zu tun haben, nicht in einer Komponente bzw. einem Modul zusammengefasst werden sollten. Außerdem soll nur eine Komponente innerhalb eines Programms allein für das Lösen einer Problemstellung verwendet und somit die Duplizierung von Quellcode vermieden werden. Dieses Konzept verfolgt das *DRY – Don't Repeat Yourself* Prinzip [15]. Um diese Ziele zu erreichen, die Softwarequalität zu verbessern und die Verständlichkeit des Codes zu erhöhen, können sogenannte *Design Pattern* verwendet werden [21]. Zum einen stellen sie *Best Practices* dar, also eine möglichst gute und einfache Lösung für bekannte Probleme, zum anderen kann mit dem Wissen über die Ziele und Umsetzung der Design Pattern schneller das nötige Verständnis seitens der Entwickler aufgebaut werden.

In den Entwurf des Mockup Recorders sind eine Vielzahl von unterschied-

lichen Design Patterns eingeflossen, welche im Folgenden erläutert werden. Im Abschnitt 4.1 „Technische Grundlagen“ wurde bereits das Model-View-Controller Pattern genannt, das durch die Trennung von Logik, Daten und Darstellung zu einer geringeren Kopplung beiträgt. Dieses Pattern wird heutzutage in allen Bereichen der Softwareentwicklung eingesetzt, um der Software eine grundlegende Struktur zu geben. Das *Mediator* (dt. Vermittler) Design Pattern ist nicht so weit verbreitet, ist aber sehr nützlich, wenn komplexe (Teil-)Systeme kooperieren müssen und trotzdem explizite Abhängigkeiten vermieden werden sollen [21]. Im Mockup Recorder wird das Pattern verwendet, um verschiedene Anwendungsfenster und Controller miteinander kooperieren bzw. kommunizieren zu lassen, ohne konkrete Abhängigkeiten zwischen den Bestandteilen zu erzeugen und somit die Kopplung zu verringern. Dies wird erreicht, indem jede Klasse nur via dem Vermittler Befehle an sogenannte Kollegen senden darf. Ein einfaches Beispiel soll den Vorteil verdeutlichen. Dazu wird ein Graph erstellt, bei dem jeder Knoten eine Komponente und jede Kante eine Abhängigkeit darstellt. Im Worst-case entsteht ein vollständiger Graph (Clique), welcher bei einer Anzahl von n Komponenten exakt $\binom{n}{2} = \frac{n(n-1)}{2} \in \mathcal{O}(n^2)$ Abhängigkeiten untereinander enthält (siehe Abbildung 4.5). Mit einem Mediator erhöht sich zwar die Anzahl der Komponenten auf $n + 1$, es entstehen aber lediglich $n \in \mathcal{O}(n)$ Abhängigkeiten (siehe Abbildung 4.6). Die Verwendung ist daher ab 4 oder mehr Komponenten (ohne den Mediator) empfehlenswert. Allerdings nimmt mit steigender Anzahl von Komponenten die Größe und Komplexität der Mediator Klasse zu.

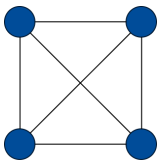

Abbildung 4.5: Vollständiger Graph ohne Mediator

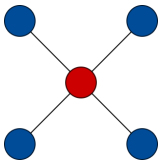

Abbildung 4.6: Graph mit Mediator (rot)

Des Weiteren wird das *Facade* Design Pattern verwendet [21]. Es dient dazu, komplexe Abläufe und Systeme hinter einer – von außen betrachtet – einfachen Schnittstelle zu verbergen. Die Implemetierungsdetails können, nach dem Prinzip des *Information Hidings*, vor dem Benutzer der Schnittstelle versteckt werden und verringern zudem die Kopplung. Konkret wird dieses Pattern zum Verbergen der Komplexität der einzelnen Controller vor dem Mediator und der Videogenerierung durch FFmpeg vom restlichen System verwendet.

Das *Data Access Object* (DAO) Design Pattern wird eingesetzt, um die Herkunft und Verwendung von gespeicherten Daten zu trennen [21]. Die DAO Klasse ist dafür zuständig, die angeforderten Daten zu laden und in einem Modell verfügbar zu machen. Auf diese Weise kann das zur Speicherung genutzte Verfahren leicht ausgetauscht werden, es kann z.B. von JSON Dateien zu einer SQL Datenbank gewechselt werden. Dafür wären nur Anpassungen an der DAO Klasse notwendig, nicht jedoch am verwendetem Modell. Dies führt zu einer Entkopplung der

Aufgabenbereiche (vgl. *Single Responsibility* Prinzip [29]) und einer besseren Wiederverwendbarkeit der einzelnen Komponenten.

Zum Abspielen der Interaktionen werden das *Strategy* und das *Builder* Design Pattern kombiniert [21]. Das Strategy Pattern wird verwendet, um eine abstrakte Aufgabenstellung von der konkreten Implementierung zu trennen. Es wird ein Interface erstellt, welches die Inputs & Outputs eines Algorithmus abbildet. Dadurch ist die Implementierung entkoppelt und kann bei Bedarf leicht ausgetauscht werden. Dieses Pattern wird von der *DefaultActionTimelineBuilder* Klasse implementiert. Sie verfolgt das Konzept des Builder Patterns und dient dazu, die Erzeugung komplexer Objekte von ihrer Verwendung zu entkoppeln. Auf diese Weise kann das JavaFX Timeline Objekt zur Wiedergabe eines Szenarios beim Erstellen manipuliert werden, z.B. um die Abspielgeschwindigkeit zu verändern, ohne dass dies bei der Verwendung beachtet werden muss.

Für die Ausführung der unterschiedlichen Interaktionsarten bei der Wiedergabe eines Szenarios wird das *Command* Pattern verwendet [21]. Es stellt ein gemeinsames Interface für alle Interaktionen dar und erlaubt es, alle konkreten Klassen auf die gleiche Weise anzusteuern. Dafür wird von allen Interaktionen eine *execute()* Methode implementiert, welche den spezifischen Ablauf enthält. Für den Aufrufer bleibt die genaue Funktionsweise verborgen und somit wird die Komplexität verringert.

Letztlich wird das *Factory* Design Pattern zum Erzeugen von Mockup Chooser und Recorder Timeline Items verwendet. Die beiden Bedienelemente stellen Listen von Items dar. Die einzelnen Items müssen allerdings auf Funktionalität zugreifen, welche außerhalb der Verantwortlichkeit der Bedienelemente liegt. Um *Circular dependencies* zu vermeiden, muss die Erzeugung der Items vom Controller verändert werden können. Dieser definiert hierfür eine Factory Methode, welche von den Bedienelementen zur Erzeugung ihrer Items verwendet wird. Auf diese Weise können Abhängigkeiten vom Controller injiziert werden (vgl. *Dependency Injection* Prinzip [17]).

Um zu überprüfen, ob die beschriebenen Design Pattern bzw. Konzepte einen positiven Einfluss auf die Qualität des Quellcodes haben, wurde das *Metrics2*[13] Plugin für die *Eclipse IDE*[14] zum Berechnen von Code-Metriken verwendet. Die Komplexität des Quellcodes wurde mit der *McCabe Cyclomatic Complexity* [23] und das Package Design mit der *Normalized Distance from Main Sequence* [28] Metrik bewertet.

Die *Cyclomatic Complexity* misst die Anzahl der Verzweigungen einer Methode durch u.a. if, else, for Ausdrücke. Sie eignet sich gut als Maß für die Komplexität von Quellcode, da sie abbildet, wie viele unterschiedliche Pfade durch eine Methode führen. Mit zunehmendem Verzweigungsgrad steigt die Anzahl möglicher Pfade bzw. für eine vollständige Pfadabdeckung notwendiger Testfälle exponentiell an, daher sollte die *Cyclomatic Complexity* möglichst gering sein. Ein Wert zwischen eins und

[13] `http://metrics2.sourceforge.net`
[14] `https://www.eclipse.org`

drei stellt eine geringe, zwischen drei und sieben eine mittlere und zwischen sieben und zehn eine hohe Komplexität dar. Größere Werte müssen unbedingt vermieden werden, da der Quellcode in diesem Fall schwer verstehbar ist und nur mit großem Aufwand überprüft oder gewartet werden kann [23]. Für den Mockup Recorder ergibt sich ein durchschnittlicher Wert von $1,47$ mit einer Standardabweichung von $1,2$, welches einer geringen Komplexität entspricht.

Die *Normalized Distance from Main Sequence* Metrik ist ein Maß für gutes Package Design für die objektorientierte Programmierung. Es können Werte zwischen null und eins angenommen werden, wobei null ein gutes Design darstellt. Die Metrik lässt sich durch die Formel $D_n = |A + I - 1|$ berechnen, A bezeichnet die *Abstractness* und I die *Instability* eines Packages. Im Idealfall soll ein Package entweder sehr abstrakt und stabil oder sehr konkret und instabil sein. In der Praxis lassen sich diese Eigenschaften aber nicht immer erreichen. Die Abstractness ergibt sich dabei aus dem Verhältnis von der Anzahl abstrakter Klassen und Interfaces zu der Gesamtzahl von Klassen innerhalb eines Packages. Die Instability berechnet sich aus dem Verhältnis von *Efferent Coupling* (Ce) zu der Summe aus *Efferent* und *Afferent Coupling* (Ca), d.h. $I = Ce/(Ce + Ca)$. Efferent Coupling bezeichnet die Anzahl an Klassen innerhalb eines Packages, die mindestens eine Abhängigkeit zu Klassen außerhalb besitzen. Afferent Coupling wird analog definiert, indem die Worte innerhalb und außerhalb vertauscht werden. Im Fall des Mockup Recorders beträgt der Wert für D_n durchschnittlich $0,263$ mit einer Standardabweichung von $0,239$.

4.5 Verwandte Arbeiten

Die Verwendung von Szenarien und Videos im Bereich des Requirements Engineerings und insbesondere für die Anforderungsanalyse wird bereits seit längerer Zeit erforscht. In diesem Abschnitt werden verwandte Arbeiten bezüglich des Ziels und der Thematik der vorliegenden Arbeit betrachtet und Unterschiede aufgezeigt.

Zur Spezifikation von Interaktionen mit graphischen Benutzerschnittstellen im Rahmen der Anforderungsanalyse werden die Konzepte *Szenarien als Darstellungsform für Interaktionsabläufe* und *Nutzung von Videos als Dokumentationsmedium* kombiniert. Daraus ergeben sich die Schwerpunkte: **Szenarien im Requirements Engineering** und **Videos im Requirements Engineering**.

Szenarien im Requirements Engineering:

Szenarien eignen sich als Hilfsmittel für die Anforderungsanalyse. Dabei unterstützen sie das Erfassen [40], [27] und die Kommunikation [5] von Anforderungen. Nach Stangl [41] und Brun-Cottan et al. [5] eignen sie sich besonders, um im Bereich des Prototyping Interaktionen zwischen Benutzern und Softwaresystemen

möglichst realitätsgetreu wiederzugeben. Zudem repräsentieren sie die funktionalen Anforderungen, welche für den beschriebenen Ablauf benötigt werden [41]. Laut Cockburn ist das Erstellen von Use-Cases ein geeignetes Verfahren zur Dokumentation von abstrakten Szenarien [10]. Allerdings wird zum Erzeugen eines guten Verständnisses durch Stakeholder eine konkretere Darstellung benötigt. Mannio and Nikula [27] beschreiben dafür eine Kombination aus Prototypen mit Szenarien und Use-Cases. Auch Stangl hat sich intensiv im Rahmen seiner Dissertation *A Framework for Scenario-Driven Prototyping* [41] mit der Verwendung von Szenarien im Bereich des Prototyping beschäftigt. Dabei wurde die Software SCRIPT Editor entwickelt, welche in der Lage ist Prototypen auf Grundlage von Grafiken zu erstellen, Szenarios zu definieren und diese als Video zu exportieren. Ein entscheidener Unterschied zur vorliegenden Arbeit ist, dass mit dem SCRIPT Editor erstellte Prototypen keine verwendbaren Bedienelemente, welche auf Benutzereingaben reagieren, bieten und nicht für die finale Anwendung weiterverwendet werden können.

Videos im Requirements Engineering:

Brügge, Creighton et al. [12], [13] verwenden Videos, um eine Vision von geplanten Softwaresystemen zu erstellen. Dadurch soll das Verständnis von Anforderungen und die Kommunikation zwischen Endnutzern, Requirements Engineers und Entwicklern verbessert werden. Zudem sehen sie die Vorteile des videobasierten Requirements Engineerings in der Darstellung von komplexen Szenarien [6]. Dabei werden Videos verwendet, um die textuelle Beschreibung durch eine leicht verständliche Form zu ergänzen. Auch Carter und Karatsolis [9] fordern für eine robuste Dokumentation eine Ergänzung von Text um andere Ausdrucksformen wie Video oder Audio. Nach Brügge et al. eignen sich Videos auch dazu, *Eliciation meetings* aufzuzeichnen und anschließend Aussagen der Stakeholder automatisiert zu extrahieren. Nach Brun-Cottan et al. [5] können Videos die Benutzer einer Software repräsentieren und den Entwicklern so einen Einblick in die Arbeitsweisen bzw. die Verwendung der geplanten Software ermöglichen. Brill et al. [4] haben die Verwendung von Ad-hoc Videos zum Erfassen von Anforderungen und ihre Eignung im Vergleich zu Use-Cases untersucht. Dabei wurde gezeigt, dass selbst mit geringem Aufwand erstellte Videos genauso gut oder besser als Use-Cases geeignet sind, um Missverständnisse in frühen Entwicklungsphasen zu verhindern.

Kapitel 5

Evaluation

In diesem Kapitel wird die Planung, Durchführung und Auswertung der durchgeführten Evaluation in Form eines Experiments betrachtet. Das Vorgehen basiert auf dem Buch *Experimentation in Software Engineering* [43] von Wohlin et al..

5.1 Vorbereitung und Planung

Eine Evaluation dient der kontrollierten Begutachtung und Bewertung eines Sachverhalts. Im Allgemeinen ist das Ziel, eine Aussage zu erhalten, ob und inwieweit eine veränderte Eingabegröße zu einem anderen Ergebnis führt und daher geeignet ist, ein angestrebtes Ziel zu erreichen. Ein Verfahren zur Dokumentation von Interaktionsabläufen ist die Beschreibung als Szenario. Dabei wird der Ablauf in Textform beschrieben und durch Bilder der Benutzerschnittstelle (Mockups) veranschaulicht. Da bei dieser Methode die benötigten Informationen in zwei verschiedenen Medien enthalten sind, können Verständnisprobleme entstehen. Weiterhin können bei der rein textuellen Beschreibung des Ablaufs Probleme durch Ungenauigkeiten auftreten, die dann z.B. bei fehlerhafter Interpretation zu Missverständnissen führen. In der vorliegenden Arbeit wurde ein Softwareprototyp zum Aufzeichnen, Abspielen und Editieren von Interaktionsabläufen erarbeitet. Der Ablauf kann hierbei durch den Export eines Videos dokumentiert und zur Veranschaulichung der textuellen Beschreibung verwendet werden. Im Rahmen der Evaluation wird in einem kontrollierten Experiment das Medium Video der Verwendung von Mockups als Ergänzung der Beschreibung in Textform gegenüber gestellt. Dabei soll untersucht werden, inwieweit die unterschiedlichen Dokumentationsmedien von Szenarien Auswirkungen auf die Verständlichkeit und Nachvollziehbarkeit des Ablaufs und der enthaltenen Informationen haben.

Die These der Arbeit lautet, dass *Videos die textuelle Beschreibung von Interaktionsabläufen auf geeignetere Weise unterstützen als Mockups*. Es wird davon ausgegangen, dass Videos im Vergleich zu Mockups einen Mehrwert bieten, da der Interaktionsablauf eindeutig und vollständig dokumentiert wird und somit verständlicher und leichter nachvollziehbar ist.

Die konkrete Fragestellung der Evaluation lautet, ob der in dieser Arbeit vorgestellte Ansatz der Nutzung von Videos zur Unterstützung der Beschreibung von Interaktionsabläufen in Textform Vorteile im Bezug auf Verständlichkeit und Nachvollziehbarkeit des Ablaufs und weiterer enthaltener Informationen gegenüber der herkömmlichen Unterstützung durch Mockups bietet. Es findet somit keine Evaluation der Benutzung (Usability) des entwickelten Prototypen statt. Vielmehr soll die zugrundeliegende Idee, Videos als Medium zur Spezifikation von Interaktionen mit Benutzerschnittstellen zu verwenden, im Rahmen eines Experiments überprüft werden.

Nach dem Buch *Experimentation in Software Engineering* [43] von Wohlin et al. beginnt der Prozess eines Experiments mit der Festlegung des Geltungsbereichs. Im vorliegenden Fall ist das zu untersuchende Objekt ein Szenario zur Beschreibung von Interaktionsabläufen mit graphischen Benutzerschnittstellen im Bereich des Softwareprototyping. Im Rahmen des Experimentes soll die Eignung zweier Medien zur Unterstützung der textuellen Beschreibung für Szenarien miteinander verglichen werden. Der Fokus liegt hierbei auf der Frage, welches Medium die textuelle Beschreibung besser unterstützt und somit zu einem schnelleren Verständnis des Ablaufs beiträgt. Um die Situation möglichst realistisch zu gestalten, geschieht dies aus der Perspektive eines Softwareentwicklers, welcher auf Grundlage der durch das Szenario dokumentierten Anforderungen die Benutzeroberfläche und Funktionalität einer Software implementieren muss. Diese Perspektive wurde gewählt, da sie gut der Sichtweise eines Informatikers in der Softwareentwicklung entspricht. Dementsprechend wurden Informatiker aus dem universitären Umfeld als Zielgruppe (Subjekt) zur Durchführung des Experiments ausgewählt. Im Folgenden soll die Planung und Durchführung der Evaluation detailliert beschrieben werden. Zunächst wird die zu untersuchende Hypothese formuliert. Zu Beginn der Planung muss davon ausgegangen werden, dass die Eingabegrößen (*unabhängige Variablen*) zu keiner messbaren Änderung der Ausgangsgrößen (*abhängige Variablen*) führen. Dafür wird die sogenannte *Null-Hypothese* (H_0) formuliert.

H_0 : Es gibt keinen Unterschied in der Verständlichkeit bzw. Nachvollziehbarkeit von Szenarien zwischen den verwendeten Medien Video und Mockups bei der Unterstützung der textuellen Beschreibung eines Szenarios.

Der Null-Hypothese gegenüber steht die erwartete Änderung der abhängigen Variablen. Dafür wird eine weitere Hypothese gebildet.

H_1 : Videos tragen zu einem leichteren und schnelleren Verständnis von Szenarien bei und sind somit als Unterstützung zur textuellen Beschreibung für Szenarien besser geeignet als Mockups.

Es gibt keine allgemeingültige Metrik, um die Verständlichkeit und Nachvollziehbarkeit direkt zu messen. Aus diesem Grund muss zunächst festgelegt werden, welche abhängigen Variablen in einem Experiment gemessen und ausgewertet werden können. Eine grundlegende Überlegung ist, dass die Einlese- und Bearbeitungszeit gemessen werden kann. Das heißt die Zeit zum Verschaffen eines Überblicks bzw. zur Beantwortung von Fragen. Weiterhin kann die Anzahl der korrekt beantworteten Fragen leicht ausgewertet werden. Da Verständlichkeit an sich nur subjektiv von den Probanden beurteilt werden kann, eignen sich zudem *Likert-Skalen* zur Bewertung. Dabei wird die Zustimmung zu einer vorgegebenen Aussage gemessen.

Abhängige Variablen

Im Experiment sollen die folgenden abhängigen Variablen gemessen werden:

i) die gemessene Einlesezeit zum Verschaffen eines Überblicks über den Interaktionsablauf

ii) die gemessene Bearbeitungszeit zum Beantworten der Fragen zum Interaktionsablauf

iii) die Anzahl der korrekten Antworten

iv) die subjektive Bewertung der Verständlichkeit des Interaktionsablaufs

v) die subjektive Einschätzung, wie oft für die Beantwortung nachgeschaut werden musste

vi) die subjektiv bevorzugten Medien für die Implementierung

vii) die subjektive Beurteilung des Nutzen / Aufwand Verhältnisses zum Erstellen von Mockups

Aufgrund des verhältnismäßig engen zeitlichen Rahmens für die Durchführung und Auswertung der Evaluation und um eine mögliche Beeinträchtigung der Ergebnisse durch einen Lerneffekt zu verhindern, wurde ein *Between-Subjects* Design für die Durchführung gewählt. Dies bedeutet, dass mit jedem Teilnehmer (Subject) nur ein Test mit jeweils einem Treatment, d.h. entweder Mockup oder Video, durchgeführt wird. Die statistische Signifikanz der Unterschiede in beiden Gruppen wird mit einer *Analysis of Variance*[1] (ANOVA) [43], [36] für jeden Aspekt einzeln beurteilt.

[1] ANOVA bezeichnet eine Varianzanalyse bestehend aus einer Gruppe von statistischen Verfahren zur Überprüfung einer Hypothese in faktoriellen Experimenten.

Bedrohung der Validität

Vor der Durchführung eines Experiments muss man sich mögliche Bedrohungen der Validität bewusst machen, um die Auswirkungen zu minimieren [43]. Hiermit sind Faktoren gemeint, die einen negativen Einfluss auf die Aussagekraft der Evaluation haben könnten. Eine allgemeine Bedrohung der *Conclusion validity* [43] stellt eine zu geringe Teilnehmerzahl dar. Hierdurch wird es erschwert, Muster in den Daten zu erkennen bzw. eine statistisch signifikante Aussage treffen zu können. Zudem kann die *Internal validity* [43] bedroht sein, wenn die gestellten Fragen zu einfach zu beantworten sind und somit nicht das Verständnis, sondern die Vorkenntnisse der Probanden abfragen. Dieser Einfluss wird im vorliegenden Experiment versucht zu minimieren, indem sehr detaillierte Fragen gestellt werden, deren Antworten nicht aus dem generellen Ablauf abgeleitet oder durch entsprechende Vorkenntnisse beantwortet werden können. Darüber hinaus kann die *Construct validity* [43] bedroht sein, da die befragten Probanden keine ausgiebige Praxiserfahrung als Software-Engineer vorweisen können. Allerdings wurde die Zielgruppe des Experiments auf Informatikstudenten begrenzt, welche somit vergleichbare Vorkenntnisse im Bereich des Software Engineerings besitzen sollten. Desweiteren kann die *External validity* [43] bedroht sein, da das Experiment in einer Testumgebung und nicht im Kontext eines realen Softwareentwicklungsprozesses stattfindet. Dies kann u.a. zu einem veränderten Verhalten der Teilnehmer führen. Das für die Evaluation verwendete Video bzw. die Mockups sind durch Nutzung des Mockup Recorders in einem möglichst realitätsgetreuen Umfeld entstanden.

5.2 Durchführung

Das folgende Unterkapitel dient der Darstellung der konkreten Durchführung der Evaluation. Dazu wird zunächst die Population der Probanden, die an der Evaluation teilgenommen haben, beschrieben. Anschließend wird der durchgeführte Ablauf einer Evaluationssitzung mit einem Probanden erläutert.

5.2.1 Population

Insgesamt haben 16 Probanden an der Evaluation teilgenommen. Diese unterteilen sich in 14 Bachelor- und 2 Masterstudenten aus den Studiengängen der Informatik, Technischen Informatik und Wirtschaftsinformatik. Die Probanden befanden sich zum Zeitpunkt der Evaluation zwischen dem 3. und 9. Semester, der Durchschnitt lag bei $5,75$ Semestern. Von $81,25\%$ der Teilnehmer wurde die Veranstaltung *Software-Technik* bereits bestanden. 50% bzw. $43,75\%$ gaben an, *Software-Qualität* bzw. das *Software-Projekt* absolviert zu haben. Ein Proband hat zudem an den Masterveranstaltungen *Labor Agile Software-Entwicklung* und *Architektur für Software-Systeme* teilgenommen. Ein anderer Proband studiert an der Universität Hildesheim Wirtschaftsinformatik und hat somit keine der genannten, allerdings entsprechende Veranstaltungen besucht. Die Teilnehmer des Experiments bestehen

daher aus Personen mit einem starken Bezug zur Informatik und besitzen größtenteils Vorkenntnisse im Bereich des Software Engineerings. Zudem war unter den Probanden ein Legastheniker, welcher aufgrund der Lese-/Rechtschreibschwäche die bildliche der textuellen Darstellung vorgezogen hat.

5.2.2 Ablauf einer Sitzung

Zu Beginn des Experiments wurde allen Probanden das Themengebiet der Arbeit sowie die einzunehmende Sichtweise eines Software Engineers, welcher den gezeigten Interaktionsablauf in Form einer Software implementieren soll, erläutert. Die explizite Fragestellung der Evaluation wurde hierbei jedoch bewusst nicht genannt, um eine Beeinflussung der Ergebnisse zu verhindern. Darauf folgte eine Einarbeitungsphase, um den Teilnehmern Zeit zu geben, sich mit dem vorgegebenen Szenario vertraut zu machen. Währendessen wurde mit einer Stoppuhr die Zeit gemessen, bis der Teilnehmer meinte, den Ablauf verstanden und nachvollzogen zu haben. Im Anschluss wurde der erste Fragebogen (siehe Anhang) zu Details des Ablaufs und anderen enthaltenen Informationen vorgelegt. Dabei wurde erneut die Zeit gemessen, bis alle Fragen beantwortet waren, wobei die Teilnehmer die erhaltene Beschreibung in Textform und das Video bzw. die Mockups zum Nachschauen verwenden durften. Danach wurde der zweite Fragebogen (siehe Anhang) ausgeteilt, welcher aus Likert-Skalen bestand. Zum Abschluss des Experiments wurde den Probanden die Möglichkeit gegeben, Kommentare zu den gestellten Fragen oder sonstige Anmerkungen bzw. zu beachtende Umstände zu äußern.

5.3 Auswertung

In diesem Unterkapitel werden zuerst die Ergebnisse der Evaluation, d.h. die Messdaten, in verschiedenen Darstellungsformen präsentiert. Anschließend folgt eine Bewertung bzw. Interpretation der Ergebnisse.

5.3.1 Ergebnisse der Evaluation

In diesem Abschnitt sollen die Ergebnisse der Evaluation präsentiert werden. Die Bewertung der Ergebnisse folgt im nächsten Abschnitt. Die Messdaten der einzelnen Probanden werden in Säulendiagrammen dargestellt. Zusätzlich werden der (arithmetische) Mittelwert und die Standardabweichung berechnet, welche einen Eindruck von der Verteilung und Streuung geben sollen. Um die Auswirkungen durch Ausreißer zu minimieren, wird zusätzlich der Median angegeben. Außerdem werden die Daten als Boxplot dargestellt, um die Verteilung graphisch zu veranschaulichen. Den einzelnen Probanden wird eine zufällige Zahl (P1 - P16) zugewiesen, welche nicht die Reihenfolge der Durchführung wiedergibt. Für die subjektive Bewertung der Verständlichkeit des Interaktionsablaufs wird kein Diagramm angegeben, da alle Teilnehmer ausnahmslos angegeben haben, dass der

Interaktionsablauf sehr gut verständlich war (vollkommen zugestimmt).

Einlesezeit der Probanden

	Mittelwert	**Standardabw.**	**Median**
Alle	2:30min	53s	2:22min
Video	2:00min	26s	2:08min
Mockups	3:00min	56s	2:46min

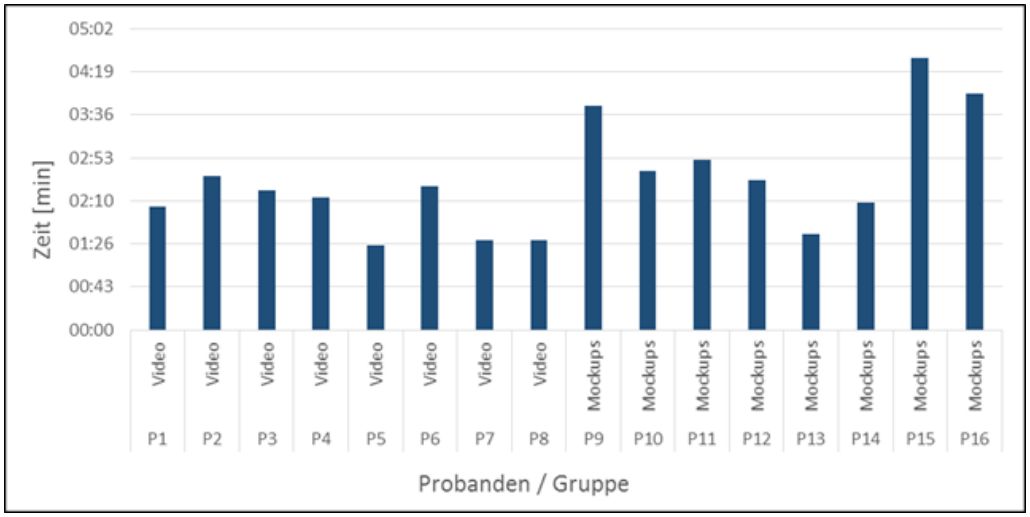

Abbildung 5.1: Einlesezeit der Probanden

Die Teilnehmer in der Video-Gruppe haben im Durchschnitt 2min benötigt, um sich einen Überblick über den Interaktionsverlauf zu verschaffen. Der Median liegt mit 2:08min knapp darüber. Es gibt einige Teilnehmer, die bereits nach ca. 1:30min der Meinung waren, den Interaktionsablauf verstanden zu haben. Daher ist der Durchschnitt niedriger als der Median. Die Standardabweichung ist mit 26s relativ gering. Im Vergleich beträgt die Standardabweichung bei der Mockup-Gruppe 56s. Die Streuung der Messdaten ist hier also deutlich größer. Der Durchschnitt der Mockup-Gruppe liegt bei 3:00min. Gesamt betrachtet, ist die Einlesezeit im Vergleich zur Video-Gruppe um 50% länger. Der Median der Mockup-Gruppe beträgt 2:46min und ist somit niedriger als der Durchschnitt. Zusätzlich werden die Messdaten in einem Boxplot (siehe Abbildung 5.2) graphisch veranschaulicht. Alle Teilnehmer der Video-Gruppe konnten sich schneller einen Überblick verschaffen als über 50% der Mockup-Gruppe. Für die Einlesezeit ergibt die Analyse der Varianz (ANOVA) die Werte $F_{1,14} = 6,63$ und $p = 0,022 < 0.05$, somit sind die Ergebnisse **statistisch signifikant**.

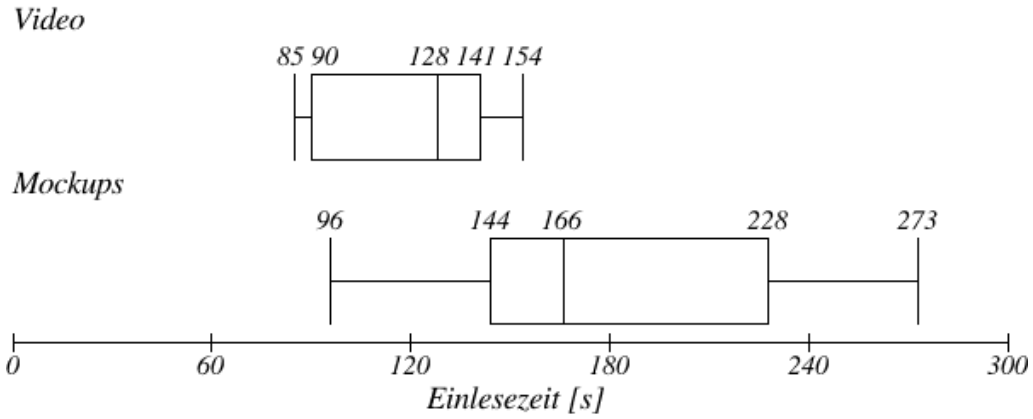

Abbildung 5.2: Boxplot der Einlesezeit in Sekunden

Bearbeitungszeit der Probanden

	Mittelwert	**Standardabw.**	**Median**
Alle	3:27min	45s	3:09min
Video	3:27min	55s	3:05min
Mockups	3:26min	34s	3:28min

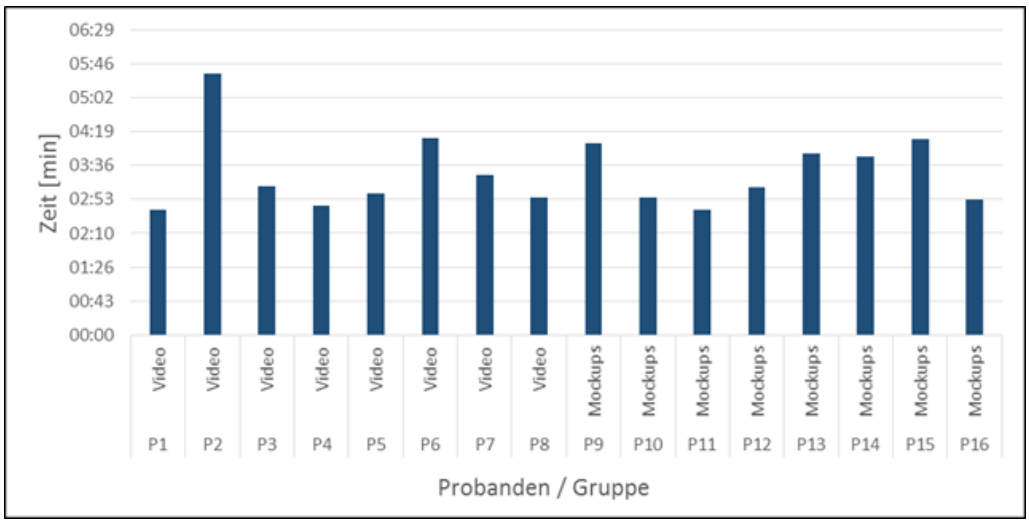

Abbildung 5.3: Bearbeitungszeit der Probanden

Die Mittelwerte der beiden Gruppen liegen dicht beieinander und unterscheiden sich nur um eine Sekunde. In den Messdaten der Video-Gruppe gibt es jedoch einen Ausreißer, der mit 5:33min deutlich mehr Zeit als alle anderen Experimentteilnehmer für die Beantwortung der Fragen benötigt hat. Betrachtet man den Median, liegt die benötigte Zeit in der Video-Gruppe bei 3:05min und somit 23s niedriger als bei der Mockup-Gruppe. Durch die Analyse der Varianz (ANOVA) konnte allerdings **keine** statistische Signifikanz für die Bearbeitungszeit nachgewiesen werden ($F_{1,14} = 0,001$ und $p = 0,9717 > 0.05$).

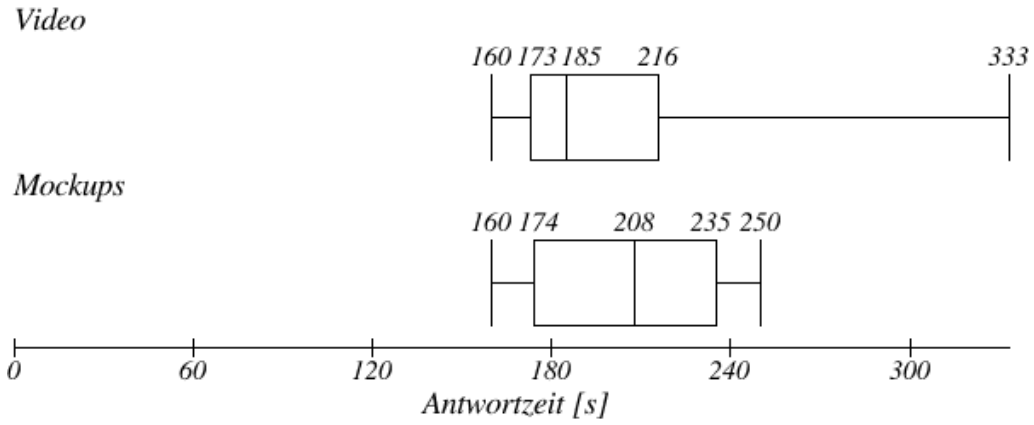

Abbildung 5.4: Boxplot der Bearbeitungszeit in Sekunden

Korrekte Antworten der Probanden

	Mittelwert	**Standardabw.**	**Median**
Alle	8,9375	1,0289	9
Video	8,5	1,1180	9
Mockups	9,375	0,6960	9,5

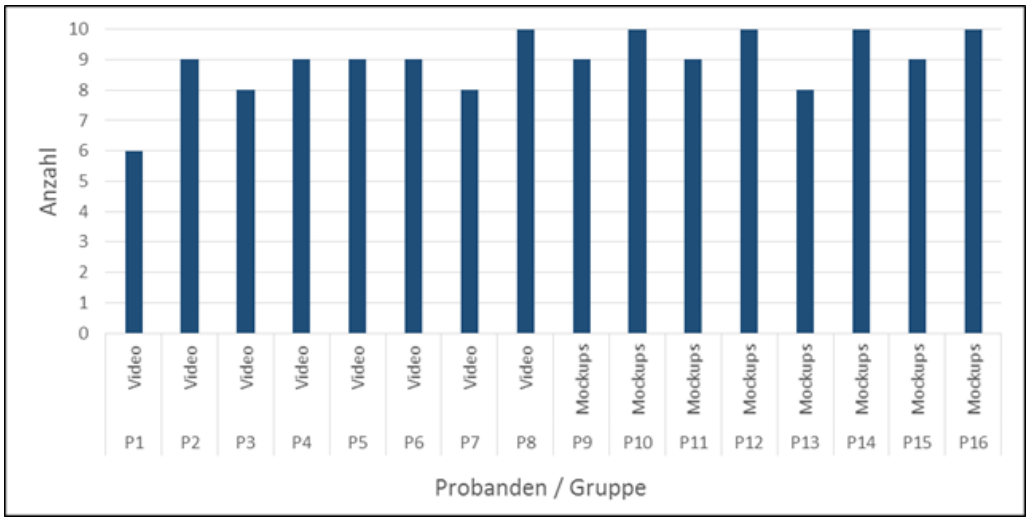

Abbildung 5.5: Korrekte Antworten der Probanden

In der Evaluation wurden zehn Fragen zum Interaktionsablauf und weiteren enthaltenen Informationen gestellt. Die 16 Teilnehmern haben, bis auf eine Ausnahme, zwischen acht und zehn korrekte Antworten angekreuzt. Die Standardabweichung bzw. Varianz ist somit relativ gering. Im Durchschnitt wurde in der Mockup-Gruppe rund eine Frage mehr als in der Video-Gruppe richtig beantwortet. Aus der Analyse der Varianz (ANOVA) folgt für die Anzahl der korrekten Antworten allerdings, dass die Ergebnisse **nicht** statistisch signifikant sind ($F_{1,14} = 3,090$ und $p = 0,1006 > 0.05$).

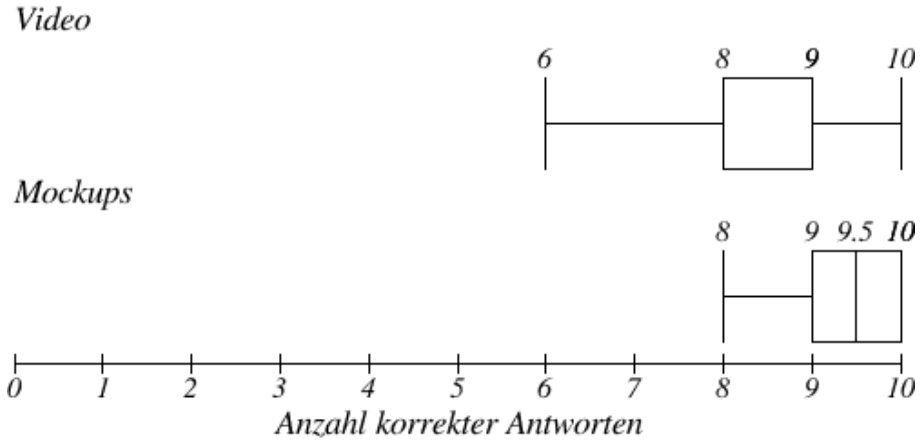

Abbildung 5.6: Boxplot der korrekten Antworten zum Interaktionsablauf

Die folgenden Säulendiagramme beziehen sich auf die im zweiten Fragebogen enthaltenen *Likert-Skalen*. Das erste Diagramm veranschaulicht die subjektive Einschätzung der Teilnehmer, wie oft zur Beantwortung der Detailfragen nachgeschaut werden musste. Die Probanden haben diese Frage am häufigsten mit *Oft* beantwortet, dabei ist der Unterschied zwischen beiden Gruppen sehr gering.

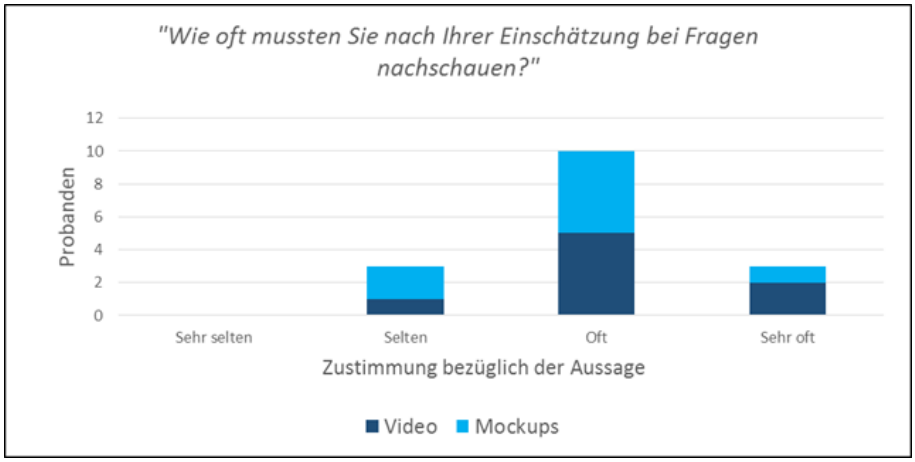

Abbildung 5.7: Einschätzung wie oft für die Beantwortung nachgeschaut wurde

Außerdem wurde die Frage gestellt, welches Medium bzw. welche Methode für die Implementierung bevorzugt wird (siehe Abbildung 5.8). Dabei konnten mehrere der vorgegebene Antwortmöglichkeiten angekreuzt werden. Zudem bestand die Möglichkeit eine eigene Antwort zu ergänzen, dies wurde aber von keinem Teilnehmer genutzt. Alle Probanden haben mindestens ein visuelles Medium, d.h. Video oder Mockups, als gewünschte Grundlage für die Implementierung angekreuzt. Die Optionen Diskussion und Erläuterung wurden in Kombination verwendet. Der Trend geht somit eindeutig in Richtung der visuellen Medien.

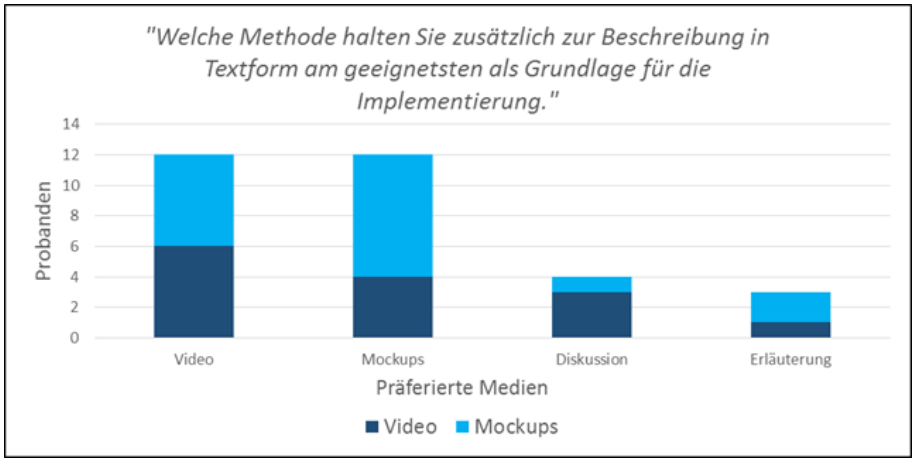

Abbildung 5.8: Präferierte Methode für die Implementierung

Letztlich wurde eine Aussage zum Nutzen/Aufwand Verhältnis von Mockups präsentiert. Durch die Zustimmung der Teilnehmer soll herausgefunden werden, wie sie den zusätzlichen Aufwand zum Erstellen von Mockups bewerten. Mockups sind eine notwendige Voraussetzung zum Aufnehmen von Interaktionen und dem Export als Video. Eine hohe Zustimmung zu dieser Aussage würde bedeuten, dass die potentiellen Benutzer des Prototyps den Aufwand höher als den Nutzen bewerten und somit nicht bereit sind, die Anwendung zu verwenden. Der Trend geht jedoch in die Richtung, dass die Vorteile von Mockups den zusätzlichen Aufwand zum Erstellen rechtfertigen.

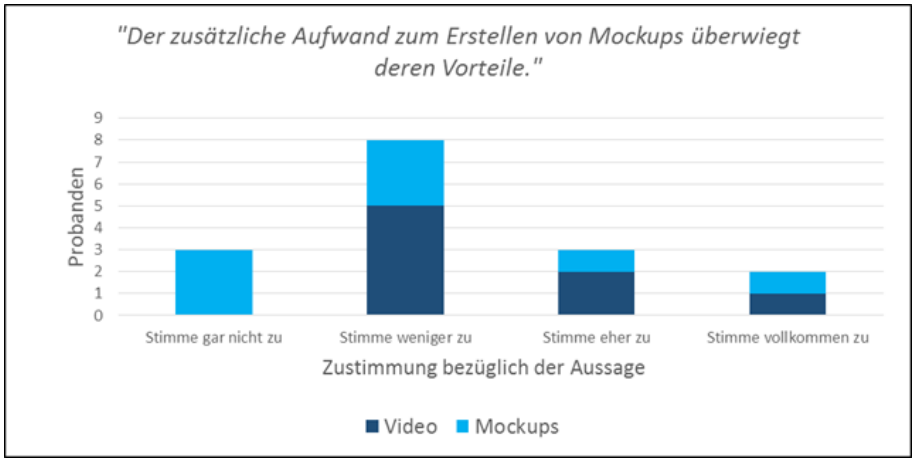

Abbildung 5.9: Bewertung des Nutzen/Aufwand Verhältnisses zum Erstellen von Mockups

5.3.2 Bewertung der Ergebnisse

Abschließend sollen die zuvor dargestellten Ergebnisse der Evaluation bewertet werden. Bezüglich der aufgestellten Hypothese, dass Videos zur Unterstützung von Text geeigneter sind als Mockups, weil sie einen Mehrwert für die Verständlichkeit und Nachvollziehbarkeit bieten, kann keine abschließende Aussage getroffen werden. In der Bewertung der Verständlichkeit haben sich keine maßgeblichen Unterschiede gezeigt. Alle Probanden gaben an, dass der gezeigte Interaktionsablauf sehr gut verständlich war. Demnach kann die Null-Hypothese nicht verworfen werden.

Eine Erkenntnis, die aus der Evaluation gewonnen werden kann, ist, dass Videos zum Verschaffen eines schnellen Überblicks geeignet sind. Diese Aussage wird durch die Auswertung der Einlesezeit unterstützt, welche bei Mockups mit 3:00min im Vergleich zu Videos um 50% länger war. Durch die Analyse der Varianz (ANOVA) konnte hierfür eine statistische Signifikanz nachgewiesen werden ($F_{1,14} = 6,63$ und $p = 0,022 < 0.05$).

Für die Bearbeitungszeit der Fragen hat sich dagegen kein erheblicher, statistisch signifikanter Unterschied ergeben ($F_{1,14} = 0,001$ und $p = 0,9717 > 0.05$). Der Median der Video-Gruppe liegt bei 3:05min und somit 23s unterhalb der Mockup-Gruppe. Für das Experiment geht der Trend also in die Richtung, dass die Fragen mit Video etwas schneller beantwortet werden konnten.

Die Anzahl der korrekten Antworten folgt diesem Trend jedoch nicht. Die Mockup-Gruppe hat geringfügig besser abgeschnitten. Die Unterschiede sind nach der Analyse der Varianz nicht signifikant ($F_{1,14} = 3,090$ und $p = 0,1006 > 0.05$). Eventuell lässt sich das leicht schlechtere Ergebnis für die Video-Gruppe durch die deutlich kürzere Einlesezeit erklären.

Für die subjektive Einschätzung wie oft zur Beantwortung der Fragen zum Ablauf nachgeschaut werden musste, wurde in den meisten Fällen *Oft* angegeben. Dieses Ergebnis bestätigt den hohen Detailgrad der gestellten Fragen, die nicht durch Vorwissen beantwortet werden konnten. Bei dieser Frage kann kein Trend für ein Medium angegeben werden, da der Unterschied zwischen beiden Gruppen sehr gering ist.

Als Grundlage für die Implementierung einer Software halten alle Teilnehmer visuelle Medien in Form von Videos oder Mockups als geeignete Ergänzung zur Beschreibung in Textform. Jedoch fallen die Unterschiede zwischen Video und Mockups sehr gering aus, so dass auch hier kein Trend angegeben werden kann.

Ein Großteil der Befragten sieht allerdings Vorteile in der Verwendung von Mockups und findet den zusätzlichen Aufwand zum Erstellen gerechtfertigt. Da das Erstellen von Mockups im Vergleich zum Aufzeichnen von Interaktionen wesentlich mehr Zeit in Anspruch nimmt, ist hiermit eine wesentliche Grundvoraussetzung für die Nutzung des Mockup Recorders in der Praxis erfüllt. Mit bestehenden Mockups können mit geringem Aufwand Interaktionen aufgenommen und ein Video des Ablaufs generiert werden.

Zusammenfassend lässt sich sagen, dass weitere Experimente folgen müssen, um eine eindeutige Aussage zu erhalten, ob das Medium Video gegenüber her-

kömmlichen Methoden überlegen ist. Zusätzlich muss berücksichtigt werden, dass die Ergebnisse aufgrund der relativ niedrigen Fallzahl von 16 Probanden und der statistisch nicht signifikanten Ergebnisse nur eingeschränkt aussagekräftig sind.

Kapitel 6

Fazit und Ausblick

Im letzten Kapitel wird zunächst eine kurze Zusammenfassung der Arbeit präsentiert. Darauf folgt ein Ausblick auf mögliche weiterführende Arbeiten auf Grundlage der Thematik dieser Arbeit.

6.1 Fazit

In dieser Arbeit wurde ein detailliertes Konzept für die Spezifikation von Interaktionen mit graphischen Benutzerschnittstellen und die Dokumentation als Video erarbeitet. Dafür wurden Mockups zum Skizzieren von graphischen Benutzerschnittstellen mit Szenarien zur Beschreibung von Interaktionsabläufen zwischen Benutzern und Softwaresystemen kombiniert. Das für die Dokumentation verwendete Medium Video eignet sich besonders gut, um komplexe interaktive Abläufe darzustellen und den Entwicklern somit ein detailliertes und konkretes Verständnis der Kundenwünsche zu ermöglichen. Durch die Erweiterung der schriftlichen Dokumentation um Videos können Szenarien auch für Stakeholder auf verständliche und nachvollziehbare Weise veranschaulicht und zusätzlich in der Anforderungsspezifikation verwendet werden.

Für die Unterstützung von Requirements Engineers bei der Spezifikation von Interaktionabläufen mit graphischen Benutzerschnittstellen wurde ein Softwareprototyp – der Mockup Recorder – entworfen und implementiert. Der Requirements Engineer kann in enger Zusammenarbeit mit Stakeholdern Mockups erstellen und Interaktionsabläufe aufnehmen, abspielen und im Verlauf des Entwicklungsprozesses Anpassungen vornehmen. Mockups können sowohl digital mit dem Gluon SceneBuilder in FXML als auch analog auf Papier erstellt werden. Analoge Mockups können anschließend durch Abfotografieren bzw. Einscannen in ein digitales Bild umgewandelt und als Grundlage für einen FXML Mockup verwendet werden. Durch die Verwendung von FXML für Mockups bzw. JSON für Szenarien können leicht Änderungen vorgenommen und somit entsprechend den Vorstellungen der Stakeholder angepasst werden. Die erstellten Szenarien sind ohne zusätzlichen Aufwand als Video bzw. herkömmlich als Bilderfolge mit textueller

Beschreibung des Ablaufs exportierbar. Dadurch kann der Nachteil eines relativ hohen Aufwands zum Erstellen und vor allem zum Bearbeiten von Videos minimiert werden.

Zu Beginn der Arbeit wurde eine ausführliche Anforderungsspezifikation für den Mockup Recorder erarbeitet. Dafür wurden mögliche Anwendungsfälle für die Verwendung zur Erhebung, Dokumentation und Validierung von Interaktionsabläufen beschrieben. Anschließend wurden konkrete Anforderungen an die Rahmenbedingungen, die benötigte Funktionalität bzw. Qualität und eine benutzerfreundliche Verwendung des Mockup Recorders aufgestellt. Der Entwurf und die Implementierung des Softwareprototypen erfolgte in Java unter Beachtung von professionellen Entwicklungstechniken und der Einhaltung von hohen Qualitätsstandards. Für die Architektur wurde eine Vielzahl von Design Pattern verwendet und so eine hohe Erweiterbarkeit und Wiederverwendbarkeit ermöglicht. Die Qualität des Quellcodes wurde fortlaufend durch die Nutzung von unterschiedlichen Metriken bewertet und verbessert.

Zusätzlich erfolgte die Durchführung einer Evaluation mit 16 Probanden. Hierbei konnte die Hypothese, dass das Medium Video besser zur Unterstützung der Beschreibung von Szenarien in Texform als Mockups geeignet ist, nicht statistisch signifikant bestätigt werden. Die Auswertung ergab, dass Videos für das Verständnis des Ablaufs in etwa genauso gut wie Mockups geeignet sind. Einen signifikanten Unterschied zwischen Videos und Mockups hat die Evaluation jedoch ergeben. Die benötigte Zeit zum Verschaffen eines Überblicks über den Interaktionsablauf war bei Verwendung von Mockups höher als bei Videos. Im Durchschnitt wurde mit Mockups 50% mehr Zeit als mit Video benötigt. Videos sind somit besonders gut geeignet, schnell einen Überblick über Interaktionsabläufe zu ermöglichen. Diese Aussage ist für den Requirements Engineering Prozess sehr relevant, da Stakeholder meist nur begrenzte Zeit für die Arbeit mit den Requirements Engineers bzw. Entwicklern zur Verfügung haben.

6.2 Ausblick

In diesem Abschnitt wird ein Ausblick auf mögliche weiterführende Arbeiten im Zusammenhang mit dem Mockup Recorder und der durchgeführten Evaluation gegeben.

Die erste und wohl grundlegendste Idee zur Fortsetzung des Mockup Recorders ist die Erweiterung um Annotationen, die es dem Requirements Engineer erlauben, während der Validierung von bereits aufgezeichneten Szenarien durch Stakeholder schnell zusätzliche Informationen an der entsprechenden Stelle hinzuzufügen. Im Rahmen dieser Arbeit wurde diese Funktionalität nicht benötigt, da die direkte Zusammenarbeit von Requirements Engineer und Stakeholder fokussiert wurde. Im

Rahmen eines Gesprächs kann der Requirements Engineer direkt auf Feedback der Stakeholder eingehen und Anpassungen vornehmen. Daher ist eine Möglichkeit zum Erstellen von Annotationen für diesen Anwendungsfall nicht zwingend erforderlich.

Des Weiteren sollte für eine bessere Unterstützung von weit verbreiteten Verfahren zur abstrakten Beschreibung von Anwendungsszenarien, wie z.B Use-Cases oder auch Storycards, eine Möglichkeit geschaffen werden, diese mit dem Mockup Recorder zu verknüpfen bzw. zu verwalten. Auf diese Weise können die in Text beschriebenen Einzelschritte jeweils am entsprechenden Zeitpunkt bei der Wiedergabe im Mockup Recorder bzw. generierten Video angezeigt werden. Außerdem bietet sich die Verknüpfung mit formalen Modellen wie etwa UML nach dem Vorbild des Software Cinema Tools von Creighton et al. [12], [13] an.

Hinsichtlich der Evaluation hat sich bei der Auswertung ergeben, dass alle Probanden die Verständlichkeit des Ablaufs mit sehr gut bewertet haben. Es wäre interessant, das Experiment mit einem komplexeren und schwerer zu verstehenden Ablauf zu wiederholen, um zu evaluieren, ob sich eventuell dadurch ein Vorteil in Bezug auf die Verständlichkeit für die Verwendung von Videos im Vergleich zu Mockups ergibt. Außerdem konnten die Teilnehmer im Experiment mehrere präferierte Methoden als Grundlage für die Implementierung wählen. Die visuellen Medien wurden eindeutig bevorzugt gewählt, allerdings hat sich kein klarer Trend für ein bevorzugtes Medium (Video oder Mockups) gezeigt. Dürfte nur eine Option gewählt werden, könnte das Ergebnis anders aussehen. Da die Datengrundlage mit 16 Teilnehmern nicht ausgereicht hat, um bei vielen Aspekten eine abschließende Aussage zu treffen, sollte das Experiment mit einer größeren Teilnehmerzahl wiederholt werden. Daraus könnten sich weitere statistisch signifikante Unterschiede zwischen Videos und Mockups ergeben.

Anhang A

Anhang

Fragebogen zum Interaktionsablauf
Name:

1) Welches Produkt wird gekauft?

   ☐ Handy Ladekabel ☐ USB Stick ☐ Sim Karte

2) Wie groß ist die maximale Preisdifferenz der passenden angezeigten Waren?

   ☐ 5€ ☐ 10€ ☐ 15€

3) Welche Farbe hat das Produkt?

   ☐ Blau ☐ Grün ☐ Rot ☐ Keine Angabe

4) Wann wird das Produkt geliefert?

   ☐ Morgen ☐ In drei Tagen ☐ Keine Angabe

5) Wie viel kostet der Versand per Morning-Express?

   ☐ 5€ ☐ 6€ ☐ Keine Angabe

6) Welche Daten werden abgefragt?

   ☐ Straße, Benutzername, Email

   ☐ Kontoinhaber, Postleitzahl, Land

   ☐ Handynummer, Geschlecht, Straße & Hausnummer

7) Wird zuerst nach den Bankdaten oder der Versandadresse gefragt?

   ☐ Zuerst Bankdaten

   ☐ Zuerst Versandadresse

8) Wird die Anrede des Käufers abgefragt (Herr oder Frau)?

   ☐ Ja ☐ Nein

9) Wie viele unterschiedliche Versandoptionen gibt es?

   ☐ 2 ☐ 3 ☐ 4

10) Was kann alternativ zur Emailadresse bei der Anmeldung angegeben werden?

   ☐ Benutzername ☐ Handynummer

Fragebogen 2
Name:

1) Wie oft mussten Sie nach Ihrer Einschätzung bei Fragen nachschauen?

   ☐ sehr selten ☐ selten ☐ oft ☐ Sehr oft

2) Der präsentierte Ablauf war gut verständlich und nachvollziehbar.

   ☐ Stimme gar nicht zu ☐ Stimme weniger zu

   ☐ Stimme eher zu ☐ Stimme vollkommen zu

3) Welche Methode halten Sie zusätzlich zur Beschreibung in Textform am geeignetsten als Grundlage für die Implementierung?

   ☐ Mockups (Skizzen der Benutzeroberfläche)

   ☐ Erläuterung des Ablaufs durch den Requirements Engineer

   ☐ Video des Ablaufs

   ☐ Diskussion mit allen Entwicklern

   ☐ _____

4) Der zusätzliche Aufwand zum Erstellen von Mockups überwiegt deren Vorteile.

   ☐ Stimme gar nicht zu ☐ Stimme weniger zu

   ☐ Stimme eher zu ☐ Stimme vollkommen zu

5) Mockups werden wenn überhaupt nur für komplexe Benutzerschnittstellen benötigt.

   ☐ Stimme gar nicht zu ☐ Stimme weniger zu

   ☐ Stimme eher zu ☐ Stimme vollkommen zu

6) Welcher Lerntyp sind Sie Ihrer Meinung nach?

   ☐ Visuell (Text / Bilder)

   ☐ Auditiv (Zuhören)

   ☐ Motorisch („Learning by doing“)

   ☐ Kommunikativ (Diskussion, Austausch)

A.1 Use-Case Diagramm

Um die Zusammenhänge und Systemgrenzen visuell darzustellen, wurde ein Use-Case Diagramm (siehe Abbildung A.1) erstellt. Das Diagramm beschreibt die benötigten Interaktion zum Erreichen der primären Systemziele zwischen dem Benutzer (links) und dem System (Mitte). Da Mockups auch außerhalb des Systems erstellt und importiert werden können, werden sie als *Actor* auf der rechten Seite dargestellt.

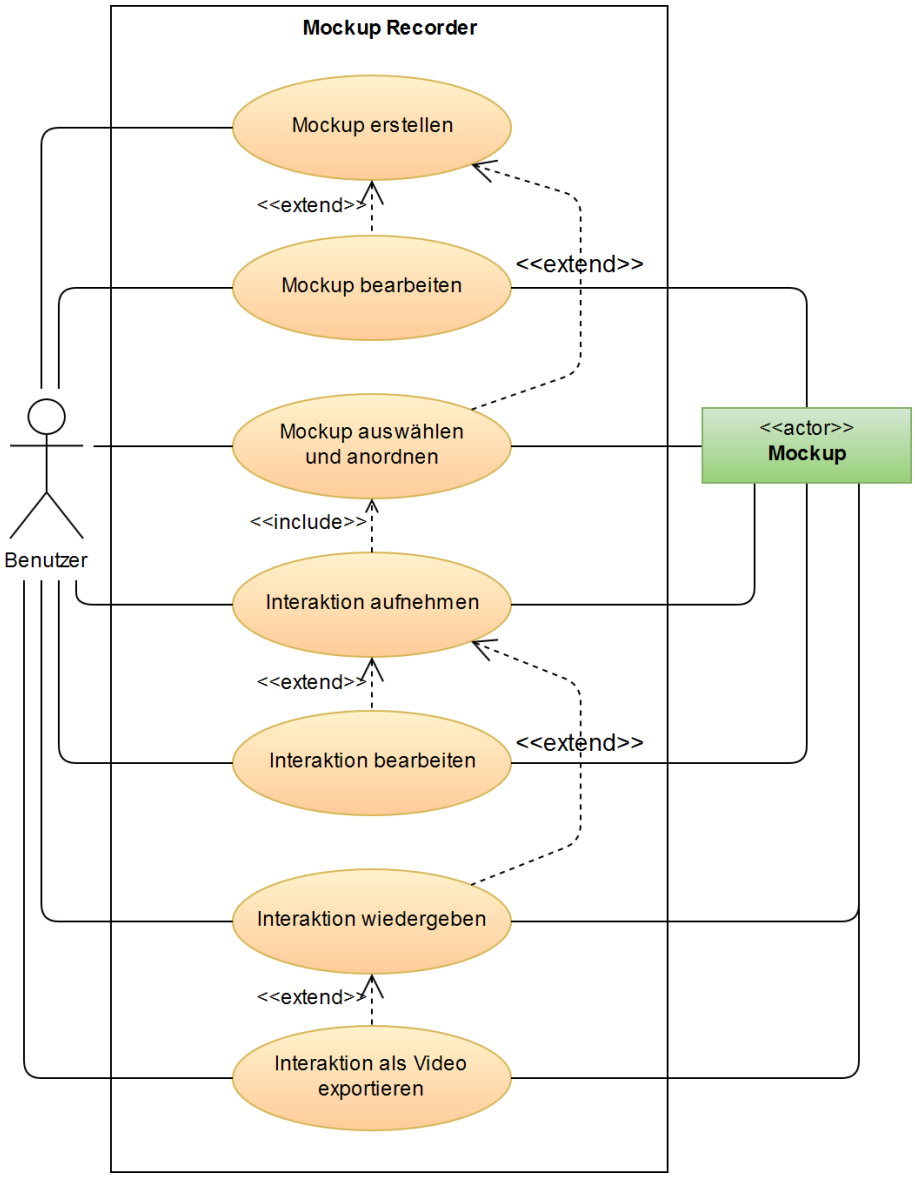

Abbildung A.1: Use-Case Diagramm des Systems

A.2 Auszug eines Szenarios im JSON Format

```
{
  "viewportWidth" : 800.0,
  "viewportHeight" : 600.0,
  "mouseSpeed" : 1.0,
  "interactions" : [ {
    "mockup" : "login.fxml",
    "actions" : [ {
      "@class" : "de.uni.hannover.application.interaction.MouseMoveAction",
      "target" : "#username",
      "targetType" : "TextField",
      "duration" : 1000.0,
      "localX" : 150.0,
      "localY" : 10.0
    },

    ...

    ]
  } ]
}
```

A.3 UML Klassendiagramm

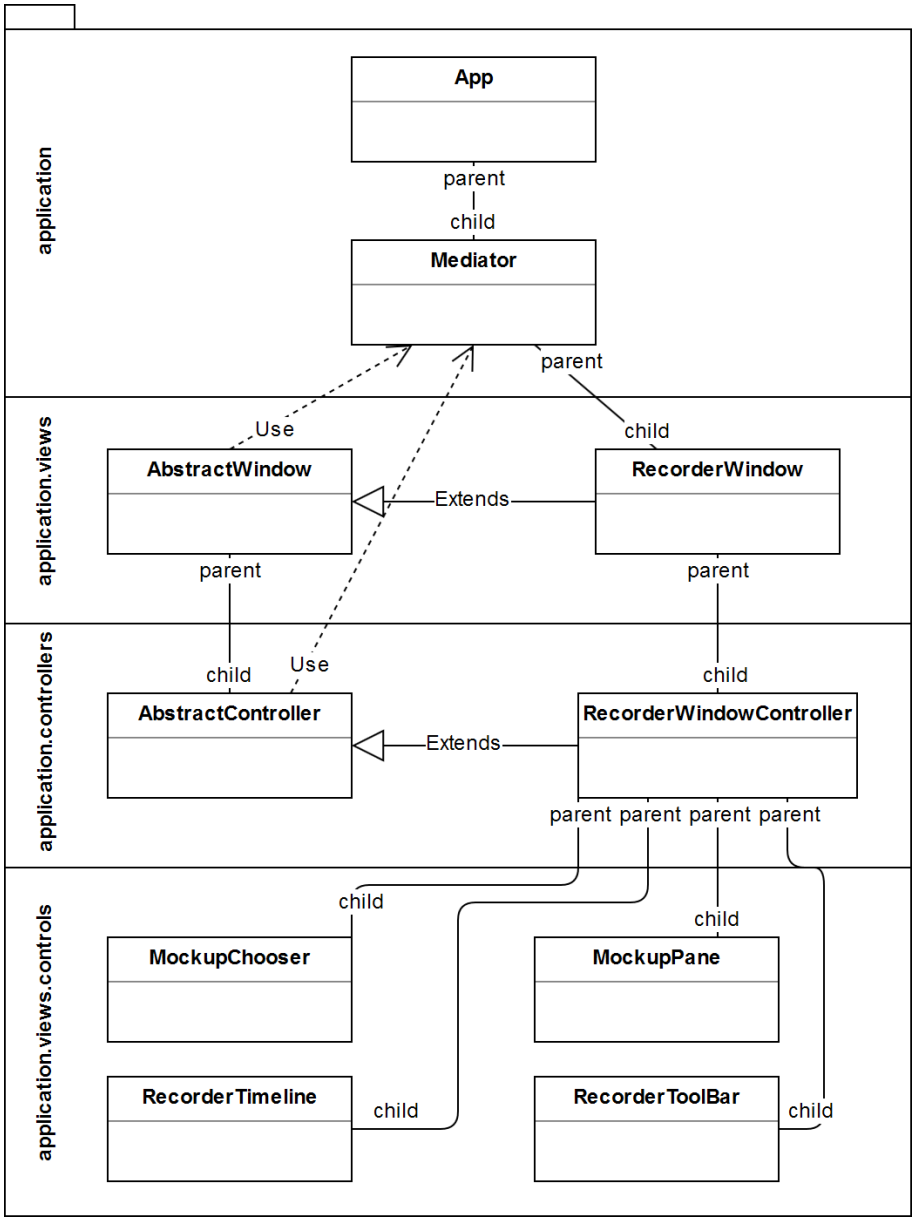

Abbildung A.2: Übersicht über die GUI Klassen und Packages

A.4 Code Beispiele

A.4.1 BooleanPropertyWithPredicate Setter

```
@Override
public void set(boolean newValue) {
    if (!predicate.test(newValue)) {
        throw new IllegalStateException(
            "Pre-condition failed");
    }
    super.set(newValue);
}
```

A.4.2 Beispiel Predicate für eine Variable

```
new BooleanPropertyWithPredicate(newValue -> {
    if (newValue) {
        return !isRecording() && !isPlaying(); // Record Button
    } else {
        return !isPlaying();                   // Stop Button
    }
});
```

Abbildungsverzeichnis

Definitionsverzeichnis

Literaturverzeichnis

[1] Systems and software engineering – Vocabulary. *ISO/IEC/IEEE 24765:2010(E)*, Dec 2010.

[2] International Standard - Systems and software engineering – Life cycle processes – Requirements engineering. *ISO/IEC/IEEE 29148:2011(E)*, Dec 2011.

[3] O. Avci. Warum entstehen in der Anforderungsanalyse Fehler? Eine Synthese empirischer Befunde der letzten 15 Jahre. In *Industrialisierung des Software Management*, volume 139 of *LNI*, pages 89–103. GI, 2008.

[4] O. Brill, K. Schneider, and E. Knauss. *Videos vs. Use Cases: Can Videos Capture More Requirements under Time Pressure?*, pages 30–44. Springer Berlin Heidelberg, Berlin, Heidelberg, 2010.

[5] F. Brun-Cottan and P. Wall. Using Video to Re-present the User. *Commun. ACM*, 38(5):61–71, May 1995.

[6] B. Brügge, O. Creighton, M. Reiß, and H. Stangl. Applying a Video-based Requirements Engineering Technique to an Airport Scenario. In *2008 Third International Workshop on Multimedia and Enjoyable Requirements Engineering - Beyond Mere Descriptions and with More Fun and Games*, pages 9–11, 2008.

[7] E. Börger, B. Hörger, D. L. Parnas, and H. D. Rombach. Requirements Capture, Documentation, and Validation. In *Dagstuhl Report 242*, Schloss Dagstuhl, 1999.

[8] J. M. Carroll. *Scenario-based Design: Envisioning Work and Technology in System Development*. John Wiley & Sons, Inc., New York, NY, USA, 1995.

[9] L. R. Carter and A. Karatsolis. Lessons from Trying to Develop a Robust Documentation Exemplar. In *Proceedings of the 27th ACM International Conference on Design of Communication*, SIGDOC '09, pages 199–204, New York, NY, USA, 2009. ACM.

[10] A. Cockburn. *Writing Effective Use Cases*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1st edition, 2000.

[11] L. L. Constantine and L. A. D. Lockwood. *Software for Use: A Practical Guide to the Models and Methods of Usage-centered Design.* ACM Press/Addison-Wesley Publishing Co., New York, NY, USA, 1999.

[12] O. Creighton. *Software cinema: employing digital video in requirements engineering.* PhD thesis, Technical University Munich, 2006.

[13] O. Creighton, M. Ott, and B. Brügge. Software Cinema-Video-based Requirements Engineering. In *Proceedings of the 14th IEEE International Requirements Engineering Conference*, RE '06, pages 106–115, Washington, DC, USA, 2006. IEEE Computer Society.

[14] J. Eckstein. *Agile Softwareentwicklung mit verteilten Teams.* dpunkt-Verlag, 2009.

[15] K. Eilebrecht and G. Starke. *Patterns kompakt: Entwurfsmuster für effektive Software-Entwicklung.* Springer Vieweg, 4th edition, 2013.

[16] G. Fischer. Symmetry of igorance, social creativity, and meta-design. In *Proceedings of the 3rd Conference on Creativity & Cognition*, C&C '99, pages 116–123, New York, NY, USA, 1999. ACM.

[17] M. Fowler. Inversion of Control Containers and the Dependency Injection pattern. `https://martinfowler.com/articles/injection.html`. [Online; Letzter Zugriff am 04.03.2017].

[18] S. Fricker. *Pragmatic Requirements Communication: The Handshaking Approach.* Shaker Verlag GmbH, Aachen, Germany, 2009.

[19] S. A. Fricker, K. Schneider, F. Fotrousi, and C. Thuemmler. Workshop Videos for Requirements Communication. *Requir. Eng.*, 21(4):521–552, Nov. 2016.

[20] M. Gall, B. Brügge, and B. Berenbach. Towards a Framework for Real Time Requirements Elicitation. In *MERE'06. First International Workshop on Multimedia Requirements Engineering*, Minneapolis, USA, 2006. IEEE.

[21] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software.* Pearson Education, 1994.

[22] T. S. Group. Chaos Report 2009. `https://www.classes.cs.uchicago.edu/archive/2014/fall/51210-1/required.reading/Standish.Group.Chaos.2009.pdf`. [Online; Letzter Zugriff am 11.03.2017].

[23] B. Henderson-Sellers. *Object-oriented Metrics: Measures of Complexity.* Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1996.

[24] S. Houde and C. Hill. What do Prototypes Prototype? *Handbook of HumanComputer Interaction*, 1997.

[25] O. Karras, J. Klünder, and K. Schneider. Enrichment of Requirements Specifications with Videos - Enhancing the Comprehensibility of Textual Requirements. Zenodo, Sept. 2016.

[26] G. Kotonya and I. Sommerville. *Requirements Engineering: Processes and Techniques*. Wiley Publishing, 1st edition, 1998.

[27] M. Mannio and U. Nikula. Requirements Elicitation Using a Combination of Prototypes and Scenarios. In *WER*, pages 283–296, Jan. 2001.

[28] R. Martin. *Agile Software Development: Principles, Patterns, and Practices*. Alan Apt series. Pearson Education, 2003.

[29] R. C. Martin. *Clean Code: A Handbook of Agile Software Craftsmanship*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 1 edition, 2008.

[30] B. Nuseibeh and S. Easterbrook. Requirements Engineering: A Roadmap. In *Proceedings of the Conference on The Future of Software Engineering*, ICSE '00, pages 35–46, New York, NY, USA, 2000. ACM.

[31] K. Pohl. *Requirements Engineering: Fundamentals, Principles, and Techniques*. Springer Publishing Company, Incorporated, 1st edition, 2010.

[32] R. S. Pressman. *Software Engineering: A Practitioner's Approach*. McGraw-Hill, Inc., New York, NY, USA, 7th edition, 2010.

[33] P. Rempel and P. Mäder. A quality model for the systematic assessment of requirements traceability. In *2015 IEEE 23rd International Requirements Engineering Conference (RE)*, pages 176–185, Aug 2015.

[34] S. Robertson and J. Robertson. *Mastering the Requirements Process*. Addison-Wesley Professional, 2nd edition, 2006.

[35] M. Rohs. *Vorlesung: Mensch-Maschine-Kommunikation*. Gottfried Wilhelm Leibniz Universität, 2015.

[36] M. Rohs. *Vorlesung: Mensch-Computer-Interaktion 2*. Gottfried Wilhelm Leibniz Universität, 2016.

[37] C. Rupp and die SOPHISTen. *Requirements-Engineering und -Management: Aus der Praxis von klassisch bis agil*. Carl Hanser Verlag GmbH & Co. KG, 6th edition, 2014.

[38] C. Rupp and R. Ehrlinger. FAQ's Requirements-Engineering und -Management. `https://www.sophist.de/anforderungen/requirements-engineering/faq-requirements-engineering/`. [Online; Letzter Zugriff am 16.02.2017].

[39] K. Schneider. *Vorlesung: Requirements Engineering*. Gottfried Wilhelm Leibniz Universität, 2016.

[40] N. Seyff, N. Maiden, K. Karlsen, J. Lockerbie, P. Grünbacher, F. Graf, and C. Ncube. Exploring How to Use Scenarios to Discover Requirements. *Requir. Eng.*, 14(2):91–111, Apr. 2009.

[41] H. F. Stangl. *Script: A Framework for Scenario-Driven Prototyping.* PhD thesis, Technical University Munich, 2012.

[42] L. Williams and A. Cockburn. Agile software development: it's about feedback and change. *Computer*, 36(6):39–43, June 2003.

[43] C. Wohlin, P. Runeson, M. Hst, M. C. Ohlsson, B. Regnell, and A. Wessln. *Experimentation in Software Engineering.* Springer Publishing Company, Incorporated, 2012.