Institut für Softwaretechnologie
Universität Stuttgart
Universitätsstraße 38
D–70569 Stuttgart

Diplomarbeit Nr. 3135

Annotierung von Use Cases mit Usability Patterns

Ruslana Brull

Studiengang:	Softwaretechnik
Prüfer:	Prof. Dr. rer. nat. Jochen Ludewig
Betreuer:	Dipl.-Inf. Holger Röder
begonnen am:	07. Februar 2011
beendet am:	09. August 2011
CR-Klassifikation:	D.2.1, H.5.2, D.2.2

Zusammenfassung

In den letzten Jahren erlangte die Usability steigende Bedeutung im Softwareentwicklungsprozess. Bisher wird während des Entwicklungsprozesses das Hauptaugenmerk auf die funktionalen Anforderungen gelegt. Dabei werden die nichtfunktionalen Anforderungen sowie die Usability der Software leicht vernachlässigt. Aufgrund des unscharfen Konzeptes der Usability werden viele Usability-Anforderungen während der Entwicklung einer Software nicht einheitlich beachtet. Vielmehr wird unter Usability nur die Gestaltung der graphischen Oberfläche verstanden, obwohl bestimmte Aspekte der Usability einen direkten Einfluss auf die Funktionalität der Software haben. Daher besteht die Notwendigkeit, diese bereits in der frühen Entwicklungsphase mit zu berücksichtigen. Um die eben genannten Aspekte in die Anforderungsspezifikation einzubinden, kann das Konzept der Usability Patterns angewandt werden. Usability Patterns beschreiben die funktionalen Aspekte, die nachweisbar die Usability von Software verbessern.

Das Ziel dieser Arbeit liegt in praktischer Anwendung, Bewertung und Verbesserung des Konzeptes der Aufnahme der Usability Patterns in die Anforderungsspezifikation. Dies wird erreicht, indem die Use-Case-Struktur mit Elementen erweitert wird, die die Annotierung mit Usability Patterns erlauben. Der Prozess der Erstellung einer erweiterten Use-Case-Spezifikation wurde durch die Entwicklung des Use-Case-Editors Tulip, der die Anwendung von Usability Patterns unterstützt, optimiert. Während der einzelnen Phasen der Entwicklung von Tulip wurde das Konzept von Usability Patterns angewandt und bewertet.

Abstract

In recent years usability is the area of focus in software development. In modern practice mainly functional requirements are considered during the development phase and the usability aspects together with other non functional requirements get neglected. Due to the fussy nature of the usability concept it is not followed consistently during the development phase. It is widely known only as a User Interface Design, nevertheless, particular usability aspects have a direct influence on the functional aspects of software. Thus, this presents a need to incorporate these usability aspects into the early development stages. Integrating above mentioned into the requirements specification can be conducted through a method called Usability Patterns. These Patterns describe functional aspects which are known to improve the usability of software.

The scope of work in this thesis is to evaluate and improve the concept of integrating Usability Patterns into use case based functional requirements specification. This is achieved by extending the use case structure with elements which allow their annotation with Usability Patterns. The process of creating an extended use case specification has been optimized by the development and using of a use case editor tool Tulip which supports the application of usability patterns. Throughout the development process of Tulip the concept of Usability Patterns has been applied and evaluated.

Inhaltsverzeichnis

Abbildungsverzeichnis **11**

Tabellenverzeichnis **13**

1 Einleitung **15**
1.1 Motivation . . . 15
1.2 Aufgabenstellung . . . 15
1.3 Aufbau der Arbeit . . . 16
1.4 Sprachliche Konvention . . . 17

2 Grundlagen **19**
2.1 Usability . . . 19
2.1.1 Usability-Merkmale . . . 20
2.1.2 Usability Engineering . . . 22
2.1.3 Definition der Usability-Anforderungen . . . 25
2.2 Usability Patterns . . . 25
2.2.1 Funktionale Usability-Merkmale . . . 25
2.2.2 Aufbau der Usability Patterns . . . 26
2.2.3 Usability-Pattern-Katalog . . . 27
2.2.4 Usability Patterns im Entwicklungsprozess . . . 28
2.3 Use Cases . . . 28
2.3.1 Aufbau von Use Cases . . . 29
2.3.2 Erstellung von Use Cases . . . 31
2.4 Anforderungsspezifikation im Entwicklungsprozess . . . 32

3 Konzept der Erweiterung von Use Cases **33**
3.1 Erstellung einer erweiterten Spezifikation . . . 33
3.1.1 Auswahl der Usability Patterns aus dem Katalog . . . 34
3.1.2 Spezifikation der Anwendung von Usability Patterns . . . 34
3.1.3 Erstellung der Annotationen . . . 34
3.2 Erweiterung der Use-Case-Struktur . . . 35
3.3 Spezifikationsschablonen in Usability Patterns . . . 36
3.4 Ergänzung von Use Cases . . . 36

4 Werkzeugunterstützung für das Konzept **41**
4.1 Anforderungen an das Werkzeug . . . 41
4.1.1 Verwaltung von Use Cases . . . 42

4.1.2 Anzeige der Usability Patterns im Katalog . . . 42
4.1.3 Spezifikation der Anwendung von Usability Patterns . . . 43
4.1.4 Annotierung von Use-Case-Elementen . . . 43
4.1.5 Export der annotierten Use Cases . . . 43
4.2 Evaluierung existierender Werkzeuge . . . 43
4.2.1 Case Complete 2011 . . . 44
4.2.2 HeRA . . . 46
4.2.3 UCEd . . . 47
4.2.4 Remas . . . 48
4.2.5 Zusammenfassung . . . 49
4.3 Evaluierung einer Erweiterungsmöglichkeit . . . 50

5 Realisierung von Tulip **53**
5.1 Erstellung der Spezifikation . . . 53
5.2 Erweiterung der Spezifikation um Usability Patterns . . . 54
5.2.1 Auswahl der Patterns . . . 54
5.2.2 Spezifizierung der Anwendung der Patterns . . . 56
5.2.3 Annotierung der Use-Case-Elemente . . . 58
5.3 Entwurf und Implementierung . . . 59
5.3.1 Datenmodell . . . 60
5.3.2 Komponenten . . . 61
5.3.3 Externe Bibliotheken . . . 62
5.4 Systemtest . . . 63

6 Evaluation von Tulip **65**
6.1 Einsatz von Tulip . . . 65
6.1.1 Spezifizierung der Use Cases . . . 65
6.1.2 Pattern Browser . . . 67
6.1.3 Spezifizierung der Anwendung der Patterns . . . 68
6.1.4 Annotierung der Use Cases . . . 69
6.1.5 Generierung eines Reports . . . 69
6.2 Bewertung von Tulip als Spezifikationswerkzeug . . . 70
6.3 Bewertung der Qualität der Software . . . 71
6.3.1 Zuverlässigkeit . . . 71
6.3.2 Nützlichkeit . . . 72
6.3.3 Bedienbarkeit . . . 72
6.3.4 Prüfbarkeit . . . 73
6.3.5 Änderbarkeit . . . 73
6.3.6 Portabilität . . . 73

7 Bewertung des Konzeptes „Usability Patterns“ **75**
7.1 Anforderungsanalyse . . . 75
7.2 Spezifikation . . . 76
7.3 Entwurf und Implementierung . . . 76
7.4 Testphase . . . 77

7.5 Zusammenfassung 77

8 Zusammenfassung und Ausblick **79**

Literaturverzeichnis **81**

Abbildungsverzeichnis

2.1 Usability im Entstehungsprozess einer Software 20
2.2 Optimierung der Usability-Aspekte im Entwicklungsprozess 21
2.3 Auswirkung der Usability-Techniken 23
2.4 Usability Pattern „Abbruch“ 27
2.5 Einordnung der Usability Patterns 28

3.1 Erweiterung eines Use Cases mit einem Usability Pattern (Angelehnt an [Röd11b]) 35
3.2 Spezifikationsschablonen im Usability Pattern „Gute Standardwerte“ 37

4.1 Eingabemaske für Use Cases in CaseComplete 45
4.2 Eingabemaske für Use Cases in HeRA 46
4.3 Eingabemaske für Use Cases in UCEd 47
4.4 Bearbeitung von Use Cases in remas 49

5.1 Use Case „Projekt öffnen“ 54
5.2 Annotierter Use Case „Projekt öffnen“ 59
5.3 Datenmodell von Tulip 60
5.4 Architektur von Tulip 61

6.1 Projektbaum und Anzeige eines Use Cases in Tulip 66
6.2 Pattern Browser in Tulip 67
6.3 Spezifikation der Anwendung für Usability Pattern „Direkte Validierung“ in Tulip 68
6.4 Bearbeiten eines Use-Case-Ablaufs in Tulip 69
6.5 Ausschnitt aus dem Use Case „System starten“ im RTF-Report 70
6.6 Qualitätsbaum [LL10] 72

Tabellenverzeichnis

2.1 Use Case „System starten“ 30
3.1 Annotierter Use Case „System starten“ 38
4.1 Vergleich der Use Case Editoren 50
5.1 Spezifikation der Anwendung der Usability Patterns in Tulip 58
6.1 Spezifizierung der Use Cases 66

1 Einleitung

1.1 Motivation

Usability-Anforderungen werden im Softwareentwicklungsprozess oft vernachlässigt. Wenn überhaupt, werden Maßnahmen zur Verbesserung der Usability erst spät und oft in einem separaten Prozess durchgeführt. Zudem wird unter Usability häufig nur die Gestaltung der Benutzungsoberfläche verstanden, der Einfluss der funktionalen Aspekte auf die Usability wird dabei nicht beachtet. Dies kann dazu führen, dass die Usability-Anforderungen erst nach der Implementierung der Software auftauchen, z.B. bei der UI-Gestaltung oder in der Testphase. Dabei ist die Umsetzung dieser Anforderungen oft nicht mehr möglich oder mit einem großen Aufwand verbunden. Dies kann verhindert werden, indem man die Usability-Anforderungen bereits in früheren Phasen des Entwicklungsprozesses berücksichtigt, etwa bei der Anforderungsanalyse und -spezifikation.

Das Konzept der Usability Patterns sieht vor, dass Usability-Merkmale, die die Struktur und Funktionalität der Software beeinflussen, bereits bei der Anforderungsanalyse berücksichtigt werden und als Erweiterungen der funktionalen Anforderungen in die Spezifikation der Software einfließen. Dadurch werden diese strukturiert sowie einheitlich beschrieben und können beim Entwurf, der Implementierung und dem Testen der Software berücksichtigt werden.

Usability Patterns beschreiben bewährte Usability-Merkmale, die die Usability der Software verbessern. Jedes Usability Pattern enthält Einweisungen zur Spezifikation des entsprechenden Usability-Merkmals. Patterns werden mittels Annotierung der Use Cases spezifiziert. Dafür müssen Use Cases um entsprechende Elemente erweitert werden. Für den praktischen Einsatz einer solchen Annotierung erscheint eine Werkzeugunterstützung notwendig, existierende Werkzeuge zur Erstellung der Use Cases bieten jedoch keine solche Unterstützung.

1.2 Aufgabenstellung

Im Rahmen dieser Arbeit soll das Konzept der Verwendung von Usability Patterns im Softwareentwicklungsprozess angewendet, evaluiert und verbessert werden. Ziel der Arbeit ist die Entwicklung eines Use-Case-Editors, der den Einsatz der Usability Patterns unterstützt. Zu den Kernfunktionalitäten des Editors zählen insbesondere:

- Verwaltung von Use-Case-Beschreibungen
- Anzeige und Auswahl von Usability Patterns aus einem Katalog

- Spezifizierung von Usability-Merkmalen durch Annotierung von Use-Case-Elementen entsprechend den Anweisungen in den Usability Patterns
- Export der annotierten Use Cases

Im Rahmen der Entwicklung des Use-Case-Editors sollen für diesen geeignete Usability Patterns ausgewählt und die entsprechenden Usability-Merkmale spezifiziert werden. Es sollen auch die für die Spezifizierung notwendigen Erweiterungen der Use-Case-Struktur modelliert werden. Die Verwendung eines existierenden Use-Case-Editors als Ausgangspunkt für die weitere Entwicklung soll im Rahmen dieser Arbeit geprüft und ggf. in Erwägung gezogen werden. Für die Implementierung soll die Programmiersprache Java verwendet werden.

1.3 Aufbau der Arbeit

Kapitel 2 stellt die theoretischen Grundlagen der Arbeit vor. Es werden die Begriffe „Usability“ und „Use Case“ erläutert sowie ein Überblick über das Konzept „Usability Patterns“ gegeben. Zudem wird auf die Rolle der Anforderungsspezifikation im Softwareentwicklungsprozess eingegangen.

Kapitel 3 widmet sich dem Konzept der Erweiterung der Spezifikation und der Use Cases für die Spezifizierung der Usability Patterns. Es werden die Vorgehensweise bei der Erstellung einer erweiterten Spezifikation sowie die Elemente, um die die Use-Case-Notation erweitert wird, beschrieben.

In Kapitel 4 werden Anforderungen an das Werkzeug gestellt, das den Entwickler bei der Erstellung einer erweiterten Spezifikation unterstützen soll. Es werden auch existierende Werkzeuge evaluiert und deren Erweiterungsmöglichkeit beurteilt.

Kapitel 5 beschreibt den Prozess der Realisierung eines Use-Case-Werkzeugs mit Verwendung von Usability Patterns beschrieben.

In Kapitel 6 wird das erstellte Werkzeug einer Evaluierung unterzogen. Es wird der Prozess der Erstellung einer erweiterten Use-Case-Spezifikation mit Hilfe des Werkzeugs beschrieben und anschließend die Eignung der Software als Spezifikationswerkzeugs sowie die Qualität der Software bewertet.

Im 7. Kapitel wird das Konzept „Usability Patterns“ anhand der gewonnenen Erkenntnisse beim Einsatz in einzelnen Phasen des Softwareentwicklungsprozesses evaluiert.

Kapitel 8 bildet mit einer Zusammenfassung der Ergebnisse und einem Ausblick auf mögliche zukünftige Schritte, die der Weiterentwicklung des Konzeptes und des Werkzeuges dienen, den Abschluss der Arbeit.

1.4 Sprachliche Konvention

Zum Zwecke der besseren Lesbarkeit wird in dieser Arbeit für Rollenbezeichnungen entsprechend der sprachlichen Konvention die männliche Form verwendet, wie z.B. „Entwickler“ oder „Benutzer“. Es sind jedoch stets Personen beider Geschlechter gemeint.

2 Grundlagen

In diesem Kapitel werden die theoretischen Grundlagen dieser Arbeit vorgestellt. Zunächst wird der Begriff der *Usability* erläutert und einige Komplikationen aufgezeichnet, die bei der Erhebung und Dokumentation der Usability-Anforderungen in der Praxis auftreten. Darauffolgend werden *Usability Patterns* vorgestellt, die eine konzeptuelle Grundlage darstellen, auf der aufgebaut wird. Anschließend folgt ein Überblick über die Struktur von *Use Cases* und deren Rolle im Entwicklungsprozess, da im Rahmen dieser Arbeit deren Erweiterung um Usability-Merkmale untersucht wird.

2.1 Usability

In dieser Arbeit wird Usability (engl. für Benutzbarkeit)[1] als eine wichtige Teilqualität der Software betrachtet.

Die ISO-Norm 9241-11 [DIN06] definiert Usability bzw. Gebrauchstauglichkeit als

> *„Das Ausmaß, in dem ein Produkt durch bestimmte Benutzer in einem bestimmten Nutzungskontext genutzt werden kann, um bestimmte Ziele effektiv, effizient und zufriedenstellend zu erreichen."*

Diese Definition macht deutlich, dass man Usability nicht ausschließlich als eine Eigenschaft eines Produktes betrachten kann. Vielmehr ist sie von der Benutzergruppe und deren Zielen bezüglich der Software abhängig. Das Empfinden der Benutzer kann man nicht verallgemeinern. Was einem erfahrenen Benutzer als verständlich erscheint, kann für einen anderen ohne Vorkenntnisse sehr verwirrend sein. Deswegen muss Usability bezüglich einer bestimmten Benutzergruppe und deren Ziele bewertet werden.

Usability bestimmt, wie zufrieden die Benutzer mit der Software sind. Somit ist sie auch für die Akzeptanz der Software bei den Benutzern und folglich für die Vermarktung des Produktes entscheidend. Eine gute Usability reduziert Fehler der Benutzer im Umgang mit der Software, was Kosten für die Implementierung, Schulung, technische Unterstützung und Wartung reduziert [Kar90]. Eine schlechte Usability kann dagegen dafür verantwortlich sein, dass eine Software, die eine gute Qualität bezüglich der Funktionalität aufweist, auf die Ablehnung der Benutzer stößt.

[1] Im deutschen Sprachgebrauch hat sich Usability als Begriff durchgesetzt, so dass in dieser Arbeit im Weiteren ebenfalls Usability den Begriffen „Benutzbarkeit" und „Gebrauchstauglichkeit" vorgezogen wird.

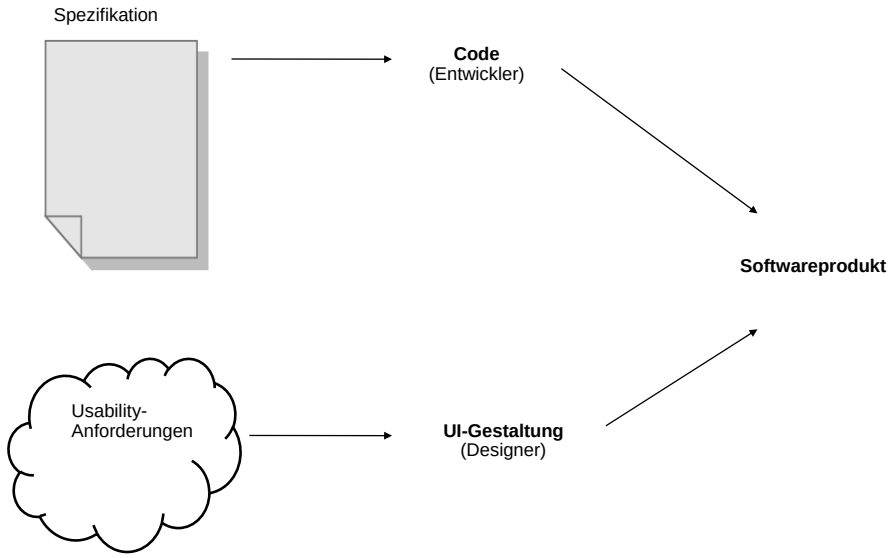


Abbildung 2.1: Usability im Entstehungsprozess einer Software

Ferner werden die einzelnen Merkmale der Usability erläutert und die Techniken vorgestellt, wie diese optimiert werden können.

2.1.1 Usability-Merkmale

Unter Usability-Merkmalen wird oft nur die Gestaltung der Benutzungsoberfläche verstanden. In diesem Abschnitt wird der Begriff der Usability um weitere Merkmale ausgedehnt, die ebenfalls zur Zufriedenheit der Benutzer mit der Software beitragen.

Die Abbildung 2.1 zeigt, wie die Usability üblicherweise in den Softwareentwicklungsprozess eingeordnet wird. Die funktionalen Anforderungen werden in einer Spezifikation festgehalten und von einem Entwickler im Code realisiert. Usability-Anforderungen, wie z. B. Anordnung der Elemente in Dialogen, Anzahl erforderlicher Klicks, Verständlichkeit und Ästhetik der Oberfläche [RF07], werden, wenn überhaupt, von einem UI-Designer in einem separaten Prozess umgesetzt.

Dabei benötigen einige Merkmale, die die Benutzung der Software angenehmer machen, die zusätzliche Funktionalität und Anpassungen in der Struktur der Software. So stellt z.B. die Undo-Funktion (Möglichkeit, ausgeführte Aktionen rückgängig zu machen), eine zusätzliche funktionale Anforderung dar. Es ist nicht mehr möglich, solche Merkmale in

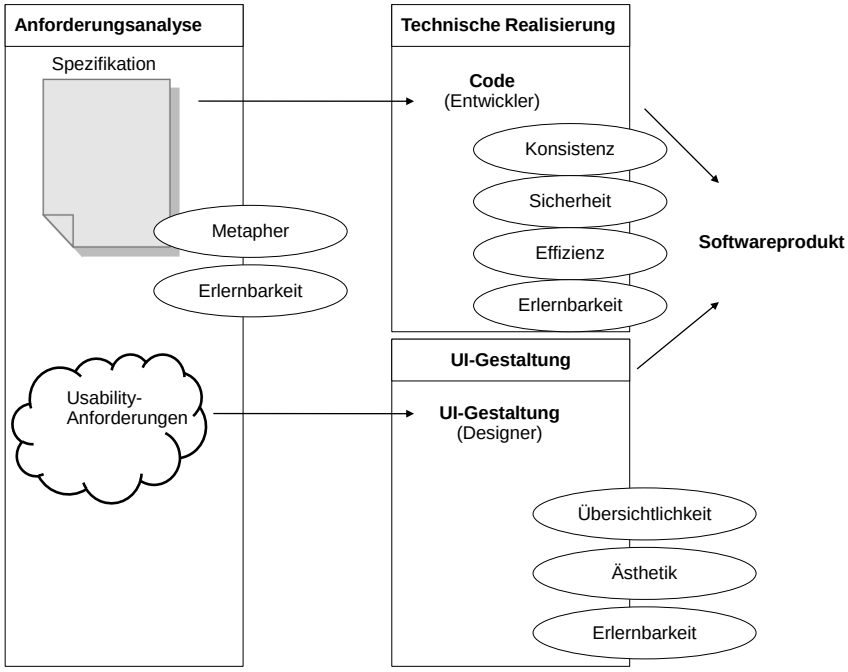


Abbildung 2.2: Optimierung der Usability-Aspekte im Entwicklungsprozess

einem separaten Prozess zu implementieren, da diese bei Entwurf und Realisierung der Software vom Entwickler berücksichtigt werden sollen.

Für die Akzeptanz der Software ist es auch entscheidend, dass diese die Benutzer optimal bei der Erreichung ihrer Ziele unterstützt. Die Usability der Software kann bereits bei der Anforderungsanalyse optimiert werden, indem Ziele und Anforderungen der späteren Benutzer ermittelt werden. Fehler bei der Anforderungsanalyse können dazu führen, dass die später entwickelte Software die Benutzer bei der Erledigung ihrer Aufgaben nicht oder nur unvollständig unterstützt. Solche Mängel lassen sich in der Regel nicht mehr oder nur mit einem großen Aufwand beheben.

Somit ergeben sich drei Bereiche im Softwareenteicklungsprozess, in denen man Usability-Aspekte optimieren kann: Anforderungsanalyse, technische Realisierung und UI-Gestaltung. Die Abbildung 2.2 zeigt diese zusammen mit einzelnen Usability-Merkmalen auf. Die Merkmale werden jeweils den Bereichen zugeordnet, in denen sie optimiert werden können. Dabei wurde folgende Abgrenzung der Usability-Merkmale von [LL10] übernommen:

- **Metapher, Benutzungsmodell:** Szenarien der Interaktion mit dem System sollen für die Benutzer intuitiv sein. Am Besten erreicht man dies, indem man ein Schema nimmt, das dem Benutzer entweder aus dem Alltag oder durch eine andere weit verbreitete Software bekannt ist. Zum Beispiel verstehen die meisten Benutzer, dass eine rote Ampel als Warnung eingesetzt wird.

- **Übersichtlichkeit:** . Alle relevanten Informationen und Elemente sollten so platziert werden, dass sie leicht erkennbar sind.
- **Konsistenz:** Ähnliche Inhalte, z.B. eine Warnung über eine irreversible Operation des Benutzers, sollen im ganzen System ähnlich oder gleich dargestellt werden.
- **Sicherheit:** Es soll sichergestellt werden, dass der Benutzer irrtümlich keinen irreversiblen Schaden anrichten kann, z.B. aus Versehen Daten löschen, die man nicht mehr wiederherstellen kann.
- **Ästhetik:** Die Benutzer sollen die Benutzungsoberfläche als angenehm empfinden.
- **Effizienz:** Der Aufwand für die Bedienung der Software soll von den Benutzern nicht als unnötig hoch empfunden werden.
- **Erlernbarkeit:** Die Benutzer sollen in der Lage sein, den Umgang mit der Software schnell zu erlernen.

2.1.2 Usability Engineering

Usability Engineering befasst sich mit Methoden und Techniken, die im Laufe des Softwareentwicklungsprozesses angewendet werden, um die Usability des Endproduktes zu verbessern.

Den Entwicklungsprozess kann man in folgende Abschnitte unterteilen:

1. **Anforderungsanalyse:** Planung des Projekts, Sammlung der Anforderungen, Modellierung der Prozesse, Erstellung des Angebots.
2. **Spezifikation:** Spezifikation der funktionalen und nichtfunktionalen Anforderungen, Erstellung von Use Cases.
3. **Entwurf und Implementierung:** Erstellung der Architektur und Realisierung der Lösung.
4. **Evaluation:** Komponenten-, Integrations- und Systemtests.

Im Folgenden wird auf die einzelnen Usability-Methoden und -Techniken eingegangen, die man in jeweiligem Abschnitt anwenden kann. Diese werden in der Abbildung 2.3 den entsprechenden Bereichen zugeordnet, in denen sie ihre Wirkung zeigen.

Anforderungsanalyse

Während dieser Phase kann eine gründliche Analyse von Benutzergruppen und dem Nutzungskontext durchgeführt werden. Dabei werden über *Interviews*, *Workshops*, *Befragungen*, *Beobachtungen* und *Aufgabenanalysen* die Arbeitsabläufe und Verhaltensmuster der Benutzer analysiert und ausgewertet. So werden die Anforderungen verschiedener Benutzergruppen bei der Erstellung der Anforderungsspezifikation berücksichtigt.

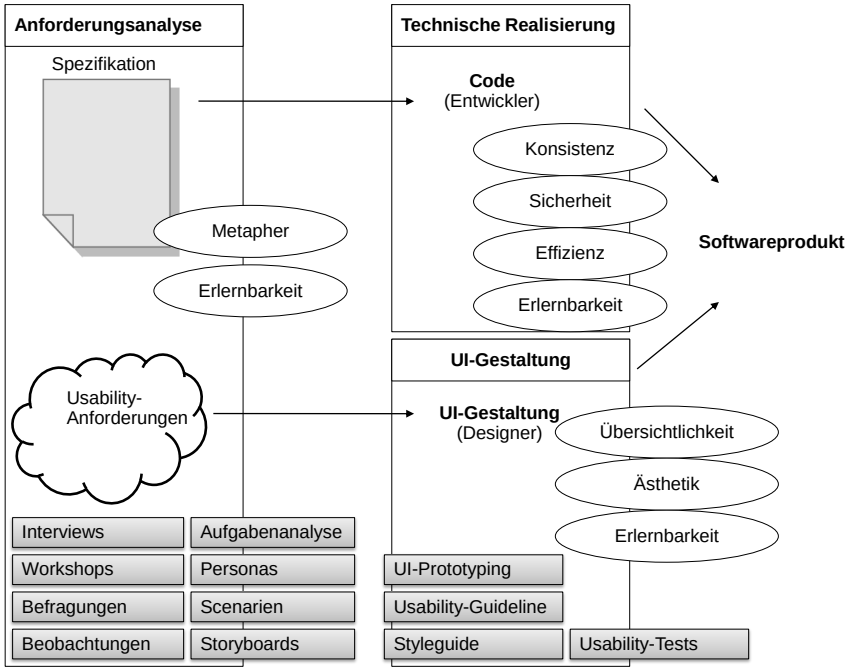


Abbildung 2.3: Auswirkung der Usability-Techniken

Spezifikation

Die Anforderungsspezifikation, die während dieser Phase des Entwicklungsprozesses entsteht, ist ein wichtiges Dokument, das später als Referenz für den Entwurf, die Entwicklung, die Tests und das Handbuch dienen wird. Deswegen ist es sinnvoll, möglichst viele Usability-Anforderungen bereits in der Anforderungsspezifikation festzuhalten. Zahlreiche Techniken, wie z.B. *Personas*, *Szenarien*, *Storyboards*,*UI-Prototyping*, etc. unterstützen die Entwickler dabei.

Bei *Personas* handelt es sich um prototypische Benutzerprofile, die verschiedene Eigenschaften, wie Erfahrung, Ziele, Verhaltensweisen, fachliche Ausbildung, Computerkenntnisse, Erwartungen etc. verschiedener Benutzergruppen beschreiben. *Personas* werden aus den Informationen über zukünftige Benutzer abgeleitet.

Szenarien sind Beispielabläufe, die beschreiben, wie Benutzer mit dem System interagieren werden. *Szenarien* werden zusammen mit den zukünftigen Benutzern entwickelt und bei der Erstellung der Systemanforderungen berücksichtigt.

Storyboards dienen zur Visualisierung von Szenarien und erleichtern die Kommunikation zwischen Auftraggeber und Entwickler. *Storyboards* können, abhängig von dem Präzisionsgrad, sehr unterschiedlich dargestellt werden, von skizzierter Benutzungsschnittstelle bis zur

bildhaften Darstellung der ganzen Geschichte aus einem Szenario, auf der Akteure, System und Umgebung visualisiert werden.

Bei *UI-Prototypen* handelt es sich um die Modellierung der Benutzeroberfläche. Dies kann in Form von Skizzen oder Mockups sein. Abhängig vom Ziel, können UI-Prototypen auch einen Teil der Funktionalität des Systems aufweisen.

Die in dieser Phase angewendeten Techniken dienen zum größten Teil zur Verbesserung der Qualität der Spezifikation. Die erstellten Prototypen und Storyboards dienen später als Referenz für die Gestaltung der Benutzungsoberfläche. Somit zeigen sie ihre Wirkung in den Bereichen der Anforderungsanalyse und der UI-Gestaltung (vergleiche Abbildung 2.3). Für die Berücksichtigung bei der technischen Implementierung fehlt es den Techniken aber an Strukturierung und Anbindung an funktionale Anforderungen.

Entwurf und Implementierung

In der Implementierungsphase wird Usability des Produktes dadurch gesteigert, dass man bei der Entwicklung *Usability Guidelines* und *Styleguides* beachtet. Sie beinhalten Vorschriften für die Konsistenz und Ästhetik der Benutzungsschnittstelle.

Evaluation

Es gibt eine Reihe von Testverfahren, die die Endbenutzer einbeziehen und somit die Usability der Software überprüfen. Man kann das fertige System oder bereits der Prototyp auf die Usability testen. Bei einem formalen *Usability-Test* wird das Verhalten der Benutzer in einem Usability Lab beobachtet und ausgewertet. Die nicht formalen Methoden beinhalten Walkthroughs, Fragebögen, Checklisten und Expertenreviews.

Zusammenfassung

Wie man der Abbildung 2.3 entnehmen kann, können bei der technischen Realisierung einer Software einige Usability-Aspekte optimiert werden. Zum Beispiel werden die Konsistenz und Effizienz einer Software werden durch die Berücksichtigung der Usability-Anforderungen beim Entwurf der Komponenten und des Datenmodells einer Software erreicht. Es fehlt aber an Techniken zur Optimierung der Usability, die den Entwicklern in dieser Phase zur Verfügung stehen. So sind die Entwickler oft bei der Definition der Usability-Anforderungen und bei der Umsetzung und Evaluierung dieser auf sich alleine gestellt.

2.1.3 Definition der Usability-Anforderungen

Ingenieure neigen dazu, sich auf den funktionalen Kern ihres Problems zu konzentrieren [LL10]. Beim Spezifizieren der Funktionalität einer Software werden formale Modelle, wie z.B. Use Cases, eingesetzt, die alle funktionalen Anforderungen bis ins kleinste Detail beschreiben. Die nichtfunktionalen Anforderungen werden dagegen nur am Rande erwähnt und bei der Entwicklung später oft vernachlässigt.

Usability-Aspekte, die im Laufe einer Anforderungsanalyse festgestellt werden, fließen in Form der nichtfunktionalen Anforderungen, höchstens auch Personas und Szenarien, in die Anforderungsspezifikation ein. Sie sind, wie auch andere nichtfunktionale Anforderungen, unstrukturiert und in natürlicher Sprache beschrieben. Insbesondere sind Usability-Anforderungen nicht an funktionale Anforderungen angebunden, was es für die Entwickler schwer macht, diese beim Entwurf und der Implementierung umzusetzen [Röd10]. Ferner werden Usability Patterns als Lösungsansatz für dieses Problem vorgestellt.

2.2 Usability Patterns

Usability Patterns sind strukturierte Beschreibungen der Lösungsansätze für wiederkehrende Usability-Probleme. Sie helfen dabei, die vagen Usability-Anforderungen strukturiert festzuhalten. Das Konzept sieht vor, dass funktionale Merkmale, die zur Usability der Software beitragen, bereits früh im Entwicklungsprozess erhoben und explizit spezifiziert werden.

2.2.1 Funktionale Usability-Merkmale

Einige Usability-Merkmale haben eine große Auswirkung auf die Architektur der Software. Wenn man z.B. erst nach den Usability-Tests in der Evaluationsphase feststellt, dass die Undo-Funktion die Usability des Produktes steigern würde, ist es oft zu spät, dieses Feature in das Produkt einzubauen. Eine Änderung dieses Grades kann das Umbauen ganzer Komponenten erfordern und soll deswegen bereits beim Entwurf des Produktes berücksichtigt werden. Daher ist es sinnvoll, solche Usability-Merkmale als funktionale Anforderungen während der Analysephase zu erfassen und deren Umsetzung durchgehend zu kontrollieren.

Es gab einige Versuche von Usability-Experten, allgemeingültige funktionale Usability-Merkmale zu erfassen. Nach Juristo et.al. zählen z.B. folgende Merkmale zu den funktionalen Usability-Anforderungen [JMSS07]:

- Feedback für die Benutzer über die Vorgänge im System
- Undo-Funktion
- Abbruchmöglichkeit
- Fehlerkorrektur bei den Eingaben
- Assistent für Aktionen mit mehreren Schritten

- Hilfefunktion

Für die erfolgreiche Umsetzung genügt es aber nicht, die funktionalen Usability- Anforderungen zusammen zu fassen und an die Entwickler weiter zu geben. Usability-Experten sind sich einig, dass z.B. die Undo-Funktion die Usability jeder Software steigern würde [JMSS07]. Es gibt aber eine Reihe von produktspezifischen Fragen, die sich ein Entwickler stellen wird, wenn er diese Funktion entwerfen und realisieren muss: „Wie viele Schritte sollen gespeichert werden?“, „Soll es auch eine Redo-Möglichkeit geben?“, „Falls mehrere Fenster verwaltet werden, ist eine globale Undo-Geschichte, oder jeweils eine pro Fenster erwünscht?“, „Wie verhält sich das System, wenn die Wiederherstellung eines vorherigen Standes zu Inkonsistenzen im Datenmodell führt?“ usw. Es ist also wünschenswert, die vagen Usability-Vorgaben in strukturierte funktionale Vorgaben zu übersetzen und zusammen mit der Anforderung zu dokumentieren. Das Konzept der Usability Patterns bietet dafür einen geeigneten Ansatz.

2.2.2 Aufbau der Usability Patterns

Usability Patterns sind bewährte und wiederverwendbare Lösungen, die man in einem bestimmten Zusammenhang einsetzen kann, um gut benutzbare Systeme zu bauen [Röd11b]. Sie beschreiben wiederkehrende Probleme, die erfahrungsgemäß zu schlechter Usability der Software führen können, und Lösungsschablonen für deren Behebung. Ein Pattern beinhaltet strukturierte Informationen darüber, in welchem Zusammenhang und auf welche Weise der Einsatz des Patterns die Usability der Software steigern kann. Jedes Pattern ist auch mit anschaulichen Beispielen versehen.

Auf der Abbildung 2.4 ist zu sehen, wie das Usability Pattern „Abbruch“ aufgebaut ist. Im Folgenden werden einzelne Bausteine eines Patterns erläutert:

- **Name:** Name des funktionalen Usability-Merkmals.
- **Problem:** Beschreibung des Problems, das mit Hilfe des Patterns behoben werden kann.
- **Lösung:** Ausführliche Beschreibung der Problemlösung.
- **Illustration:** Ein Beispielszenario, das das Problem und die Lösung demonstriert.
- **Beispiele:** Screenshots und Beschreibung der Anwendung dieser Lösung in weit verbreiteter Software.
- **Nutzungskontext:** Abgrenzung für den Einsatz des Patterns.
- **Begründung:** Argumentation, warum das Pattern die Usability steigert.
- **Zusammenspiel:** Abhängigkeit von anderen Patterns.
- **Risiken, Nachteile, Kosten.**

Usability Pattern	**Abbruch**
Problem	Benutzer möchten die Aktion, die sie gerade ausführen, abbrechen, ohne Änderungen zu übernehmen.
Lösung	**Erlaube Benutzern, Aktionen abzubrechen.** Verwirf dabei alle Änderungen, die in den bisherigen Schritten der Aktion gemacht wurden. Bei Abbruch einer Aktion soll diese aus Sicht des Benutzers folgenlos bleiben, System und Daten sollen sich also möglichst im Zustand *vor* Ausführung der (abgebrochenen) Aktion befinden. Wenn beim Abbruch einer Aktion umfangreiche Eingaben des Benutzers verworfen werden, weise den Benutzer darauf hin und lasse den Abbruch bestätigen, z. B. durch eine Warnung .
Beispiel	*Mozilla Firefox 4* Der Web-Browser Firefox erlaubt Benutzern, Aktionen abzubrechen. Die Abbildung zeigt den Dialog zum Import von Daten aus anderen Web-Browsern. Der Import kann vom Benutzer abgebrochen werden, ohne das tatsächlich Daten importiert werden.  Abbruch-Möglichkeit in Firefox-Dialog
Nutzungskontext	■ Aktionen, die einen Dialog nutzen: die Möglichkeit zum Abbruch ist für die meisten Aktionen, bei denen Benutzer Eingaben in einem aktionsbezogenen Dialog vornehmen, sinnvoll ■ Aktionen, die Benutzer nicht auf einfache Weise (z. B. über ein Globales Undo) rückgängig machen können.
Begründung	Die Möglichkeit zum Abbruch der aktuellen Aktion nimmt Benutzern die Furcht vor einer Fehlbedienung des Systems. Versehentlich aufgerufene Aktionen können folgenlos abgebrochen werden. Benutzer erhalten durch die Abbruchmöglichkeit eine größere Kontrolle über den Interaktionsablauf. Die Steuerbarkeit des Systems wird somit aus Benutzersicht erhöht.

Abbildung 2.4: Usability Pattern „Abbruch“

2.2.3 Usability-Pattern-Katalog

Usability Patterns sind in einem Pattern-Katalog [Röd11a] gesammelt. Der Katalog beinhaltet zurzeit 20 Patterns und kann erweitert werden. Alle Patterns sind nach dem gleichen Muster aufgebaut und können Referenzen auf andere Patterns beinhalten. Das Pattern „Abbruch“ verweist z.B. im Feld „Lösung“ auf das Usability Pattern „Warnung“ mit dem Hinweis, dass im Fall des Abbruchs einer Aktion, bei dem eine große Menge nicht gespeicherter Daten verworfen werden, eine Warnung sinnvoll ist. Im Feld „Nutzungskontext“ wird die Abgrenzung vom „Abbruch“ zum Usability Pattern „Globales Undo“ erläutert.

Die Auswahl der Patterns hat sich aus den in der Praxis erprobten Mustern ergeben. Der Einsatz der Patterns aus dem Katalog in der Anforderungsanalysephase hilft dabei, die auf Usability bezogenen Schwachstellen der Software frühzeitig zu erkennen und den Entwurf entsprechend den Empfehlungen aus den Patterns anzupassen, um diese vorzubeugen.

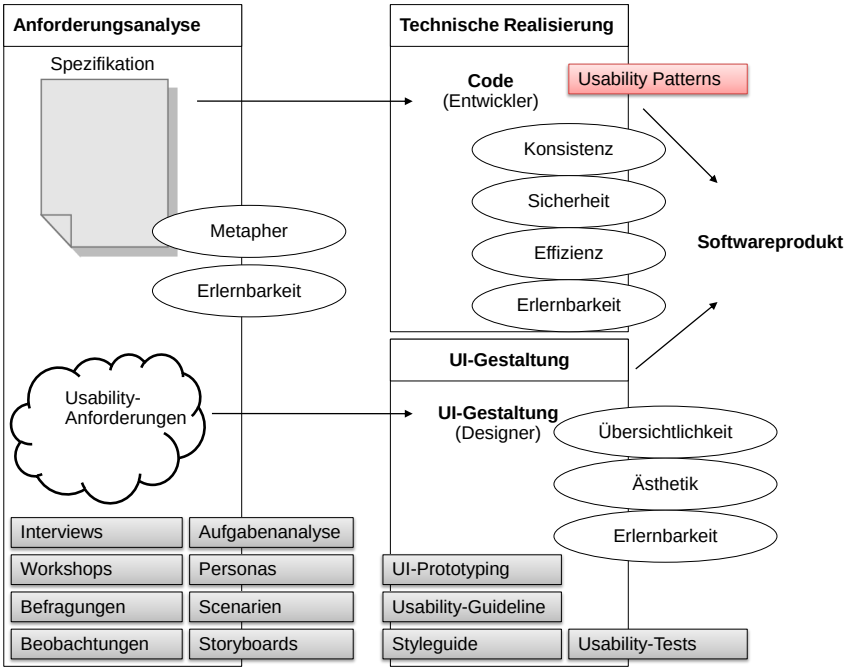


Abbildung 2.5: Einordnung der Usability Patterns

2.2.4 Usability Patterns im Entwicklungsprozess

Im Rahmen dieser Arbeit wird untersucht, wie Usability Patterns zur Optimierung der Usability auf der Ebene der technischen Realisierung eingesetzt werden können. Diese Einordnung wird auf der Abbildung 2.5 dargestellt. Damit die Entwickler beim Entwerfen und Implementieren Usability Patterns berücksichtigen, sollen die Patterns in die funktionale Spezifikation integriert werden, da diese als eine wichtige Referenz in diesen Phasen der Entwicklung dient. Für die Integration der Usability Patterns in die funktionale Spezifikation bieten sich *Use Cases* an, weil diese die Funktionalität der Software aus der Benutzersicht beschreiben.

2.3 Use Cases

Use Cases (engl. für Anwendungsfälle) sind eine bekannte Technik für die Spezifikation der funktionalen Anforderungen. Sie beschreiben die Art und Weise, wie ein Benutzer mit dem System interagieren kann und dienen als eine Beschreibung für das äußerlich sichtbare Systemverhalten [RE-07]. Use Cases beschreiben, wie ein System den Benutzer in seinen Aufgaben, die er mit Hilfe von Software erledigt, unterstützt.

Da Use Cases die Abläufe aus der Benutzersicht in natürlicher Sprache beschreiben [2], braucht man keine besonderen technischen Kenntnisse, um sie zu verstehen. Aus diesem Grund eignen sich die Use Cases gut für die Kommunikation und Abstimmung zwischen Kunden und Entwicklern. Eine vollständige Use-Case-Spezifikation ist auch als Vorlage für Entwurf, Implementierung, Testvorbereitung, Abnahme der Software und auch als Referenz für spätere Erweiterung oder Re-Implementierung geeignet [LL10].

2.3.1 Aufbau von Use Cases

Use Cases wurden bereits in den 80er Jahren von Ivar Jacobson vorgestellt [Jac92]. In all den Jahren hat sich aber kein Standard etabliert, der den Aufbau von einem Use Case vorschreibt. Vielmehr macht die Flexibilität die Use Cases so beliebt. In der Praxis werden abhängig von der Analysephase unterschiedlich stark formalisierte Vorlagen für Use Cases verwendet. Für diese Arbeit wurde eine Vorlage, angelehnt an die Empfehlungen von Cockburn [Coco7], ausgearbeitet. Sie wird am Beispiel eines Use Cases „System starten", welcher in der Tabelle 2.1 abgebildet ist, vorgestellt. Jeder Use Case besteht aus Metadaten, die dessen Eigenschaften erfassen, und Abläufen, die schrittweise die Interaktion von Benutzern mit dem System beschreiben. Die Bestandteile einer Use-Case-Beschreibung werden in diesem Abschnitt genauer erläutert.

Metadaten

- **Name:** Beschreibt das Ziel der Interaktion zwischen System und Akteur.
- **ID:** Muss eindeutig sein.
- **Beschreibung:** Stellt knapp den Zweck des Use Cases dar.
- **Akteure:** Beteiligte Personen oder Objekte. Man unterscheidet zwischen einem Primärakteur, der unmittelbar mit dem System interagiert, und Sekundärakteuren, die das System überwachen, warten und den Primärakteur bei seiner Zielerreichung unterstützen.
- **System:** Das System, das auf Interaktion der Akteure reagiert.
- **Ziel:** Das Ziel oder der Endzustand, den die Akteuren mit der Interaktion verfolgen.
- **Priorität:** Erlaubt die Priorisierung für Use Cases.
- **Ebene:** Beschreibt die Granularität der Interaktion. Use Cases der oberen Ebene beschreiben sehr allgemeine Prozesse, die weiter in Unterpozesse einer unteren Ebene zerlegt werden können. Use Cases der unteren Ebenen beschreiben die technischen Details der Interaktionen. Wie viele Ebenen für die Use-Case-Spezifikation definiert werden, entscheidet das Spezifikationsteam.

[2] In dieser Arbeit wird die UML-Modellierung der Use Cases mittels der Use-Case-Diagramme nicht betrachtet

Name		System starten	
Ziel		Der Benutzer möchte das System starten.	
Akteure		Benutzer	
Primärakteur		Benutzer	
Ebene		Übersicht	
Priorität		hoch	
Normalablauf			
Vorbedingung		Das System ist installiert und lauffähig.	
1	Entwickler	Ruft die Funktion „System starten“ auf.	
2	System	lädt das zuletzt bearbeitete Projekt.	
		Fehler: Kein zuletzt bearbeitetes Projekt verfügbar.	Alternativablauf 2a
Nachbedingung		Das System ist gestartet. Das System zeigt das zuletzt bearbeitete Projekt.	
Alternativablauf 2a			
Vorbedingung		Kein zuletzt bearbeitetes Projekt verfügbar.	
2a1	System	erstellt ein neues leeres Projekt.	
Nachbedingung		Das System ist gestartet. Das System zeigt ein leeres Projekt.	

Tabelle 2.1: Use Case „System starten“

Diese Liste stellt nur die wesentlichen Eigenschaften dar, die einen Use Case beschreiben, und kann bei Bedarf um beliebige Feldern erweitert werden.

Normalablauf

Jeder Use Case verfügt über einen Normalablauf, dieser beschreibt die Art und Weise, auf die der *Primärakteur* das definierte Ziel erreicht. Eine *Vorbedingung* beschreibt den Zustand des Systems, der vor der Interaktion vorhanden sein muss. Der Use Case „System starten“ verlangt z.B., dass das System auf dem Rechner installiert ist, bevor es gestartet wird. Die Interaktion wird in einzelnen *Schritten* beschrieben. Jeder Schritt repräsentiert genau eine Aktion, die entweder von einem Akteur oder vom System ausgeführt wird. Ein Schritt kann auch aus der Ausführung eines anderen Use Cases einer niedrigeren Ebene bestehen. Somit werden die Hierarchien von Use Cases erzeugt. Eine *Nachbedingung* beschreibt den Zustand des Systems nachdem alle Schritte erfolgreich ausgeführt werden. Dieser Zustand entspricht der erfolgreichen Erreichung des Ziels von dem Use Case.

Alternativabläufe

Die Fehlerfälle und Ausnahmen, die bei der Interaktion auftreten können und zu einem Abbruch der normalen Interaktionsfolge führen, werden mittels Alternativabläufe behandelt.

Der Use Case „System starten“ beschreibt z.B. einen Fehlerfall für Schritt 2. Für jeden Fehlerfall wird ein Alternativablauf angelegt, dessen Name aus der Nummer des Schritts abgeleitet wird, in dem der Fehler auftritt. Jeder Alternativablauf verfügt ebenfalls über Vor- und Nachbedingung und wird mit Hilfe von Schritten beschrieben. Ein Alternativablauf stellt entweder eine Fehlerbehebung dar und beschreibt den Weg zurück zum Hauptablauf, in dem der Fehler passiert ist, oder er führt zum Ende der Interaktion ohne dass das Ziel des Primärakteurs erreicht wird.

2.3.2 Erstellung von Use Cases

Abhängig von der Granularität, die für die Use Cases benötigt wird, wird auch die Strategie für deren Erstellung gewählt. Für diese Arbeit wurde die von [ABC03] beschriebene iterative Vorgehensweise übernommen, bei der zunächst ziemlich grob das Gerüst der Use Cases erstellt wird. Dieser wird später in einzelnen Iterationsschritten mit Details angereichert. Die Vorgehensweise wird in vier Abschnitte unterteilt:

1. **Akteure und Ziele definieren:** Zunächst werden alle Akteure, die mit dem System interagieren werden, zusammen mit ihren Zielen, definiert. Die komplette Liste der Ziele soll vollständige funktionale Anforderungen an das System repräsentieren. Für jedes Ziel wird ein Use Case angelegt und mit Metadaten gefüllt.
2. **Normalabläufe ausarbeiten:** Für jedes vorher definierte Ziel wird der Ablauf der Interaktion vom Akteur mit dem System in Details ausgearbeitet. Am Ende des Ablaufs wird das Ziel vom Akteur erfolgreich erreicht. Das Ergebnis dieser Phase ist eine Liste von Use Cases mit jeweils komplett ausgearbeitetem Normalablauf.
3. **Sonderfälle hinzufügen:** Für jeden Ablauf werden Schritte initialisiert, in denen ein Sonderfall (ein Fehler oder eine nicht geplante Situation) auftreten kann. Diese Schritte werden mit der Beschreibung des Sonderfalls versehen.
4. **Behandlung von Sonderfällen:** Für alle vorher definierten Sonderfälle wird ein Alternativablauf angelegt, der beschreibt, wie das System den Sonderfall behandeln muss.

Nach jedem Abschnitt entsteht eine vollständige Use-Case-Spezifikation mit unterschiedlicher Granularität. Diese Spezifikation kann man evaluieren und ggf. korrigieren, bevor man mit dem neuen Abschnitt beginnt und detailliertere Use Cases entwirft. Je nach Situation und Verfügbarkeit von Ressourcen werden Use Cases bis zur gewünschten Granularität entwickelt.

Nachdem die gesamte Funktionalität der Software mittels Use Cases beschrieben wurde, wird die Use-Case-Spezifikation in die Anforderungsspezifikation des Produktes aufgenommen.

2.4 Anforderungsspezifikation im Entwicklungsprozess

Anforderungen an das System beschreiben, was das System leisten soll und in welcher Qualität. Anforderungen werden in einer Anforderungsspezifikation festgehalten. Dabei unterscheidet man zwischen funktionalen und nichtfunktionalen Anforderungen. Funktionale Anforderungen beschreiben gewünschte Funktionalitäten des Systems und dessen Verhalten. Die nichtfunktionalen Anforderungen beschreiben die Qualität, in welcher die geforderte Funktionalität zu erbringen ist [RE-07]. In der Anforderungsspezifikation werden die funktionalen Anforderungen oft mit Hilfe von Use Cases beschrieben, und die nichtfunktionalen mittels natürlicher Sprache. Die Usability-Anforderungen liegen im Grenzbereich und können sowohl die Funktionalität (z.B. Undo-Funktion) als auch die Qualität (z.B. Vorgaben für die UI-Gestaltung) beschreiben. Diese werden aber üblicherweise zu den nichtfunktionalen Anforderungen gezählt und entsprechend beschrieben.

Use Cases beschreiben die Reaktion des Systems auf die Aktionen der Benutzer bei der Interaktion. Somit dient eine detaillierte Use-Case-Spezifikation als eine vollständige Referenz für die Entwickler bei Entwurf und Implementierung der Funktionalität der Software. Use Cases bieten den Entwicklern alle Black-Box-Verhaltensanforderungen an die Software, ohne ihre Freiheit bei der Methodenwahl einzuschränken. Dabei muss man beachten, dass die Use Cases lediglich als Referenz dienen und nicht direkt in den Entwurf einer Software umgewandelt werden. Der Entwickler liest die Use-Case-Spezifikation und entwirft anhand der Vorgaben die Software. Später kann dieser Entwurf mit Hilfe der Use-Case-Spezifikation auf die Erfüllung der Anforderungen überprüft werden.

Im Unterschied zu klar formulierten Vorgaben aus den Use Cases, denen man Qualitätskriterien für die Evaluation des Produktes entnehmen kann, sind die nichtfunktionalen Vorgaben nur vage formuliert, so dass der Entwickler bei der Evaluation der Erfüllung dieser auf sich allein gestellt ist.

Im Rahmen dieser Arbeit wird die Möglichkeit der Erweiterung der Use Cases um Usability-Vorgaben evaluiert. Formal beschriebene Usability-Merkmale sollen eine bessere Referenz für den Entwickler darstellen, als unklar definierte nichtfunktionale Anforderungen.

3 Konzept der Erweiterung von Use Cases

In diesem Kapitel wird das Konzept der Integration der Usability Patterns in die Use-Case-Spezifikation ausgearbeitet. Es sieht vor, dass bereits in der Spezifikationsphase die für die Software relevanten Usability Patterns aus dem Katalog ausgewählt und in die Use-Case-Spezifikation eingearbeitet werden. Durch diese Maßnahme entsteht ein neues Dokument, in dem Usability-Vorgaben strukturiert, einheitlich und eng verzahnt mit der Beschreibung der Funktionalität spezifiziert sind.

Die Abbildung 3.1 stellt schematisch dar, wie die Patterns in die Spezifikation eingebunden werden. Die Erstellung einer mit Patterns erweiterten Spezifikation erfolgt in drei Schritten:

1. Eine vollständige Use-Case-Spezifikation wird erstellt.
2. Aus dem Katalog werden passende Usability Patterns ausgewählt. Für jedes Pattern wird die Anwendung spezifiziert. Dabei werden globale Anforderungen für das Pattern definiert.
3. Für jedes spezifizierte Pattern werden einzelne Use-Case-Elemente annotiert und die entsprechenden lokalen Anforderungen angelegt.

In diesem Kapitel werden die Schritte für die Erstellung einer erweiterten Spezifikation ausführlich erläutert, die Erweiterungen für die Use Cases und Spezifikation beschrieben, die dafür notwendig sind, und die Anwendung dieses Verfahrens in der Praxis angesprochen.

3.1 Erstellung einer erweiterten Spezifikation

Usability Patterns beschreiben Vorgaben, deren Erfüllung die Usability der Software verbessert. Es erscheint also wünschenswert, diese Vorgaben in die Anforderungsspezifikation aufzunehmen und in späterem Entwicklungsprozess systematisch zu berücksichtigen. Viele in Usability Patterns definierte Vorgaben betreffen nicht nur eine, sondern mehrere Funktionen der Software, können also an mehreren Stellen eingesetzt werden. Deswegen ist es sinnvoll, sowohl globale Vorgaben aus dem Pattern in der Anforderungsspezifikation zu definieren, als auch alle Stellen der Interaktion der Benutzer mit dem System zu identifizieren, an denen diese Patterns eingesetzt werden. Somit ergeben sich zwei Aspekte für die Spezifikation der Usability Patterns: Anwendungsspezifikation für Patterns und Annotationen mit individuellen Vorgaben. Für die Anwendungsspezifikationen der Usability Patterns wird ein neues Kapitel in der Anforderungsspezifikation der Software angelegt. Für die Annotationen werden Use-Cases um weitere Elemente ergänzt. Im Folgenden werden

der auf der Abbildung 3.1 dargestellte Prozess der Erstellung einer erweiterten Spezifikation beschrieben.

3.1.1 Auswahl der Usability Patterns aus dem Katalog

Zunächst werden Usability Patterns ausgewählt, die später in die Spezifikation aufgenommen werden. Die Auswahlkriterien für die Patterns basieren darauf, ob die vom Pattern beschriebene Funktionalität für das System sinnvoll ist und ob diese von dem Kunden erwünscht ist. Der Katalog [Röd11a] bietet dem Use-Case-Entwickler Unterstützung bei der Auswahl, indem für jedes Pattern Anwendungsbeispiele und Beispielszenarien angegeben werden. Für jedes ausgewählte Pattern wird ferner die Anwendung spezifiziert.

3.1.2 Spezifikation der Anwendung von Usability Patterns

Für jedes eingesetzte Pattern wird eine Beschreibung verfasst, die das Ziel der Anwendung erläutert. Außerdem werden globale Vorgaben spezifiziert, die beschreiben, wie das Pattern in diesem Produkt eingesetzt wird und für alle später angelegten Annotationen gültig sind. Hier werden z.B. Design-Vorgaben und Verhaltensmuster festgelegt, was die einheitliche Erscheinung des Merkmals sicherstellt, auch wenn das Pattern an mehreren Stellen im System angewendet wird. Die Abbildung 3.1 zeigt diesen Vorgang im zweiten Schritt.

3.1.3 Erstellung der Annotationen

Nachdem die Anwendung für alle Patterns spezifiziert wurde, werden Use Cases identifiziert, die für die Anwendung der Patterns in Frage kommen. Dann werden einzelne Use-Case-Elemente mit Annotationen versehen, die eine Referenz auf das entsprechende Usability Pattern beinhalten. Wenn nötig, werden auch zusätzliche individuelle Vorgaben spezifiziert, die das Verhalten oder Aussehen speziell für dieses Element vorschreiben. Lokale Vorgaben können auch die Erweiterung vom Use Case um zusätzliche Elemente definieren. In diesem Fall werden zusätzliche Schritte oder Abläufe erstellt.

Auf der Abbildung 3.1 wird in Schritt 3 schematisch dargestellt, wie ein Use Case annotiert wird. Es wird eine Annotation an *Schritt 3* im Normalablauf angehängt. Diese definiert einen lokalen Parameter und überschreibt lokal eine globale Vorgabe, die in der Anwendungsspezifikation von dem Usability Pattern definiert wurde. Außerdem enthält die Annotation zwei Referenzen. Für die Erfüllung der in der Annotation definierten Vorgaben wurde *Schritt 4* im *Normalablauf* und ein neuer *Alternativablauf 3a* angelegt.

Wie man der Abbildung entnehmen kann, werden neue Elemente in der Use-Case-Notation benötigt, um Annotationen mit Usability Patterns zu beschreiben. Diese werden im nächsten Kapitel erläutert.

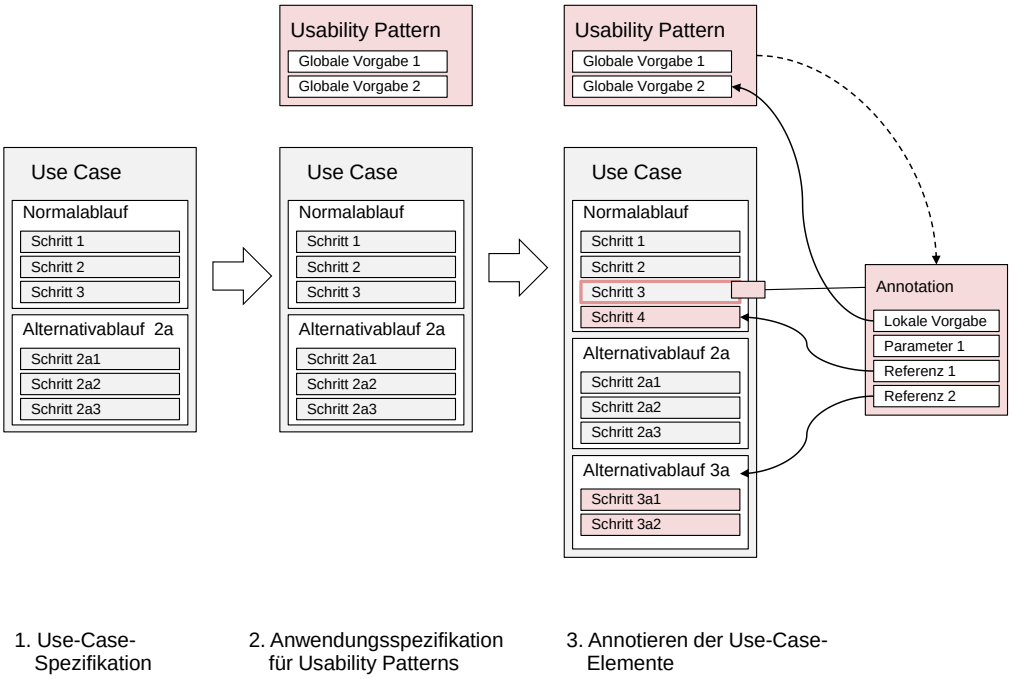


Abbildung 3.1: Erweiterung eines Use Cases mit einem Usability Pattern (Angelehnt an [Röd11b])

3.2 Erweiterung der Use-Case-Struktur

Um die Annotierung der Use Cases mit Usability Patterns zu ermöglichen, wurde im Rahmen dieser Arbeit die Use-Case-Notation um Annotationselemente erweitert. Use Cases, einzelne Abläufe oder Schritte eines Ablaufs werden mit Annotationen versehen. Annotationen sind Referenzen zu einer Anwendungsspezifikation eines Usability Patterns und kennzeichnen die Stellen, an denen die Vorgaben eines Usability Patterns erfüllt werden müssen.

Annotationen beinhalten konkrete Vorgaben, die die im Use Case beschriebenen funktionale Anforderungen präzisieren und erweitern. Eine Annotation kann folgende Elemente definieren:

- **Parameter,** die Vorgaben für die Anwendung des Usability Patterns auf das konkrete Element beinhalten.
- **Lokale Vorgaben,** die in der Anwendungsspezifikation eines Patterns definierte *Globalen Vorgaben* überschreiben und somit für das annotierte Element außer Kraft setzen.

- **Schritt- oder Ablaufreferenzen,** die Schritt- und Ablauferweiterungen definieren, die für die Erfüllung der Vorgaben von Usability Pattern eingefügt werden.

Use Cases, Abläufe und Schritte sind Use-Case-Elemente, die mit einer Annotation versehen werden können. Jede Annotation kann nur an einen Typ der Elemente angehängt werden. Welches das ist, wird mittels Spezifikationsschablonen definiert. Diese werden im nächsten Abschnitt beschrieben.

3.3 Spezifikationsschablonen in Usability Patterns

Um die Entwickler bei der Spezifikation der Usability Patterns zu unterstützen, beinhalten diese semiformalen Spezifikationsvorgaben [Röd11b]. Diese Schablonen beschreiben, wie das Pattern in die Use-Case-Spezifikation eingebunden werden kann. Es gibt drei Typen von Spezifikationsschablonen:

- **Globale Vorgaben:** Einheitliche Verhaltens- und Gestaltungsvorgaben. Sie können obligatorisch oder optional sein. Bei den optionalen Vorgaben ist es dem Entwickler überlassen, ob diese in die Anforderungsspezifikation übernommen werden. Die nach dieser Schablone erstellten Elemente sind in der Anwendungsspezifikation des Usability Patterns zu finden.
- **Globale Funktionen:** Zusätzliche Use Cases, die im Prozess der Anwendung dieses Patterns entstehen. Funktionen können ebenfalls obligatorisch oder optional sein und befinden sich auf der Ebene der Anwendungsspezifikation des Patterns.
- **Annotationsvorgaben** legen fest, welche Elemente der Use-Case-Spezifikation (Schritte, Abläufe oder ganze Use-Cases) annotiert werden können. Hier werden auch Parameter für die Annotationen definiert.

Dank Schablonen bieten Usability Patterns eine Unterstützung für die Entwickler bei der Erweiterung der Use-Case-Spezifikation. Alle Parameter und Annotationsmuster sind bereits vorgegeben, der Entwickler muss diese lediglich mit Informationen ausfüllen. Wie diese Technik in der Praxis aussieht, wird im nächsten Abschnitt gezeigt.

3.4 Ergänzung von Use Cases

In diesem Abschnitt wird der Prozess der Erweiterung der Use Cases am Beispiel des Use Cases „System starten“ aus der Abbildung 2.1 beschrieben.

Zunächst wurde aus dem Usability-Pattern-Katalog ein für den Use Case geeignetes Pattern ausgewählt. Die Wahl fiel auf das Pattern „Gute Standardwerte“, welches für den *Schritt 2a1* die Voreinstellungen festlegen soll, welche bei der Erstellung eines neuen Projektes greifen. Danach wurde die Spezifikation der Anwendung des Patterns vorgenommen. Die Abbildung 3.2 zeigt die entsprechenden Spezifikationsschablonen. Es wurde eine Beschreibung des

Abbildung 3.2: Spezifikationsschablonen im Usability Pattern „Gute Standardwerte“

Patterns in die Anforderungsspezifikation aufgenommen. Weiterhin wurden die vorgegebenen Schablonen umgesetzt. Die erste Schablone bietet eine optionale *Globale Funktion* „Standardwerte verwalten“ an, die eine zusätzliche Funktionalität für die Verwaltung der Standardwerte vorsieht. Diese wurde in einem separaten Use Case beschrieben und in der Anwendungsspezifikation des Patterns referenziert. Weiterhin enthält das Pattern eine Annotationsschablone „Standardwerte“ vom Typ *Schritt* mit einem obligatorischen Parameter *Werte*. Diese Annotationsvorschrift wurde auf den Use Case „System starten“ auf den *Schritt 2a1* angewendet. Der annotierte Use Case ist in der Tabelle 3.1 dargestellt.

Use Cases werden üblicherweise in Form von Tabellen erstellt. Dazu eignen sich gut gängige Textverarbeitungsprogramme, wie z.B. *Microsoft Word* oder *Open Office Writer*. Annotierung von Use Cases mit Usability Patterns erfolgt nach der Fertigstellung der Use-Case-Spezifikation. Die erweiterte Spezifikation wird in weiteren Phasen des Softwareetwicklungsprozesses verwendet. Sie dient als wichtige Referenz für die Entwickler bei der Erstellung von Entwurf und Code, als Vorlage bei der Erstellung von Testfällen und einem Handbuch. Entwickler, die Annotationen einpflegen oder mit der erweiterten Spezifikation arbeiten, stoßen dabei auf einige Schwierigkeiten.

Zum einen erweist sich die Pflege der Annotationen als mühsam. Sie werden als zusätzliche Zeilen in bereits bestehende Tabellen eingepflegt, was das ursprüngliche Layout zerstören kann. Außerdem werden die Annotationen mit der zentralen Usability-Pattern-Anwendungsspezifikation nicht verbunden, d.h. eventuelle Änderungen an dem Pattern (z.B. ein zusätzlicher Parameter) müssen an allen annotierten Stellen einzeln nachgezogen werden. Es ist auch nicht möglich, ein Pattern einfach zu entfernen. Alle dazugehörige Annotationselemente müssen aufgefunden und aus den Tabellen rausgenommen werden.

Zum anderen hat man keinen Überblick über die Annotationen, wenn man mit einer erweiterten Spezifikation arbeitet. Es ist z.B. nicht möglich, sich eine Liste von Use Cases ausgeben zu lassen, die mit einem bestimmten Pattern annotiert sind.

Name		System starten	
Ziel		Der Benutzer möchte das System starten.	
Akteure		Benutzer	
Primärakteur		Benutzer	
Ebene		Übersicht	
Priorität		hoch	
Normalablauf			
Vorbedingung		Das System ist installiert und lauffähig.	
1	Entwickler	Ruft die Funktion „System starten“ auf.	
2	System	lädt das zuletzt bearbeitete Projekt.	
		Fehler: Kein zuletzt bearbeitetes Projekt verfügbar.	Alternativablauf 2a
Nachbedingung		Das System ist gestartet. Das System zeigt das zuletzt bearbeitete Projekt.	
Alternativablauf 2a			
Vorbedingung		Kein zuletzt bearbeitetes Projekt verfügbar.	
2a1	System	erstellt ein neues leeres Projekt.	
		Standardwerte *Werte:* Projektname: „Neues Projekt“, Systemname: „System“; Unterordner: Usability Patterns, Akteure, Use Cases; Prioritäten: hoch, mittel, niedrig;. Ebenen: Übersicht, Benutzerebene, Technische Details.	
Nachbedingung		Das System ist gestartet. Das System zeigt ein leeres Projekt.	

Tabelle 3.1: Annotierter Use Case „System starten“

Außerdem lässt es sich nicht überprüfen, ob ein Pattern richtig angewendet wurde. Eine richtige Anwendung setzt voraus, dass:

- nur die im Pattern beschriebenen Elemente annotiert wurden (Use Case, Ablauf oder Schritt),
- alle obligatorischen Vorgaben spezifiziert wurden und
- alle zusätzlichen Elemente eingefügt wurden.

Diese Validierung lässt sich mittels von Textverarbeitungsprogrammen angebotener Funktionalität nicht automatisch durchführen und wird von Entwicklern manuell durchgeführt.

Usability Patterns bieten ein Werkzeug für die Verbesserung der Usability. Das Konzept setzt aber voraus, dass die in dem Katalog beschriebenen Schablonen während der Annotierung eingehalten werden, was sich in der Praxis als schwierig erweist, da gängige Textverarbeitungsprogramme keine speziell auf die Use Case-Modellierung zugeschnittenen Bedienkonzepte und Funktionalitäten anbieten. Bei der Erweiterung der Use Cases ist der

Entwickler selber dafür verantwortlich, auf die Konsistenz und die Einhaltung der Schablonen zu achten. Diese Tatsache hat die Überlegung angestoßen, ein Werkzeug zu entwickeln, das die Entwickler bei der Erstellung einer mit Usability Patterns annotierten Spezifikation unterstützt.

4 Werkzeugunterstützung für das Konzept

Eine der Teilaufgaben dieser Arbeit war, das Konzept der Usability Patterns zu evaluieren indem eine erweiterte Use-Case-Spezifikation erstellt wird. Dies umfasst folgende Aufgaben:

- Auswahl einer Anforderungsspezifikation, auf die das Konzept angewendet wird
- Identifizierung der Usability Patterns, die für die Anwendung auf die Spezifikation in Frage kommen
- Spezifizierung der Anwendung der ausgewählten Patterns
- Annotierung einzelner Elemente der Spezifikation

Üblicherweise wird eine Use-Case-Spezifikation als Teil der Anforderungsspezifikation mit Hilfe von Textverarbeitungsprogrammen erstellt. Für die Erweiterung der Use Cases mit Usability Patterns sind deren Funktionalitäten nicht ausreichend, wie im Abschnitt 3.4 gezeigt wurde. Aus diesem Grund erscheint für den praktischen Einsatz der Annotierung der Use Cases eine Werkzeugunterstützung notwendig. In diesem Kapitel werden die Anforderungen an das Werkzeug ausgearbeitet und existierende Use-Case-Werkzeuge im Bezug auf diese Anforderungen evaluiert. Ferner wird die Entwicklung des Werkzeugs beschrieben.

4.1 Anforderungen an das Werkzeug

Das Use-Case-Werkzeug soll die Entwickler bei der Anwendung des Konzeptes der Annotierung von Use Cases mit Usability Patterns unterstützen. Um dies zu bewerkstelligen, sind folgende Funktionen notwendig:

- Verwaltung von Use Cases
- Anzeige der Usability Patterns aus dem Katalog mit dazugehörigen Informationen und Spezifikationsschablonen
- Auswahl der Patterns im Katalog, die in der Spezifikation eingesetzt werden
- Spezifizierung der Anwendung der Patterns, die aus dem Katalog ausgewählt wurden
- Annotierung von Use-Case-Elementen entsprechend der Schablonen in den Usability Patterns
- Export der annotierten Use Cases und Import dieser in die Anforderungsspezifikation der Software

Im Folgenden werden einzelne Anforderungen in separaten Abschnitten beschrieben.

4.1.1 Verwaltung von Use Cases

Das Werkzeug soll grundlegende Funktionen zur Verwaltung von Akteuren und Use Cases anbieten. Diese werden innerhalb eines Projekts verwaltet.

Akteure

Akteure werden für ein Projekt angelegt und können den Use Cases aus diesem Projekt zugeordnet werden. Jeder Akteur verfügt über einen Namen und eine Beschreibung, die jederzeit bearbeitet werden können.

Use Cases

Use Cases werden ebenfalls für ein Projekt angelegt und können zusätzlich mit Hilfe eines Ordnersystems gruppiert werden. Für das Anlegen der Use Cases gibt es eine vorgefertigte Eingabemaske, die den Entwickler dabei unterstützt, alle für den Use Case relevanten Daten strukturiert zu erfassen. Für die Beschreibung der Use Cases wird die in Abschnitt 2.3 beschriebene Struktur verwendet.

Abläufe und Schritte

Um später eine Annotierung der Elemente mit Usability Patterns zu ermöglichen, ist ein strukturierter Aufbau der Abläufe und einzelner Schritte notwendig. Die Eingabemaske für Abläufe unterstützt das Anlegen der Vor- und Nachbedingungen und Verwaltung der einzelnen Schritte im Ablauf. Alternativabläufe werden automatisch angelegt, nachdem ein Sonderfall für einen Schritt definiert wurde. Einzelne Schritte können Referenzen auf andere Use Cases enthalten.

4.1.2 Anzeige der Usability Patterns im Katalog

Der Usability-Pattern-Katalog bietet eine Übersicht über alle vorhandenen Usability Patterns. Für jedes Usability Pattern sollen folgende Informationen angezeigt werden:

- Die komplette Beschreibung des Patterns, welches die Problembeschreibung, den Lösungsansatz, den Kontext, die Illustration und die Kosten für das Pattern beinhaltet sowie die Verbindung zu anderen Patterns aufzeichnet.
- Beispiele der Anwendung des Patterns mit Beschreibung und einem Screenshot.

- Spezifikations- und Annotationsschablonen, die Anweisungen für die Anwendung des Patterns beinhalten.

Die Anzeige des Usability-Pattern-Katalogs [Röd11a] soll über eine zusätzliche Komponente erfolgen, die eventuell in den Editor integriert werden kann.

4.1.3 Spezifikation der Anwendung von Usability Patterns

Über den Katalog werden Usability Patterns ausgewählt, die für die Annotation der Use Cases in Frage kommen. Diese müssen dann in die Projektstruktur im Use-Case-Editor übernommen werden. Für die übernommenen Patterns werden im Use-Case-Editor neue Elemente erstellt, die die Anwendung von den Patterns im Projekt entsprechend den Spezifikationsschablonen spezifizieren.

4.1.4 Annotierung von Use-Case-Elementen

Jedes Usability Pattern schreibt über die Annotationsschablonen vor, für welche Use-Case-Elemente dieses Pattern angewendet werden kann. Das Werkzeug soll die Annotierung von Use Cases, einzelner Abläufe und Schritte entsprechend dieser Vorgaben unterstützen. Eine Annotation ist ein Element, welches das entsprechende Usability Pattern referenziert und über eine Liste von lokalen Parametern verfügt. Für jedes in die Spezifikation übernommene Pattern soll eine Liste von mit diesem Pattern annotierten Use Cases angezeigt werden.

4.1.5 Export der annotierten Use Cases

Das Werkzeug soll eine um Usability Patterns und Annotationen erweiterte Use-Case-Spezifikation im *PDF*-, *RTF*-, *XML* und *HTML*-Format generieren. Es soll ebenfalls möglich sein, nur eine Auswahl von Use Cases zu exportieren. Das Format des Exportes ist wichtig, weil die erweiterte Use-Case-Spezifikation eventuell in die Anforderungsspezifikation der Software übernommen wird, welche mit einem Textverarbeitungsprogramm erstellt wird.

4.2 Evaluierung existierender Werkzeuge

Üblicherweise werden Use Cases mit Hilfe von Textverarbeitungsprogrammen erstellt. Aus diesem Grund bestand kein großer Bedarf nach einer speziellen Software. Dennoch gibt es einige wenige Werkzeuge, die sich auf die strukturierte textbasierte Beschreibung von Use Cases spezialisiert haben. In diesem Abschnitt werden diese vorgestellt. Es wird auch untersucht, ob diese Werkzeuge für die Erweiterung der Use-Case-Spezifikation mit Usability Patterns geeignet sind. Dabei sind folgenden Fragen von einem besonderen Interesse:

1. Ist die strukturierte, textbasierte Beschreibung von Use Cases möglich?

2. Werden Vorlagen für die Use Cases angeboten, die den Entwickler bei der Erstellung unterstützen?
3. Wird die Möglichkeit für die Abbildung der Beziehungen zwischen den Use Cases angeboten, oder ist deren Repräsentation dem Entwickler überlassen?
4. Wird Reportgenerierung angeboten? Ist das Format der Reports für die Weiterbearbeitung geeignet?
5. Gibt es eine Möglichkeit, die Anwendung der Usability Patterns mit Hilfe des Werkzeugs zu beschreiben?
6. Gibt es die Möglichkeit für die Darstellung der Annotationen für Use Cases, Abläufe und Schritte?

Im Folgenden werden einzelne Werkzeuge hinsichtlich der gestellten Fragen betrachtet. Alle aufgeführten Werkzeuge bieten Möglichkeiten zur strukturierten Bearbeitung von Use-Case-Beschreibungen, unterscheiden sich aber in Details wesentlich. Jedes Werkzeug wird in einem eigenen Abschnitt beschrieben. Es wird nicht auf den gesamten Funktionsumfang eingegangen, sondern nur auf die bewertungsrelevante Merkmale. Zum Zweck eines anschaulichen Vergleichs wird ein einfacher Use Case als Beispiel genommen und mit Hilfe der vorgestellten Werkzeuge modelliert. Der Use Case „Kunde anlegen“ hat einen Normalablauf und eine Verzweigung zum Alternativablauf im Schritt 4. Zu jedem Werkzeug gibt es eine Abbildung, welche die Darstellung dieses Use Cases zeigt.

4.2.1 Case Complete 2011

CaseComplete [Cas11] ist ein kommerzielles Werkzeug zur Erstellung der Use Cases. Es bietet eine interaktive Anleitung zur Vorgehensweise bei der Use-Case-Modellierung und mehrere Dokumentenvorlagen für die im Modellierungsprozess entstehende Artefakte. Es gibt einen Bereich zur Definition von Akteuren und Use Cases. Darüber hinaus wird die Möglichkeit angeboten, ein Glossar und eine Liste von Anforderungen und Verknüpfungen zu beliebigen Dokumenten zu pflegen und mit Use Cases zu verknüpfen. Es ist möglich, über Referenzen Inklusionsbeziehungen zwischen Use Cases explizit festzuhalten. Detaillierte Beschreibung von Use Cases können in einem separaten Dialog mit Hilfe einer Eingabemaske vorgenommen werden. Die Eingabemaske dafür wird in Abbildung 4.1 dargestellt.

Wie man der Abbildung entnehmen kann, ist die Struktur der Beschreibung der Use Cases in *CaseComplete* der in dieser Arbeit definierten ähnlich. Die wichtigsten Informationen werden im Hauptreiter erfasst. Zwei zusätzliche Reiter bieten Möglichkeiten für mehrere detaillierte Eingaben und Verknüpfungen. Ebenfalls im Hauptreiter werden der Normalablauf und Alternativabläufe angezeigt. Schritte werden in Form einer durchgehend nummerierten Liste dargestellt, in der Beschreibung der Schritte vorkommende Namen der Akteure werden hervorgehoben und mit einem Hyperlink versehen, der auf die Akteurbeschreibung verweist.

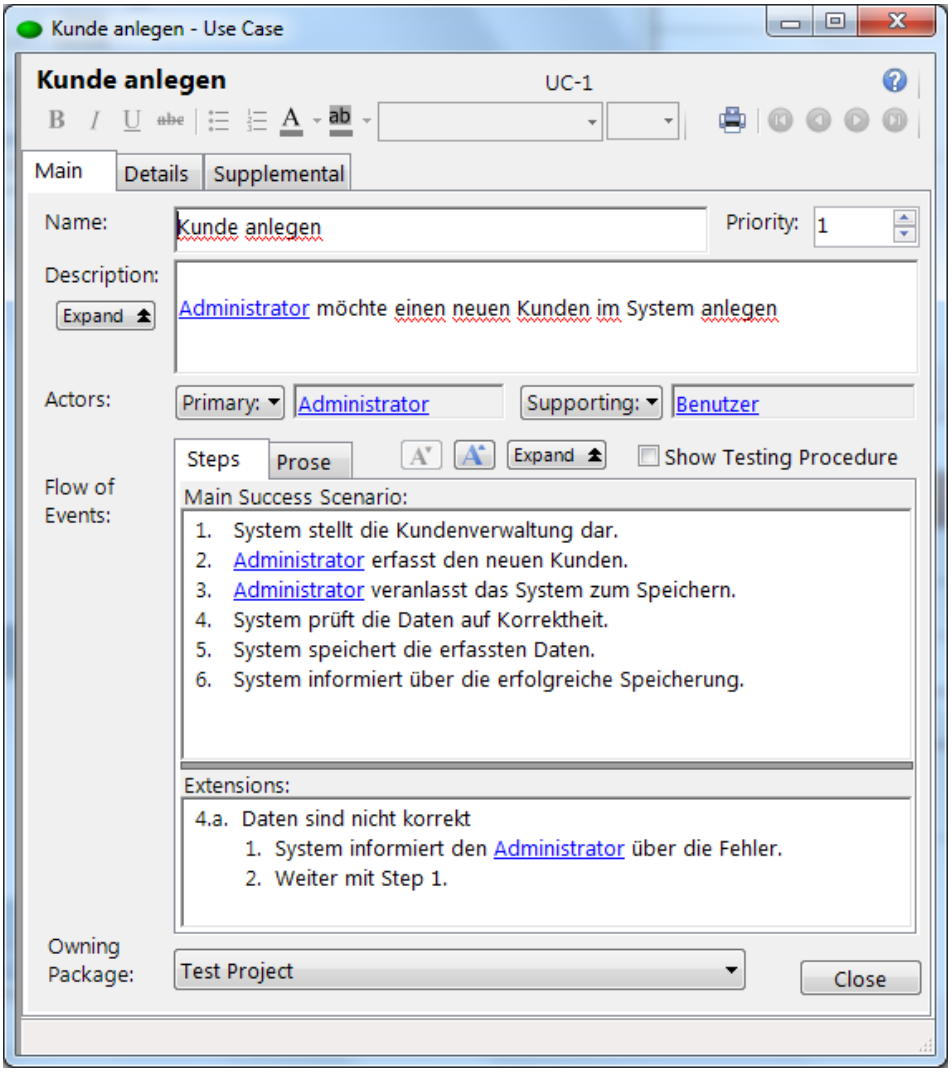


Abbildung 4.1: Eingabemaske für Use Cases in CaseComplete

Für die Darstellung der Usability Patterns würden sich zahlreiche zusätzliche Felder in der Use-Case-Beschreibung anbieten. Es gibt allerdings keine Möglichkeit, Patterns zentral zu definieren und diese in einzelnen Use Cases zu verlinken.

CaseComplete bietet neben der Bearbeitung auch die Möglichkeit, erstellte Use Cases zu exportieren. Für das gesamte Projekt oder für die einzelnen Elemente (Use Cases, Akteure, Anforderungen) werden Reports in *HTML*-, *Word*- und *Excel*-Format mit Hilfe zahlreicher Vorlagen generiert.

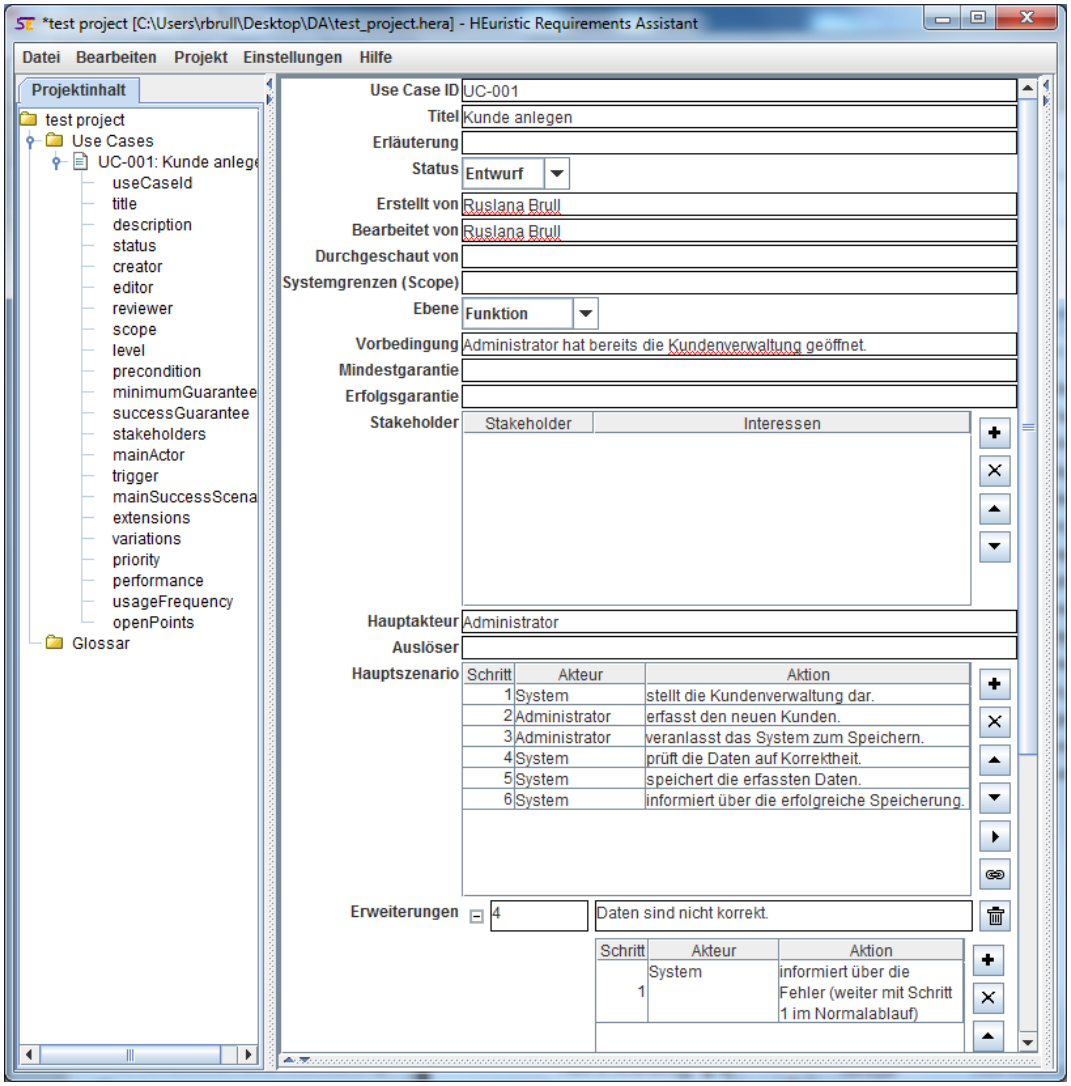


Abbildung 4.2: Eingabemaske für Use Cases in HeRA

4.2.2 HeRA

HeRA (**He**uristic **R**equirements **A**ssistant) ist ein Projekt des Fachgebietes Software Engineering der Leibniz Universität Hannover. Es ist im Laufe einer Masterarbeit entstanden und wird bis heute im Rahmen mehrerer Projekte weiterentwickelt [HeR11]. *HeRA* dient zur Erstellung und Bearbeitung von Use Cases und Anforderungen und verfügt über zusätzliche Plugins, die seine Funktionalität erweitern. Das Glossarplugin erlaubt die Verwaltung von Begriffen und zugehörigen Definitionen und verfügt über semantische Verifizierung [Heco9]. Die besondere Funktionalität von *HeRA* ist das heuristische Feedback. Die Eingaben der Benutzer werden ständig analysiert und auf die Vollständigkeit geprüft. Der Benutzer

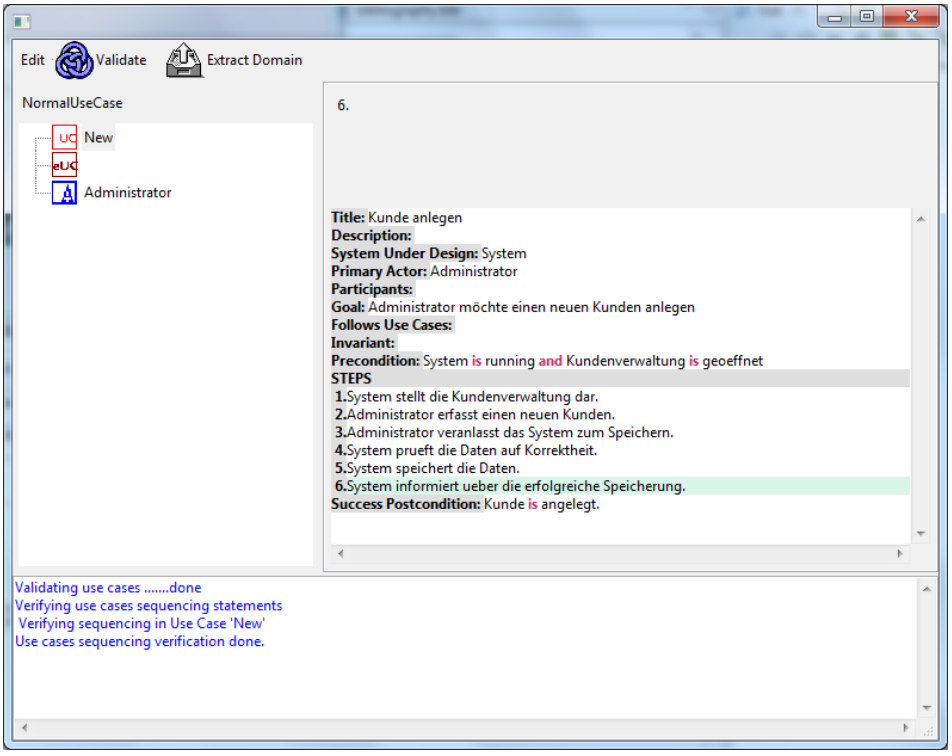


Abbildung 4.3: Eingabemaske für Use Cases in UCEd

bekommt die Rückmeldung über den aktuellen Stand seiner Arbeit in Form von Warnungen und Empfehlungen.

Abbildung 4.2 zeigt die Eingabemaske für die Bearbeitung der Use Cases in *HeRA*. Das Werkzeug sieht mehrere Felder für die Speicherung der Informationen über Use Cases vor. Abläufe werden mit Hilfe von Tabellen dargestellt. Einzelne Schritte können um Alternativabläufe erweitert werden. Es ist möglich, aus einem Schritt weitere Use Cases zu referenzieren.

In *HeRA* erstellte Elemente können in *HTML*- und *TEX*- Format exportiert werden. Use Cases werden dabei in Tabellenform dargestellt.

4.2.3 UCEd

UCEd (**U**se **C**ase **Ed**itor) ist ein frei verfügbares Werkzeug, das die automatisierte Unterstützung der Anforderungsanalyse anstrebt [UCE11]. Neben der Erstellung und Bearbeitung von Use-Case-Beschreibungen ermöglicht *UCEd* die Erzeugung der Beschreibung eines endlichen Automaten und simuliert die im Use Case beschriebenen Abläufe mittels dieses Automaten.

Wie in der Abbildung 4.3 zu sehen ist, unterstützt *UCEd* die strukturierte Beschreibung von Use Cases. Allerdings verlangt das Werkzeug für die Weiterverarbeitung der Informationen die Einhaltung einer vorgegebenen Grammatik, die in dem auf der Webseite verfügbaren Handbuch [Som07] erläutert wird. Der syntaktische Freiraum bei der Benutzung dieser semiformaler Sprache ist eng begrenzt, was einen Einarbeitungsaufwand von den Use-Case-Entwicklern erfordert. Dies wirkt sich negativ auf die Benutzbarkeit des Werkzeugs aus.

UCEd bietet auch die Möglichkeit, erstellte Modelle im *HTML*-Format zu exportieren. Die exportierte Darstellung enthält die Beschreibung von Use Cases sowie die Darstellung des generierten Automaten.

4.2.4 Remas

Remas (**Re**quirements **Ma**nagement **S**ystem) ist eine auf Eclipse basierte Open-Source-Software zur Anforderungsverwaltung. Es bietet die Möglichkeit zur Verwaltung von Use Cases, Akteuren, sowie funktionalen und nichtfunktionalen Anforderungen [rem11].

Die Abbildung 4.4 zeigt die graphische Oberfläche von *remas*. Ganz unten ist der Projektbaum zu sehen, in dem die zu einem Projekt gehörende Akteure, Systeme, Use Cases, Anforderungen, Metriken, Glossareinträge und Referenzen verwaltet werden. Oben links ist die Eingabemaske für Use Cases dargestellt. Für die Bearbeitung der Abläufe ist ein zusätzliches Fenster vorgesehen (siehe Abbildung 4.4, unten links). Für jeden Use Case kann eine Liste von Schritten angelegt werden. Es wird keine Möglichkeit angeboten, Alternativabläufe anzulegen, diese können aber als Abzweigungen im Normalablauf mittels zusätzlicher (*Substep*) und alternativer (*Altstep*) Schritte dargestellt werden. *Remas* bietet auch die Möglichkeit, alle Elemente mittels so genannter *Links* miteinander zu verknüpfen.

Für die Abbildung von Usability Patterns in *remas* sind die Metrikenelemente am besten geeignet. Oben rechts in Abbildung 4.4 sind die Eigenschaften einer *Metrik* zu sehen. Diese umfassen lediglich einen Namen und eine Beschreibung. Für die Spezifikation der Anwendung der Usability Patterns wären noch zusätzliche Felder notwendig, die Parameter und Vorgaben darstellen. Eine *Metrik* wird immer für ein bestimmtes System definiert und kann mittels Links mit beliebigen Elementen verbunden werden. Unten rechts in der Abbildung sieht man einen *Link* von einer *Metrik* zu einem Schritt. Diese *Links* können Annotationen einzelner Elemente mit Usability Patterns darstellen. Allerdings lassen diese keine zusätzlichen Angaben zu, was für die Darstellung lokaler Vorgaben allerdings unabdingbar ist.

Das in *remas* erstellte Projekt und einzelne Elemente können in *HTML*-Format exportiert werden. Dabei kann die Darstellung mittels angebotener Schablonen vom Entwickler selbst gestaltet werden.

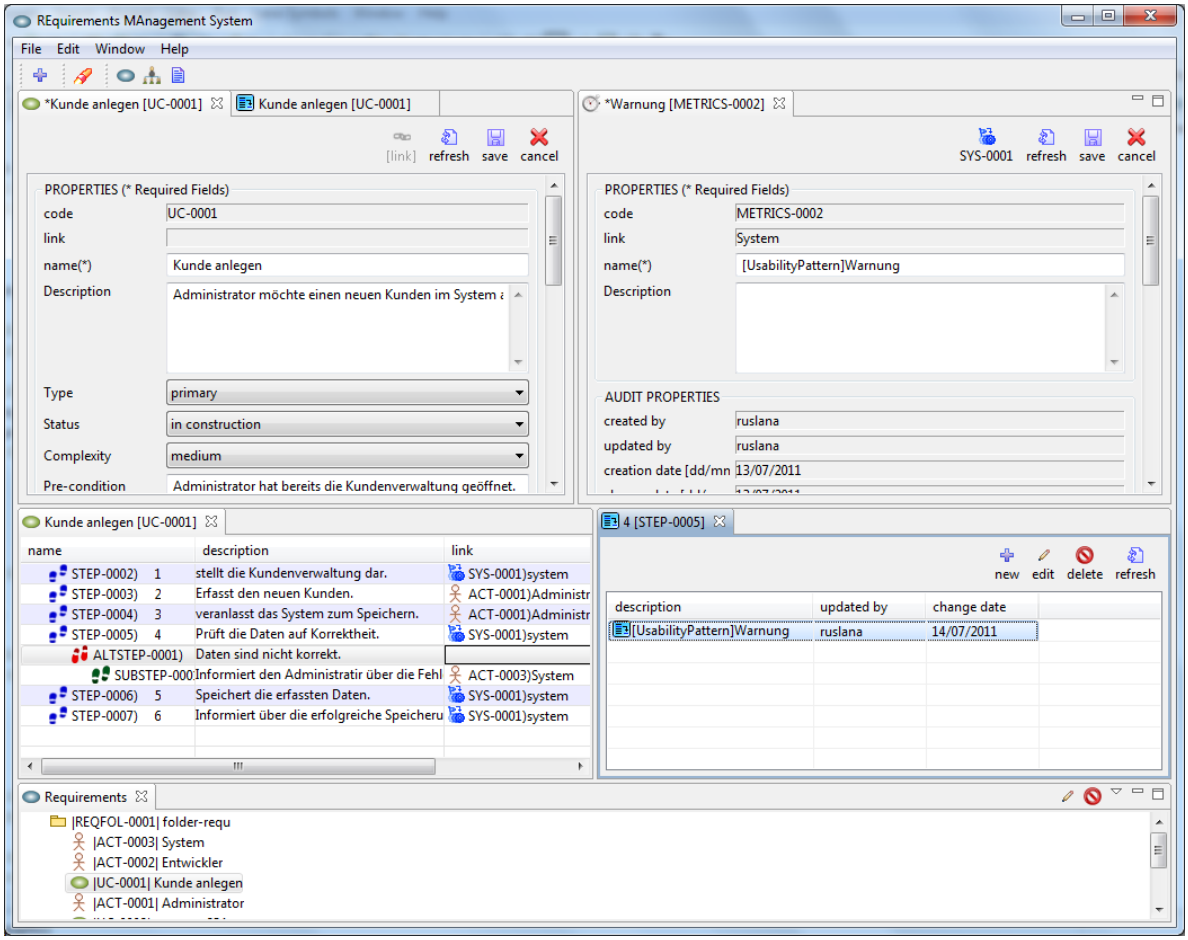


Abbildung 4.4: Bearbeitung von Use Cases in remas

4.2.5 Zusammenfassung

In Tabelle 4.1 wird der Vergleich der untersuchten Werkzeuge bezüglich der oben definierten Merkmale dargestellt. Über die Grundfunktionalität zur Verwaltung von Use Cases verfügen alle vier Werkzeuge. Es ist möglich, Use Cases mit Hilfe einer unterschiedlich detaillierter Vorlage zu erstellen und diese in einem Projektbaum zu verwalten. Alle Werkzeuge außer *UCEd* bieten darüber hinaus die Abbildung der Beziehungen zwischen den Use Cases an. Die Export-Funktion wird von allen Produkten angeboten, in einem weiterverwendbaren Format allerdings nur von *HeRA* und *CaseComplete*. *CaseComplete* und *remas* weisen ein breiteres Spektrum an Funktionalitäten für die Verwaltung von Use Cases und Anforderungen auf, während die anderen zwei Werkzeuge höher spezialisiert sind.

Keines der Werkzeuge ist im aktuellen Zustand zur Erstellung der mittels Usability Patterns erweiterten Use Cases geeignet. Wegen der vorhandenen Grundfunktionalität und der

Werkzeug	Strukturierte Beschreibung	Vorlagen	Beziehungen	Export	Patterns	Annotationen
Case Complete	ja	ja	ja	HTML Word Excel	nein	teilweise
HERA	ja	ja	ja	HTML TEX	nein	nein
UCEd	ja	ja	nein	HTML	nein	nein
remas	ja	ja	ja	HTML	ja	teilweise

Tabelle 4.1: Vergleich der Use Case Editoren

Verfügbarkeit des Quellcodes bieten sich aber *HeRA* und *remas* an, um deren Erweiterungsmöglichkeit zu untersuchen.

4.3 Evaluierung einer Erweiterungsmöglichkeit

Die Evaluierung der Werkzeuge hat ergeben, dass *HeRA* und *remas* dafür geeignet sind, nach entsprechender Erweiterung für die Evaluierung des Konzeptes der Annotierung von Use Cases mit Usability Patterns eingesetzt zu werden. Die Erweiterung umfasst folgende Punkte:

- Import der Usability Patterns aus dem Katalog
- Verwaltung von Usability Patterns im Projekt
- Erweiterung der Struktur der Use-Case-Elemente um Annotationen zu ermöglichen
- Annotation der Use-Case-Elemente

Eine technische Systemanalyse hat ergeben, dass beide Werkzeuge eine komplexe, modular aufgebaute Architektur aufweisen. Die Erweiterung um eine Usability-Patterns-Komponente kann ohne großen Einfluss auf andere Module erfolgen. Die Annotierungsfunktionalität verlangt aber die Erweiterung der Use-Case-Struktur um Annotationselemente. Da Use-Case-Verwaltung die Kernfunktionalität dieser Werkzeuge darstellt, könnte eine solche Änderung der Struktur Auswirkungen auf andere Komponenten haben. Die Analyse der Abhängigkeiten wurde dadurch erschwert, dass die Dokumentation für beide Werkzeuge nur spärlich vorhanden ist.

Eine grobe Schätzung hat ergeben, dass unter diesen Umständen der Einarbeitungs- und Umbauaufwand für die Erweiterung eines der beiden Werkzeuge für die Durchführung im Rahmen dieser Arbeit zu umfangreich wäre. Eine Neuentwicklung würde dagegen folgende Vorteile mit sich bringen:

- Das Werkzeug bleibt übersichtlich. Es werden nur die Funktionen umgesetzt, die auch für die Evaluation nötig sind.
- Die Struktur und Architektur des Werkzeugs sind nicht vorgegeben. Somit können beliebige Usability Patterns bei der Entwicklung berücksichtigt werden.
- Da der gesamte Softwareentwicklungsprozess durchlaufen wird, kann die Anwendung der Usability Patterns in allen Phasen beobachtet und untersucht werden.

Nach der Gewichtung aller Vor- und Nachteile wurde die Entscheidung getroffen, ein neues Werkzeug zu entwickeln, das die Anwendung der Usability Patterns demonstriert. Bei der Entwicklung sollen ausgewählte Patterns angewendet werden.

5 Realisierung von Tulip

Dieser Abschnitt beschreibt die Realisierung des Werkzeugs namens *Tulip* (**T**ool for **U**se Case Specification with Usabi**li**ty **P**atterns). Die Realisierung des Werkzeugs selbst stellt eine praktische Anwendung des Konzeptes der Usability Patterns im Entwicklungsprozess dar. In der Anforderungsanalysephase werden einige Patterns aus dem Katalog ausgewählt. In der Spezifikationsphase werden diese in die Spezifikation aufgenommen und Use-Case-Elemente annotiert. Beim Entwurf und Implementierung werden die Vorgaben aus den Patterns und Annotationen berücksichtigt. Die Vorgaben aus den Patterns stellen auch Qualitätskriterien für die Softwaretests dar. Mit Hilfe des fertigen Werkzeugs wird anschließend eine erweiterte Use-Case-Spezifikation erstellt.

5.1 Erstellung der Spezifikation

In diesem Abschnitt wird die Erstellung der mit Usability Patterns erweiterten Use-Case-Spezifikation [RB11] für das Werkzeug *Tulip* beschrieben.

Die Anforderungen für *Tulip* wurden vollständig geklärt, analysiert und in einer Anforderungsspezifikation festgehalten. Die funktionalen Anforderungen wurden mit Hilfe von Use Cases beschrieben. Dabei wurde die Notation aus dem Abschnitt 2.3 verwendet. Bei der Anforderungsanalyse wurden folgende Funktionalitäten identifiziert:

- System starten
- Projekt verwalten
- Akteure verwalten
- Use Cases verwalten
- Abläufe verwalten
- Usability Patterns verwalten
- Use-Case-Element annotieren
- Use Cases exportieren
- Use Cases importieren
- Einstellungen verwalten
- Infodialog aufrufen

Projekt öffnen (UC-202)		
Ziel:		Der Entwickler will ein bestehendes Projekt öffnen
Akteure:		**Entwickler**
Beschreibung:		
Ebene:		Benutzerebene
Priorität:		MUST
Normalablauf		
Vorbedingung:		Das System ist gestartet
1	Entwickler	Ruft die Funktion „Projekt öffnen“ auf.
2	System	Bietet die Möglichkeit, die Projektdatei im Dateisystem auszuwählen.
3	Entwickler	Wählt eine Datei aus.
4	Entwickler	bestätigt die Auswahl.
5	System	Schließt das aktuell geöffnete Projekt.
6	System	öffnet das vom Entwickler ausgewählte Projekt.
Nachbedingung:		Das System hat das vom Entwickler ausgewählte Projekt geöffnet.

Abbildung 5.1: Use Case „Projekt öffnen“

Für jede Funktionalität wurde ein Use Case oder eine Sequenz von Use Cases erstellt. Insgesamt wurden 26 Use Cases spezifiziert. Die Abbildung 5.1 zeigt am Beispiel des Use Cases „Projekt öffnen“, wie die ursprünglichen Use Cases in der Spezifikation aussehen.

5.2 Erweiterung der Spezifikation um Usability Patterns

5.2.1 Auswahl der Patterns

Nach der Fertigstellung der Use-Case-Spezifikation wurden Usability Patterns aus dem Katalog ausgewählt, welche für den Einsatz in *Tulip* als sinnvoll erschienen.

Zunächst wurden aus 20 Patterns, die im Katalog enthalten sind, diejenigen eliminiert, die zu dem Zeitpunkt der Entwicklung noch nicht vollständig ausgearbeitet wurden. Diese sind *Systemstatus*, *Assistent* und *Expertenmodus*. Des Weiteren wurden die Patterns ausgeschlossen, die für den Einsatz in *Tulip* mit spezifizierter Funktionalität aus verschiedenen Gründen nicht in Frage kommen. Diese sind:

- **Wiederholung** erlaubt es dem Benutzer, die einmal ausgeführte Aktion auf eine einfache Weise zu wiederholen, ohne dass alle Eingaben noch einmal gemacht werden müssen. In *Tulip* kommen keine Funktionalitäten mit mehreren komplizierten Eingaben vor, so dass dieses Pattern nicht eingesetzt werden kann.
- **Fortschrittsanzeige** sieht eine Anzeige der Dauer vor, wenn das System eine Aktion ausführt, die längere Zeit dauert. Da in *Tulip* keine komplizierten Berechnungen durchgeführt werden und kein Datenaustausch mit einer Datenbank oder anderen Systemen notwendig ist, wird davon ausgegangen, dass es keine Aktionen gibt, deren

Ausführung länger dauert, als eine Sekunde. Aus diesem Grund wird dieses Pattern nicht eingesetzt.

- **Verarbeitungsanzeige** sieht eine Anzeige für den Benutzer vor, wenn eine Aktion im Hintergrund ausgeführt wird. Da es in *Tulip* keine Aktionen gibt, die im Hintergrund ausgeführt werden, erscheint der Einsatz dieses Patterns als nicht sinnvoll.
- **Auto-Vervollständigung** sieht das Vorschlagen geeigneter Werte während der Eingabe vor. Diese Funktionalität ist sinnvoll, wenn es Eingabewerte gibt, die aus einer größeren Menge der Eingabewerte stammen. In *Tulip* kommt es nur bei der Auswahl der *Priorität*, der *Ebene* eines Use Cases, sowie bei der Auswahl eines *Akteurs* für einen Schritt, vor. Dabei wird die Menge der möglichen Eingabewerte von fünf Elementen in der Regel nicht überschritten. In diesem Fall erscheint eine Auswahl über eine Liste sinnvoller, als die Auswahl über die freie Eingabe mit einer Vorschlagmöglichkeit. Aufgrund dieser Überlegung wurde dieses Pattern verworfen.
- **Nachsichtiges Format** sieht vor, dass bei Bedarf die Eingaben vom Benutzer in das richtige Format umgewandelt werden. Da in *Tulip* kein Format für die Eingaben der Benutzer festgelegt wird, wird dieses Pattern nicht eingesetzt.
- **Vorschau** bietet eine Vorschau auf die voraussichtlichen Resultate der Aktion, ohne die Aktion vollständig auszuführen oder Änderungen durchzuführen. Dieses Pattern erscheint für den Einsatz in *Tulip* nicht sinnvoll, da keine Aktionen vorgesehen sind, die nur mit einem großen Aufwand rückgängig gemacht werden können. Außerdem wird für die meisten Aktionen eine Undo-Funktion angeboten.
- **Ausführung im Hintergrund** bietet Benutzern an, lang andauernde Aktionen im Hintergrund auszuführen. In *Tulip* werden keine Aktionen gleichzeitig ausgeführt, daher ist die Ausführung im Hintergrund nicht nötig.

Nach diesem Ausschlussverfahren wurden zehn Patterns, die für den Einsatz in *Tulip* als geeignet erschienen, näher untersucht. Es wurden zwei Gruppen von Patterns identifiziert, die eine ähnliche Funktionalität spezifizieren, so dass es ausreichend ist, nur ein Pattern aus der Gruppe für den Einsatz in *Tulip* auszuwählen.

Die erste Gruppe beinhaltet Patterns, die die Undo-Funktionalität unterstützen. Diese sind *Globales Undo* und *Objektbezogenes Undo*. Beim *Globalen Undo* hat der Benutzer die Möglichkeit, eine aus Versehen ausgeführte Aktion rückgängig zu machen. Das *Objektbezogene Undo* bietet die gleiche Funktionalität, nur jeweils auf ein bestimmtes Objekt bezogen. Das heißt, dass mehrere Historien gleichzeitig verwaltet werden, jeweils eine für das zurzeit bearbeitete Objekt. Diese Option erscheint für *Tulip* vorteilhaft, da die Möglichkeit besteht, mehrere Objekte gleichzeitig zu bearbeiten. Aus diesem Grund wurde die Entscheidung getroffen, das Usability Pattern *Objektbezogenes Undo* in *Tulip* anzuwenden.

Zur zweiten Gruppe gehören folgende Patterns: *Sicherheitskopie*, *Automatisches Speichern* und *Dokumentwiederherstellung*. Diese drei Patterns behandeln die Erstellung zusätzlicher Kopien des aktuellen Projekts. Bei *Sicherheitskopie* wird bei jeder vom Benutzer ausgelösten Speicherung der Datei ein Backup für die alte Datei gemacht, so dass man den Stand der vorletzten Speicherung wiederherstellen kann. Beim *Automatischen Speichern* wird das Projekt

in regelmäßigen Abständen gespeichert ohne dass der Benutzer das System dazu auffordern muss. Dabei wird die Projektdatei überschrieben. Bei der *Dokumentwiederherstellung* geht es darum, dass das Projekt nach jeder vom Benutzer vorgenommenen Änderung in einer separaten Datei gespeichert wird. Im Fall eines Systemfehlers kann die letzte Fassung eines Dokuments wiederhergestellt werden. Für den Einsatz im *Tulip* wurde die *Dokumentwiederherstellung* ausgewählt, da das automatische Speichern und das damit verbundene ständige Überschreiben der Datei nicht erwünscht ist. Des Weiteren geschieht das Anlegen einer Sicherheitskopie nur beim Speichern, was nicht von den Systemfehlern schützt.

Nachdem der Aufwand für die Aufnahme der sieben ausgewählten Usability Patterns in die Anforderungen an *Tulip* geschätzt wurde, wurden diese entsprechend den Wünschen des Kunden priorisiert. Dabei wurde vereinbart, dass das Pattern *Filter*, welches es dem Benutzer erlaubt, dargestellte Daten nach eigenen Kriterien zu filtern, für diese Implementierung eine niedrige Priorität hat und für die erste Version von *Tulip* nicht in die Anforderungen aufgenommen wird.

Somit ist eine Liste von sechs Patterns entstanden, welche in die Anforderungen an *Tulip* aufgenommen werden. Diese sind: *Gute Standardwerte*, *Objektbezogenes Undo*, *Abbruch*, *Warnung*, *Direkte Validierung* und *Dokumentwiederherstellung*.

5.2.2 Spezifizierung der Anwendung der Patterns

Für die Spezifikation der Anwendung der ausgewählten Usability Patterns in *Tulip* wurde ein separates Kapitel in der Spezifikation angelegt. Für jedes Pattern wurde eine Beschreibung sowie *Globale Vorgaben*, *Globale Funktionen* und *Annotationsvorschriften* entsprechend den im Usability Pattern Katalog definierten Schablonen hinterlegt. In der Tabelle 5.1 sind alle Anwendungsspezifikationen aufgeführt. Die Annotationen für jedes Pattern sind hervorgehoben.

Gute Standardwerte		
Beschreibung		Beim Anlegen neuer Elemente füllt das System einige Felder mit Standardwerten. Der Benutzer kann die Standardwerte jederzeit überschreiben.
Annotation	@Schritt	Standardwerte
Objektbezogenes Undo		
Beschreibung		Das System soll es Benutzern erlauben, Aktionen objektbezogen rückgängig zu machen.
Annotation	@Ablauf	Objekt-Undo
Vorgaben	Objekte	Projektbaum (Operationen auf Elementen im Baum, z.B. Verschieben, Löschen etc.), Projekt (d.h. Projektstammdaten), Akteur, Use Case, Usability Pattern

	Undo-Verhalten	Lineares Undo: 1-10 zuletzt ausgeführte Aktionen pro Objekt sollen in der umgekehrten Reihenfolge der ursprünglichen Ausführung rückgängig gemacht werden können. Die Undo-Historie wird geleert, wenn das jeweilige Objekt nicht mehr angezeigt wird (gilt für Akteur, Use Case, Usability Pattern, Projektstammdaten).
	Redo-Verhalten	Aktionen, die per Undo rückgängig gemacht wurden, sollen auch wiederherstellbar sein.
Abbruch		
Beschreibung		Der Benutzer soll in der Lage sein, einige Aktionen abzubrechen. Das System soll den Zustand vor der Ausführung der Aktion wiederherstellen.
Annotation	@Ablauf	Abbruch
Warnung		
Beschreibung		Das System warnt den Benutzer vor der Ausführung der Funktionen, die nicht rückgängig gemacht werden können.
Annotation	@Schritt	Warnung
Vorgaben	Darstellung	Das System zeigt Warnungen als Popup Dialog an.
Direkte Validierung		
Beschreibung		Eingaben des Benutzers sollen direkt auf Gültigkeit geprüft werden, ohne dass der Benutzer eine spezielle Funktion dafür aufrufen muss. Die Validierungsfehler und Hinweise werden sofort dem Benutzer sofort signalisiert.
Annotation	@Schritt	Direkte Validierung
Vorgaben	Darstellung	Ungültige Eingabewerte werden durch ein Fehlersymbol neben dem Eingabefeld gekennzeichnet. Ein Hinweis auf die Fehlerursache wird als ToolTip des Symbols angezeigt.
	Validierungs-zeitpunkt	Die Validierung findet unmittelbar während der Eingabe statt.
Dokumentwiederherstellung		
Beschreibung		Nach jeder Änderung der Projektdaten speichert das System Wiederherstellungsinformationen, mit denen die Projektdaten wiederhergestellt werden können.
Annotation	@Ablauf	Dok.wdhst.

Vorgaben	Strategie	Nach jeder Änderung der Projektdaten speichert das System Wiederherstellungsinformationen, mit denen die Projektdaten wiederhergestellt werden. Die Wiederherstellungsinformationen werden in einer Datei im Benutzerverzeichnis gespeichert. Bei einem normalen Systemende wird die Datei mit den Wiederherstellungsinformationen gelöscht. Nach einem unerwarteten Systemabbruch („Absturz“) bietet das System dem Benutzer beim nächsten Systemstart an, die Projektdaten anhand der Wiederherstellungsinformationen wiederherzustellen.
	Daten	Vollständiger Projektbaum, Zeitpunkt der Speicherung
Funktionen	Dokumentwiederherstellung nach Systemfehler	UC-101: System starten, Alternativer Ablauf 2b

Tabelle 5.1: Spezifikation der Anwendung der Usability Patterns in Tulip

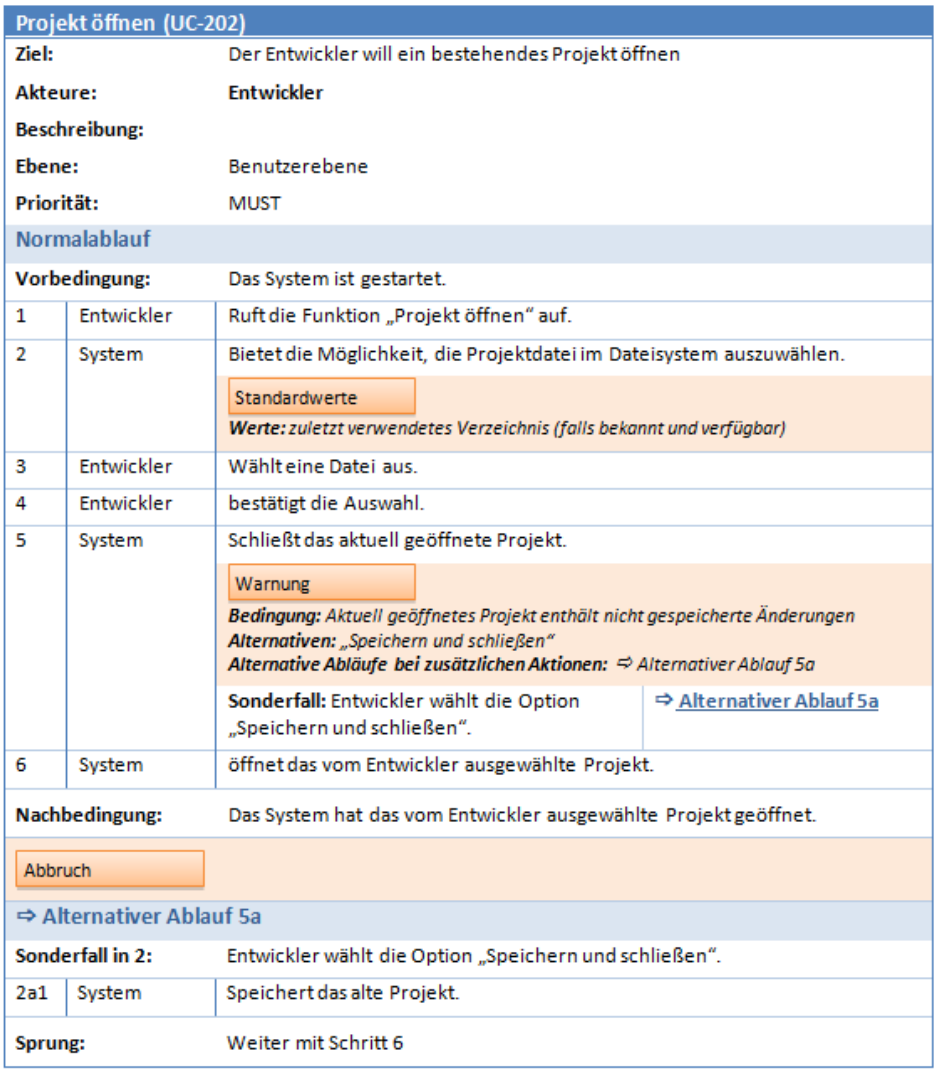
5.2.3 Annotierung der Use-Case-Elemente

Im nächsten Schritt wurden für jedes Pattern die Use Cases ausgewählt, in denen das Pattern zum Einsatz kommt. Diese Use Cases oder deren Elemente wurden mit einer Annotation versehen. Ferner wurden von dem Pattern vorgeschriebene lokale Vorgaben und Parameter beschrieben und zusätzliche Elemente angelegt und referenziert.

Beispiel

Die Abbildung 5.2 zeigt noch einmal den Use Case „Projekt öffnen“ (vgl. Abbildung 5.1). Der Use Case wurde mit drei Usability Patterns annotiert:

1. Schritt 2 aus dem Normalablauf wurde mit dem Pattern *Standardwerte* annotiert. An dieser Stelle bietet das System dem Entwickler die Möglichkeit, die Datei auszuwählen, die geöffnet werden soll. Das Pattern legt fest, dass standardmäßig ein bestimmtes Verzeichnis angezeigt wird. Der vorgeschriebene Parameter „Werte“ definiert, welches Verzeichnis als Standardverzeichnis verwendet wird.
2. Im Schritt 5 des Normalablaufs schließt das System das aktuelle Projekt. Usability Pattern *Warnung* sieht eine Warnung vor, falls das aktuelle Projekt nicht gespeicherte Änderungen enthält. Durch die Parameter wird die Bedingung für die Warnung und die Möglichkeiten für das weitere Vorgehen festgelegt. Außerdem wurde für die Annotation ein alternativer Ablauf angelegt, der den Fall behandelt, dass der Entwickler sich für die Speicherung des aktuellen Projektes entscheidet.



Projekt öffnen (UC-202)

Ziel:		Der Entwickler will ein bestehendes Projekt öffnen
Akteure:		**Entwickler**
Beschreibung:		
Ebene:		Benutzerebene
Priorität:		MUST
Normalablauf		
Vorbedingung:		Das System ist gestartet.
1	Entwickler	Ruft die Funktion „Projekt öffnen“ auf.
2	System	Bietet die Möglichkeit, die Projektdatei im Dateisystem auszuwählen. Standardwerte ***Werte:*** *zuletzt verwendetes Verzeichnis (falls bekannt und verfügbar)*
3	Entwickler	Wählt eine Datei aus.
4	Entwickler	bestätigt die Auswahl.
5	System	Schließt das aktuell geöffnete Projekt. Warnung ***Bedingung:*** *Aktuell geöffnetes Projekt enthält nicht gespeicherte Änderungen* ***Alternativen:*** *„Speichern und schließen“* ***Alternative Abläufe bei zusätzlichen Aktionen:*** ⇨ *Alternativer Ablauf 5a*
		Sonderfall: Entwickler wählt die Option „Speichern und schließen“. ⇨ Alternativer Ablauf 5a
6	System	öffnet das vom Entwickler ausgewählte Projekt.
Nachbedingung:		Das System hat das vom Entwickler ausgewählte Projekt geöffnet.
Abbruch		
⇨ Alternativer Ablauf 5a		
Sonderfall in 2:		Entwickler wählt die Option „Speichern und schließen“.
2a1	System	Speichert das alte Projekt.
Sprung:		Weiter mit Schritt 6

Abbildung 5.2: Annotierter Use Case „Projekt öffnen“

3. Der Normalablauf wurde mit dem Pattern *Abbruch* annotiert, was die Abbruchmöglichkeit für jeden Schritt des Ablaufs vorsieht. Nach dem Abbruch wird der Stand von vor dem ersten Schritt wiederhergestellt.

5.3 Entwurf und Implementierung

Dieser Abschnitt beschreibt die einzelnen Komponenten von *Tulip* und deren Implementierung. Es wird auch auf die Auswirkungen der Berücksichtigung der Usability Patterns auf den Entwurf- und Implementierungsprozess eingegangen.

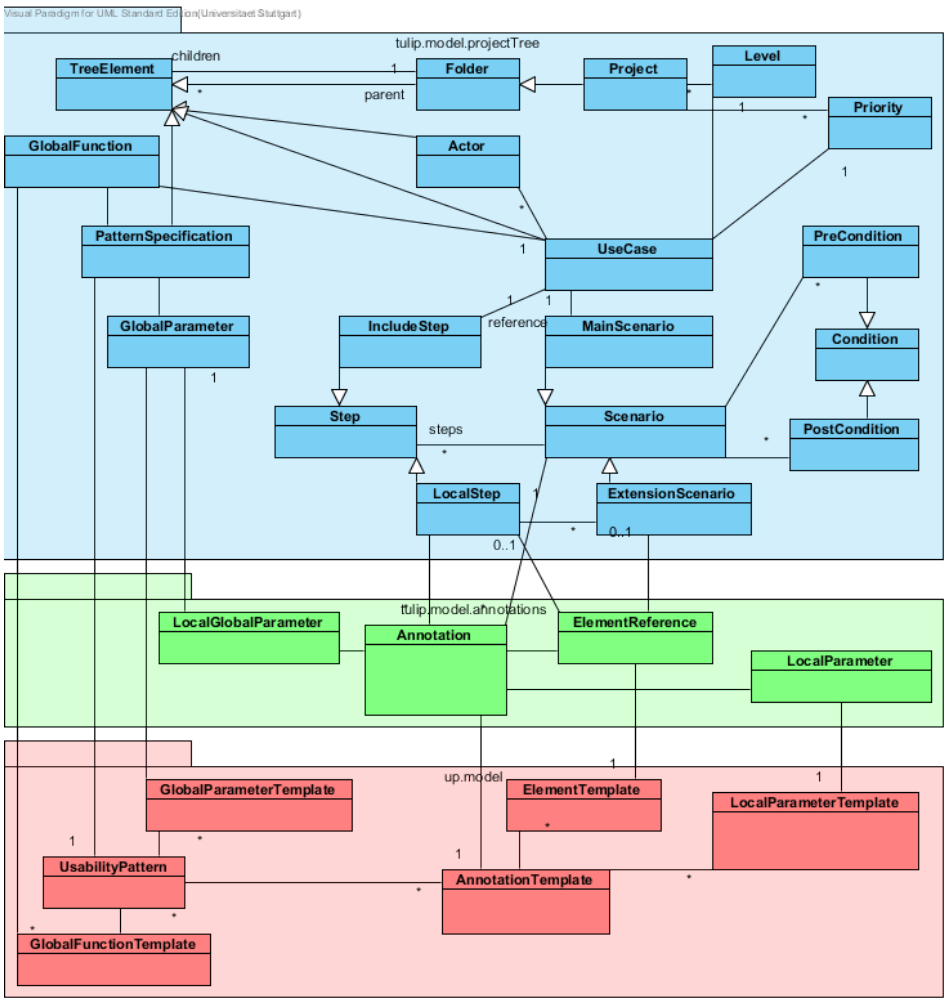


Abbildung 5.3: Datenmodell von Tulip

5.3.1 Datenmodell

Die Abbildung 5.3 bietet einen Überblick über das Datenmodell in einer an UML angelehnter Notation. Auf die Darstellung einiger Klassen und Beziehungen wurde aus Übersichtlichkeitsgründen verzichtet.

Das Paket `tulip.model.projectTree` umfasst die Projekthierarchie. Für die Abbildung der Baumstruktur wurde das Entwurfmuster *Kompositum* [GHJV04] verwendet. Die Klasse `TreeElement` repräsentiert dabei die Basiskomponente. Das Projekt ist das Wurzelelement des Baums. Die untergeordneten Use Cases, Akteure und Anwendungsspezifikationen für Usability Patterns stellen Blätterelemente des Baums dar und werden mit Hilfe von `Folder` gruppiert. Des Weiteren befinden sich auch Klassen für die Repräsentation der Bestandteile

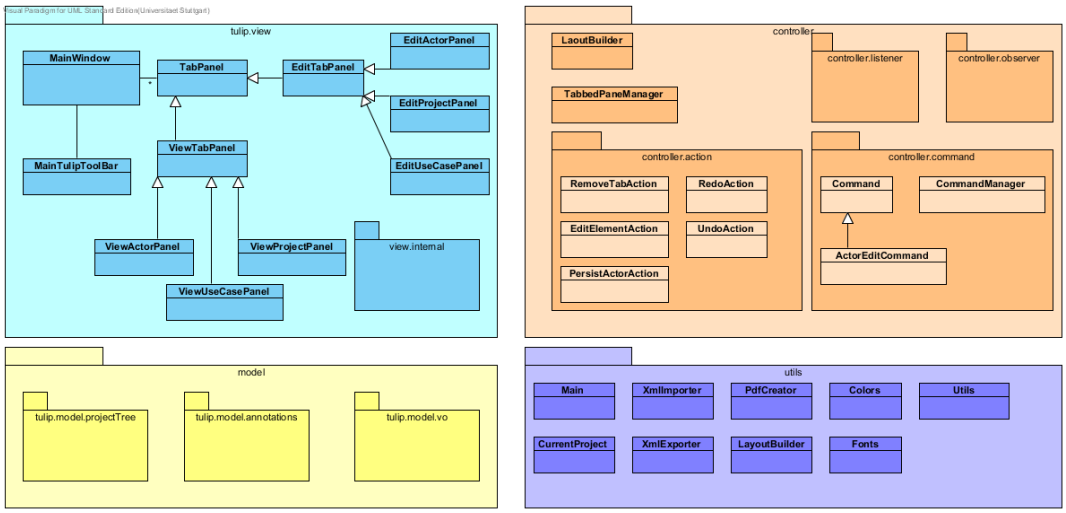


Abbildung 5.4: Architektur von Tulip

eines Projekts, eines Use Cases und einer Anwendungsspezifikation für Usability Pattern in diesem Paket.

Für das bessere Verständnis beinhaltet die Abbildung auch einen Ausschnitt aus dem Paket `up.model`. Dieses Paket ist ein Teil der Komponente *Usability Pattern Browser*. Das zentrale Element ist `UsabilityPattern`. Ein `UsabilityPattern` enthält eine Liste von Spezifikationsschablonen, repräsentiert durch `GlobalParameterTemplate`, `GlobalFunctionTemplate` und `AnnotationTemplate`. Ein `UsabilityPattern` wird im Projektbaum durch `PatternSpezifikation` instanziiert. Für die Schablonen erfolgt die Instanziierung folgendermaßen: `GlobalFunctionTemplate` und `GlobalParameterTemplate` werden entsprechend über die Klassen `GlobalFunction` und `GlobalParameter` instanziiert, welche mit einer `PatternSpecification` assoziiert sind. Eine `AnnotationTemplate` wird über eine `Annotation` aus dem Paket `tulip.model.annotations` instanziiert. Lokale Vorgaben für eine `Annotation` werden über `LocalParameter`, `LocalGlobalParameter` und `ElementReference` dargestellt. `LocalParameter` und `ElementReference` sind Instanzen von Annotationsschablonen `LocalParameterTemplate` und `ElementReferenceTemplate`, mit `LocalGlobalParameter` werden globale Vorgaben `GlobalParameter` aus der `PatternSpezification` überschrieben.

5.3.2 Komponenten

Die Abbildung 5.4 zeigt einen Ausschnitt aus dem Entwurf von *Tulip*. Für die Implementierung wurde das Architekturmuster MVC (Model View Controller) [GR01] verwendet, entsprechend sind auch die Pakete aufgeteilt.

Die Daten werden im Paket `tulip.model` verwaltet. Neben `projectTree` und `annotations` enthält dieses das Paket `vo`, welches die *Value Objects* für alle Elemente verwaltet. *Value Objects* sind die Implementierung des Entwurfsmusters *Memento* [GHJV04] und dienen zur

Zwsischenspeicherung des Zustandes eines Elements. Die Zwischenspeicherung mehrerer Zustände ist für die Erfüllung des Usability Patterns *Objektbezogenes Undo* notwendig.

Klassen für die Verwaltung der graphischen Benutzungsoberfläche befinden sich im `tulip.view`. Das Hauptfenster `MainWindow` beinhaltet drei weitere Elemente: den Projektbaum, die Haupttoolbar und die Anzeigefläche, die für die Anzeige der Inhalte verwendet wird. Das Projekt, Use Cases, Akteure und Anwendungsspezifikationen für Usability Patters werden entweder im Anzeige- oder im Bearbeiten-Modus angezeigt. Daher gibt es für die Anzeige jedes Elements jeweils zwei Klassen, die eine ist von `ViewTabPanel` und die andere von `EditTabPanel` abgeleitet. Weitere Hilfsklassen befinden sich im Paket `tulip.model.internal`.

Das Paket `tulip.controller` enthält Klassen zur Kontrolle und Steuerung der anderen Komponenten. Die Anzeige der richtigen Tabs und die rechtzeitigen Updates aller Elemente werden z.B. von der Klasse `TabbedPaneManager` gesteuert. Listener für die Ereignisse aus der GUI findet man in Paketen `listener`, `action` und `observer`. Die Undo-Funktionalität wird mittels so genannter `Commands` realisiert. Für jedes Element, welches über die Undo-Funktionalität verfügt, gibt es eine Klasse, die die abstrakte Klasse `Command` erweitert, z.B. `ActorEditCommand` für Akteure. Diese `Commands` implementieren die Methoden `undo()` und `redo()`, indem sie entweder zwei zustände des Objekts oder die Funktionen für die Wiederherstellung eines Zustandes speichern. Für jede vorgenommene Änderung wird ein `Command` angelegt. Diese werden in einem `CommandManager` verwaltet und stellen somit die Änderungshistorie eines Elements dar.

Ein weiteres Paket `tulip.utils` beinhaltet die Reportgenierung- sowie Export- und Importklassen. Außerdem werden hier allgemeine Einstellungen wie Schriftart und Farben verwaltet. Die Klasse `CurrentProject` ist nach dem Entwurfsmuster *Singleton* [GHJV04] realisiert und verwaltet alle Informationen für das aktuelle Projekt, z.B. dazugehörige GUI und den Dateinamen, unter dem das Projekt gespeichert wurde.

Die Berücksichtigung der Usability Patterns hat den Entwurf von *Tulip* erheblich beeinflusst. Die Anwendung der Patterns *Objekbezogenes Undo*, *Abbruch* und *Warnung* verlangen eine Möglichkeit der Wiederherstellung eines vorher gespeicherten Zustandes, was zur Einführung von `Commands` und *Value Objects* führte. Für die Patterns *Direkte Validierung* und *Dokumentwiederherstellung* wurden zusätzliche Funktionalitäten implementiert. Des Weiteren wirken sich alle Patterns auf die Gestaltung der graphischen Benutzungsoberfläche aus.

5.3.3 Externe Bibliotheken

Bei der Implementierung von `Tulip` wurde Gebrauch von einigen externen Open-Source-Bibliotheken gemacht.

Für die formatierten Eingaben vom Benutzer wurde der HTML-Editor von *Shef* [she09] mit einigen Änderungen eingesetzt. Diese Komponente wird vor allem für die Formatierung der Beschreibungen der Elemente eingesetzt.

Für die Implementierung der Import- und Exportfunktionalität wurde die Bibliothek *XOM* [xom11] benutzt. Diese stellt ein Framework für die Verarbeitung von XML-Objekten dar.

Für die Reportgenerierung wurden die *iText*-Bibliotheken eingesetzt. *iText for PDF* [ite11a] und *iText for RTF* [ite11b] unterstützen Erstellung und Verarbeitung von *PDF*- und *RTF*-Dokumenten.

5.4 Systemtest

Anschließend an die Implementierung von *Tulip* wurde ein Systemtest durchgeführt. Ein Systemtest dient der Überprüfung, ob die geforderte Funktionalität vollständig implementiert wurde [LL10]. Die Testdaten für den Systemtest von `Tulip` wurden aus der Spezifikation [RB11] abgeleitet. Für jeden der 26 Use Cases aus der Spezifikation wurde je ein Testfall für jede spezifizierte Reaktion des Systems auf eine Aktion des Benutzers angelegt. Danach wurden Testfälle für alle Abhängigkeiten zwischen den Use Cases erstellt. Somit wurden alle in der Use-Case-Spezifikation definierten funktionalen Anforderungen mit den Testfällen überdeckt. Da Annotationen mit Usability Patterns zusätzliche Vorgaben definieren, wurden anschließend für die Überprüfung dieser zusätzliche Testfälle abgeleitet. Die während dem Systemtest entdeckte Fehler wurden behoben, anschließend wurde das System noch einmal mit den gleichen Testdaten getestet.

Dadurch, dass die Anwendung der Usability Patterns in der Use-Case-Spezifikation spezifiziert wurde, konnten während des Systemtests neben den funktionalen Anforderungen auch die von den Patterns definierte Usability-Merkmale getestet werden.

6 Evaluation von Tulip

Dieses Kapitel beschreibt den Einsatz von *Tulip* während der Erstellung einer erweiterten Use-Case-Spezifikation. Anschließend wird die Einschätzung der Qualität der erstellten Software gemacht. Es wurde überprüft, ob die Funktionalität von *Tulip* den Anforderungen entspricht. Dabei wurde besonders darauf geachtet, inwieweit der Einsatz von *Tulip* im Vergleich zur Verwendung von Textverarbeitungsprogrammen zur Verbesserung des Prozesses der Erstellung einer mit Usability Patterns erweiterten Use-Case-Spezifikation beiträgt.

6.1 Einsatz von Tulip

Im Prozess der Evaluierung wurde die Spezifikation von *Tulip* selbst, die bereits vor der Implementierung, wie im Abschnitt 5.1 beschrieben, mittels *Microsoft Word* verfasst wurde, erneut erzeugt, dieses Mal mit der Werkzeugunterstützung von *Tulip*. Es wurden alle Use Cases in *Tulip* angelegt, Usability Patterns aus dem Katalog importiert und deren Anwendung spezifiziert, anschließend wurden Use-Case-Elemente mit Annotationen versehen. Die erweiterte Use-Case-Spezifikation wurde in *RTF*-Format exportiert und in die mittels *Microsoft Word* erstellte Anforderungsspezifikation von *Tulip* importiert. Im Folgenden werden diese Schritte in einzelnen Abschnitten beschrieben.

6.1.1 Spezifizierung der Use Cases

Für die Erstellung der Spezifikation wurde ein neues Projekt „Tulip Spezifikation“ in *Tulip* erstellt. Es wurde eine Beschreibung zum Projekt angegeben sowie Ebenen und Prioritäten angelegt. Danach wurde ein Akteurelement mit dem Namen „Entwickler“ angelegt. Anschließend wurden 26 Use Cases erstellt, gruppiert mittels Ordner, je einen pro Funktionalitätseinheit. Die Abbildung 6.1 zeigt den Projektbaum mit den Ordnern und Use Cases sowie die Anzeige des Use Cases „System starten“. Die Erstellung von Use Cases unterstützt *Tulip* mit Hilfe einer vorgefertigten Schablone. Tabelle 6.1 zeigt die Schritte bei der Spezifizierung der Use Cases auf, die von dem Entwickler durchgeführt wurden, und die Unterstützung, die für jeden Schritt von *Tulip* geleistet wurde. Außerdem wurde der Entwickler dank der Validierungsfunktion auf die Eingabefehler, wie z.B. eine fehlende Beschreibung für einen Schritt, hingewiesen.

Nachdem alle Use Cases vollständig, mit Normal- und Alternativabläufen, definiert wurden, wurde die Komponente *Pattern Browser* für die Auswahl der Usability Patterns aufgerufen.

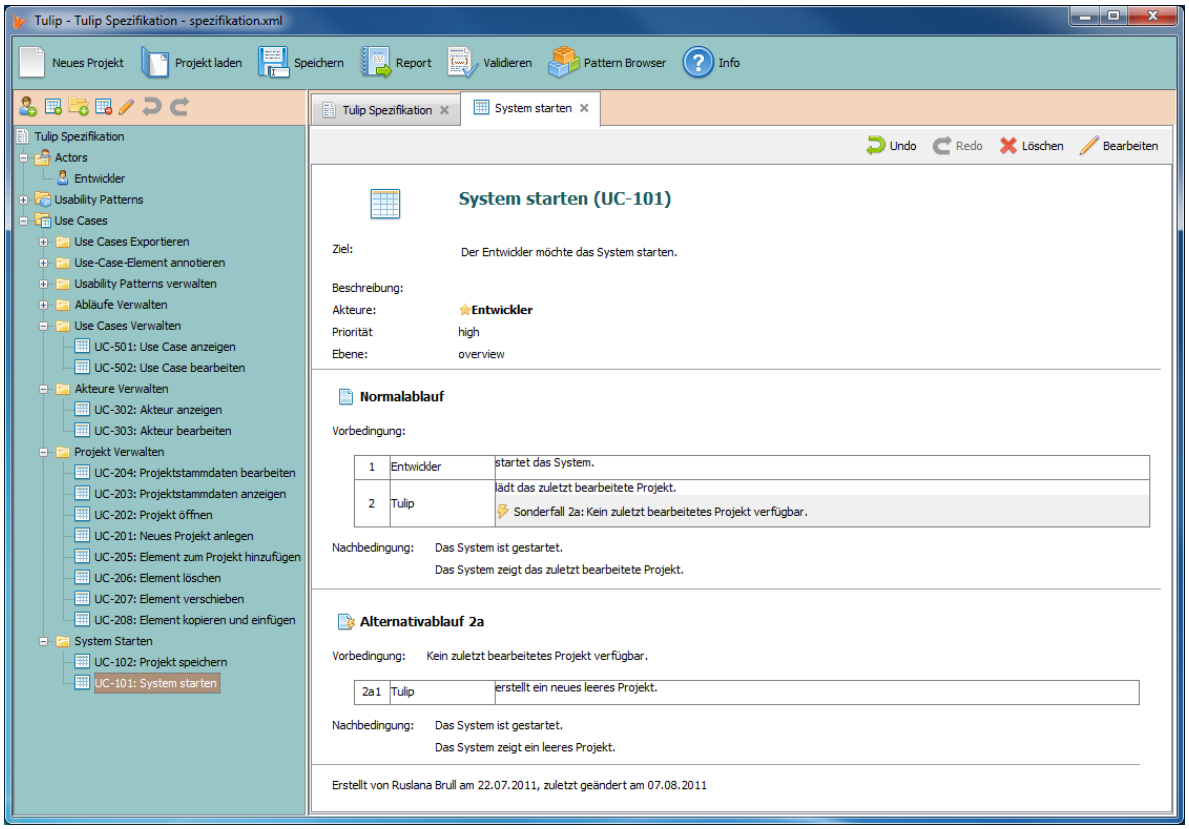


Abbildung 6.1: Projektbaum und Anzeige eines Use Cases in Tulip

Nr.	Schritt	Unterstützung durch *Tulip*
1	Projektdaten befüllen	Eingegebene Ebenen und Prioritäten werden für die Auswahl bei der Bearbeitung der Use Cases angeboten.
2	Ordner für die Funktionalitäten anlegen	
3	Use Cases anlegen	Normalabläufe werden angelegt.
4	Stammdaten für Use Cases befüllen	Es werden für Akteure, Ebenen und Prioritäten Auswahloptionen angeboten.
5	Schritte für Abläufe definieren	Schritte werden nummeriert, es werden Auswahloptionen für den Akteur angeboten.
6	Sonderfälle anlegen	Für jeden Sonderfall wird ein Alternativablauf angelegt. Der Name für den Ablauf wird entsprechend der von Cockburn in [Coco7] definierten Konvention generiert.

Tabelle 6.1: Spezifizierung der Use Cases

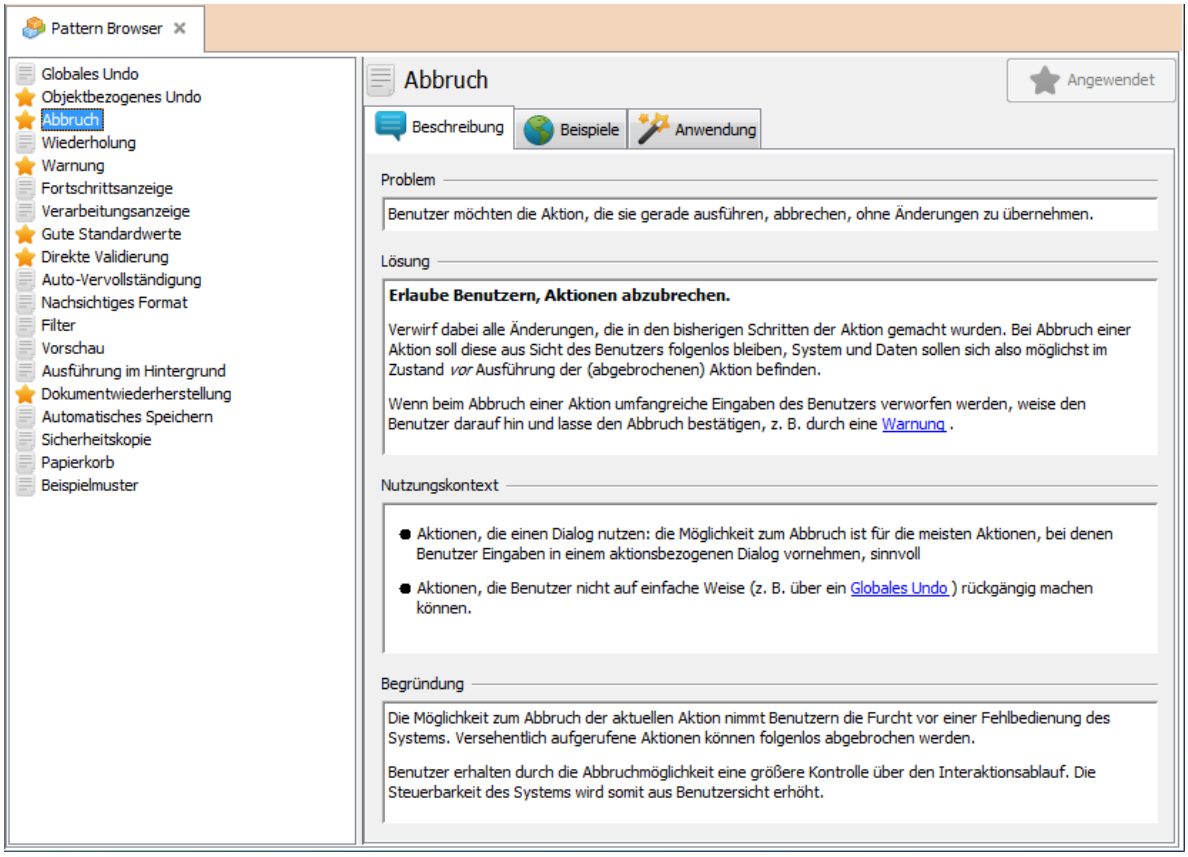


Abbildung 6.2: Pattern Browser in Tulip

6.1.2 Pattern Browser

Die in *Tulip* eingebettete Komponente *Pattern Browser* repräsentiert den Katalog „Usability Patterns“ [Röd11a]. Sie bietet eine strukturierte Übersicht über die vorhandenen Usability Patterns und verwaltet detaillierte Beschreibungen und Schablonen für die Anwendung. Der Browser ermöglicht die Navigation durch alle Usability Patterns sowie die Auswahl der Patterns, um die die Use-Case-Spezifikation erweitert werden soll. Die Abbildung 6.2 zeigt die Benutzungsoberfläche für die Pattern-Browser-Komponente und die Beschreibung des Patterns „Abbruch“.

Die sechs Usability Patterns, die in die Spezifikation übernommen werden, wurden im Pattern-Browser ausgewählt. Wie man der Abbildung entnehmen kann, wurden diese mit einem gelben Stern markiert und können nicht mehr angewendet werden.

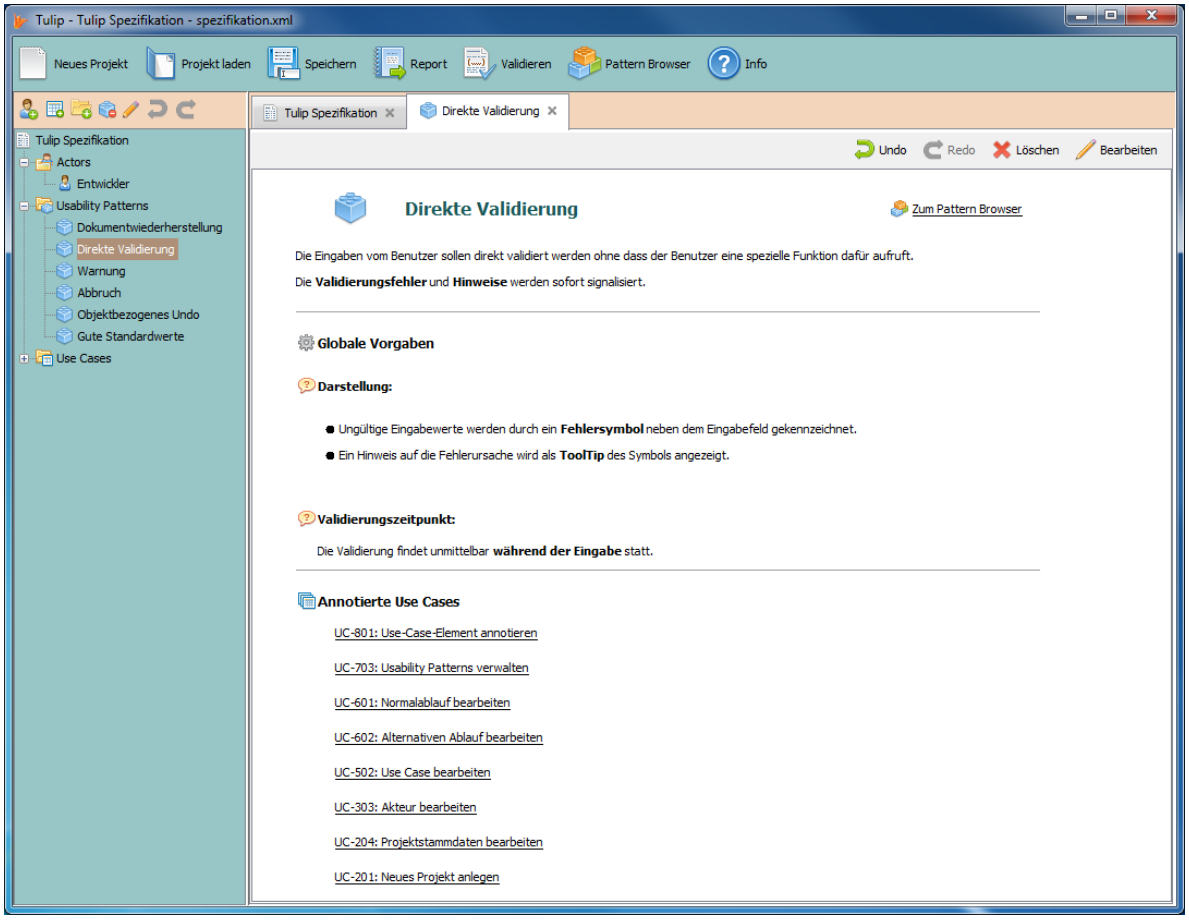


Abbildung 6.3: Spezifikation der Anwendung für Usability Pattern „Direkte Validierung“ in Tulip

6.1.3 Spezifizierung der Anwendung der Patterns

Für die im Browser ausgewählten Patterns erstellt *Tulip* Elemente, die die Anwendung der Patterns spezifizieren. Der Name wird aus dem Katalog übernommen. Es werden auch entsprechend den im Katalog definierten Schablonen globale Parameter und globale Funktionen angelegt. Optionale Vorgaben können vom Entwickler nach Bedarf hinzugefügt werden. Dem Entwickler bleibt es lediglich, die Anwendung der angelegten Parameter für die Software zu beschreiben. Abbildung 6.3 zeigt die Spezifikation der Anwendung des Patterns *Direkte Validierung* mit einer Beschreibung und globalen Parametern *Darstellung* und *Validierungszeitpunkt*.

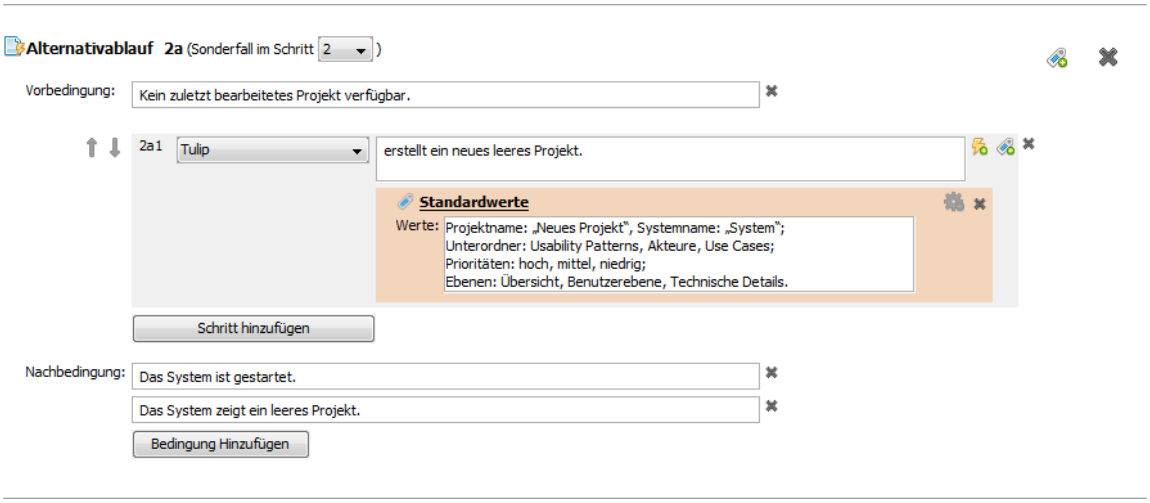


Abbildung 6.4: Bearbeiten eines Use-Case-Ablaufs in Tulip

6.1.4 Annotierung der Use Cases

Die Annotation der Use Cases erfolgt entsprechend den in den Usability Patterns definierten Annotationsschablonen. In *Tulip* wird für jeden Use Case, Ablauf und Schritt eine Liste mit Schablonen angeboten, die für die Annotation dieses Elements geeignet sind. Der Entwickler muss lediglich eine Schablone auswählen, daraufhin wird das Element mit einer Annotation, die mit dem Usability Pattern verlinkt ist, versehen. Die obligatorischen Parameter werden ebenfalls automatisch angelegt. Die Abbildung 6.4 zeigt beispielhaft die Bearbeitung des *Alternativablaufs 2a* für den Use Case *System starten*. Der Schritt *2a1* wurde mit dem Pattern *Standardwerte* annotiert.

6.1.5 Generierung eines Reports

Anschließend wurde ein *RTF*-Report von *Tulip* generiert [Tul11]. Dieser enthält Informationen über das Projekt, Anwendungsspezifikationen für die verwendeten Patterns, sowie eine tabellarische Darstellung aller Use Cases. Die Annotationen sind in die Tabellen eingebettet. Außerdem weist der Report für jedes Usability Pattern eine Liste von Use Cases auf, deren Elemente mit diesem Pattern annotiert sind. Abbildung 6.5 zeigt einen Ausschnitt aus dem Report, auf dem der Use Case „System starten" abgebildet ist. Der generierte Report wurde anschließend in die Anforderungsspezifikation von Tulip [RB11] an Stelle von der alten Use-Case-Spezifikation aufgenommen.

Pre Condition:		Kein zuletzt bearbeitetes Projekt verfügbar.
2a1	Tulip	erstellt ein neues leeres Projekt.
		Standardwerte Werte: Projektname: „Neues Projekt“, Systemname: „System“; Unterordner: Usability Patterns, Akteure, Use Cases; Prioritäten: hoch, mittel, niedrig; Ebenen: Übersicht, Benutzerebene, Technische Details.
Post Condition:		Das System ist gestartet. Das System zeigt ein leeres Projekt.
Extension Scenario 2b		
Pre Condition:		System wurde bei der letzten Ausführung unvorhergesehen beendet (z. B. aufgrund eines Systemfehlers).
2b1	Tulip	fragt Entwickler, ob die zuletzt bearbeiteten Projektdaten aus den Wiederherstellungsinformationen wiederhergestellt werden sollen.
2b2	Entwickler	bestätigt die Wiederherstellung.
2b3	Tulip	stellt die Projektdaten wieder her.
Post Condition:		Das System ist gestartet. Das System zeigt die wiederhergestellten Projektdaten.

Abbildung 6.5: Ausschnitt aus dem Use Case „System starten“ im RTF-Report

6.2 Bewertung von Tulip als Spezifikationswerkzeug

Das Werkzeug *Tulip* eignet sich dafür, eine Use-Case-Spezifikation zu erstellen und diese mit Usability Patterns aus dem Katalog zu erweitern. Dank der Automatisierung vieler Prozesse und dem Wegfallen des Formatierungsaufwands verläuft der Prozess der Use-Case-Modellierung und anschließender Annotation mit Anwendung von *Tulip* wesentlich schneller als mit gängigen Textverarbeitungsprogrammen. Durch die Validierungsfunktion, die den Entwickler auf fehlende und inkorrekte Eingaben hinweist, wird die Konsistenz in der Verwendung der Usability Patterns sichergestellt und die Qualität der Use-Case-Spezifikation erhöht. Auch alle erstellten Verweise werden bei Änderungen stets aktualisiert, bei der Verwendung eines Textverarbeitungsprogramms ist der Entwickler bei der Pflege dieser auf sich alleine gestellt.

Ein Nachteil gegenüber einem Textverarbeitungsprogramm ist allerdings der entstehende Medienbruch. Wenn z.B. mit *Word* gearbeitet wird, kann die gesamte Anforderungsspezifikation inklusive einer Use-Case-Spezifikation im gleichen Dokument erstellt und verwaltet werden. Falls für die Erstellung und Pflege der Use-Case-Spezifikation *Tulip* eingesetzt wird, muss diese entweder in ein separates Dokument ausgelagert oder nach jeder Änderung neu importiert werden.

Da *Tulip* in erster Linie für die Evaluierung des Konzeptes „Usability Patterns“ entwickelt wurde, ist die Funktionalität des Werkzeugs auf die strukturierte Erfassung von Use Cases mit der Unterstützung von Usability Patterns begrenzt. Im Vergleich zu den anderen Werkzeugen zur Use-Case-Modellierung (vergleiche Abschnitt 4.2) erscheint diese recht eingeschränkt. Es gibt in *Tulip* z.B. keine Möglichkeit, andere Anforderungen oder Dokumente zu verwalten, es werden keine Use-Case-Diagramme erstellt, keine Rechtschreibprüfung für Benutzereingaben durchgeführt. *Tulip* ist zur Zeit ein Einbenutzersystem, das heißt, dass die kollaborative Arbeit nicht unterstützt wird. Es wäre allerdings denkbar, diese und weitere Funktionalitäten in spätere Versionen von *Tulip* einzubauen, so dass die gesamte Anforderungsspezifikation mit Hilfe von *Tulip* erzeugt und gepflegt werden kann.

6.3 Bewertung der Qualität der Software

In diesem Abschnitt wird die Produktqualität der im Rahmen dieser Arbeit erstellten Software anhand der von [LL10] definierten Teilqualitäten bewertet. Die Abbildung 6.6 zeigt die Gliederung des Qualitätsbegriffs nach [LL10]. Da bei der Realisierung von *Tulip* in erster Linie das Ziel verfolgt wurde, die praktische Anwendung des Konzeptes „Usability Patterns“ zu evaluieren, wurde auf die Erhebung der nichtfunktionalen Anforderungen, die für die Benutzung der Software von breiteren Anwendergruppen relevant wären, vorerst verzichtet. Aus diesem Grund können manche Qualitätskriterien nur grob oder gar nicht eingeschätzt werden.

6.3.1 Zuverlässigkeit

Die *Korrektheit* der Software wurde wie im Abschnitt 5.4 beschrieben anhand der Anforderungsspezifikation [RB11] geprüft. Die dabei festgestellten Mängel wurden behoben, die Software wurde anschließend nochmal getestet. Somit lässt sich die *Korrektheit* als hoch einstufen.

Im Laufe der Evaluierung hat das Werkzeug die erwartete Funktionalität erbracht. Allerdings ist die *Ausfallsicherheit* nur schwer einschätzbar, da *Tulip* nicht mit großen Datenmengen getestet wurde. Es wurden auch keine speziellen Anforderungen diesbezüglich an die Software gestellt.

Das *Genauigkeit*-Kriterium ist für die Software wenig relevant, da diese nur eindeutige Resultate erzeugt znd somit eine Abweichung von der *Korrektheit* nicht in Frage kommt.

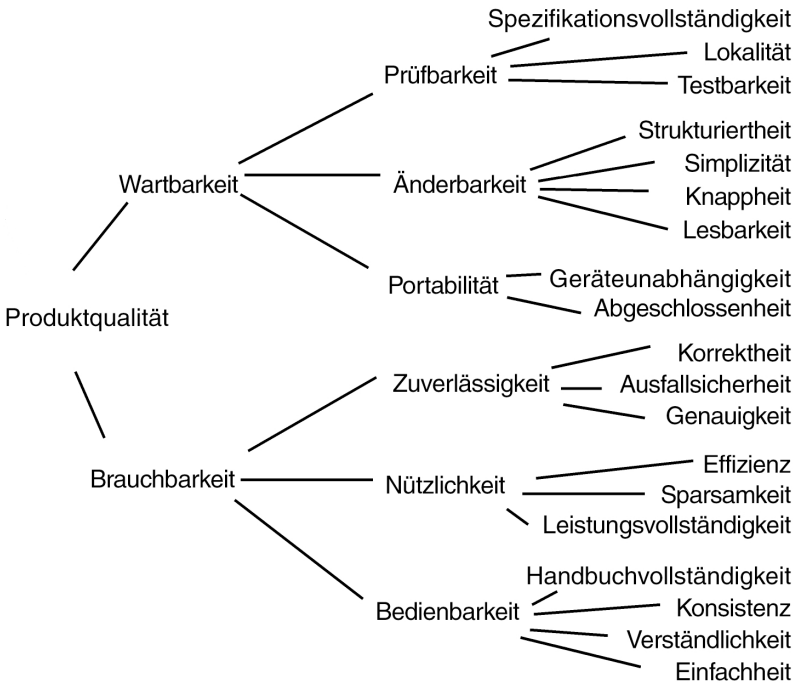


Abbildung 6.6: Qualitätsbaum [LL10]

6.3.2 Nützlichkeit

Die *Effizienz* und *Sparsamkeit* lagen bei der Evaluierung der Software hoch. Es wurden keine Einbuße in Rechenzeit oder im Speicherverbrauch beobachtet.

Die *Leistungsvollständigkeit* ist gegeben. Alle in der Spezifikation beschriebenen hochpriorisierten Funktionalitäten wurden umgesetzt.

6.3.3 Bedienbarkeit

Der Aspekt der *Handbuchvollständigkeit* ist für *Tulip* nicht relevant. Es wurden keine Handbücher für *Tulip* erstellt, da die mit Screenshots versehene ausführliche Spezifikation einen guten Überblick über die Benutzung der Software bietet.

Die hohe *Konsistenz* wurde durch die durchgehende Wiederverwendung der Komponenten für die Benutzungsoberfläche sowie durch die Anwendung der Usability Patterns *Warnung* und *Abbruch* [Röd11a] erreicht.

Die hohe *Verständlichkeit* wird durch die Anwendung des Usability Patterns *Direkte Validierung* [Röd11a] erreicht. Durch den gesamten Use-Case-Modellierungsprozess wird der Entwickler mittels Warnungen auf falsche oder fehlende Eingaben hingewiesen.

Die *Einfachkeit* der Software lässt sich als hoch einschätzen, da die Bedienung intuitiv ist und meist aus den Texteingaben in vorgefertigte Felder besteht. Allerdings sollte der Entwickler mit der Struktur der textbasierten Use-Case-Beschreibung vertraut sein.

6.3.4 Prüfbarkeit

Die *Spezifikationsvollständigkeit* von *Tulip* ist hoch, da dies die Grundlage für die spätere Evaluierung des Werkzeugs war. Alle funktionalen und Usability-Anforderungen an die Software wurden vollständig mit Hilfe von Use Cases in [RB11] spezifiziert und priorisiert.

Die *Lokalität* der Software ist dadurch gegeben, dass diese als eine isolierte Anwendung implementiert wurde. Der Zugriff auf systemfremde Daten erfolgt nur über die Komponente *Pattern Browser*. Dabei werden die Daten lediglich ausgelesen und in keiner Weise verändert. Somit werden die Fernwirkungen in der Software vermieden.

Die *Testbarkeit* von *Tulip* ist hoch, da jede Eingabesituation reproduzierbar ist und die damit verbundenen visuellen Auswirkungen immer nachvollziehbar sind.

6.3.5 Änderbarkeit

Durch die Verwendung von Entwurfmustern bei der Implementierung ist die Software in logisch abgeschlossene Einheiten gegliedert. Somit wird eine hohe *Strukturiertheit* erreicht.

Die *Simplizität* der Software ist relativ hoch, da diese nur wenige schwer verständliche Konstruktionen enthält. Wobei für das Verständnis einiger Komponenten, wie z.B. Import- und Export-Funktionalität, die Einarbeitung in die eingebundenen fremden Bibliotheken notwendig ist.

Die *Knappheit* der Software wurde durch die Wiederverwendung einiger Komponenten und die Vermeidung von Redundanz sichergestellt.

Um die *Lesbarkeit* des Codes sicherzustellen, wurde bei der Implementierung auf die Richtlinien für die Programmierung in Java geachtet. Es wurden stets ausdrucksstarke Bezeichner in englischer Sprache verwendet und der Code wurde durchgehend mit Kommentaren versehen.

6.3.6 Portabilität

Die *Geräteunabhängigkeit* ist durch die Verwendung der Programmiersprache Java gegeben. Dadurch ist die Software auf jedem System lauffähig, auf dem Java Virtual Machine installiert ist. Allerding wurde für die Darstellung der graphischen Oberfläche zwecks des besseren Erscheinungsbilds auf die Verwendung eines auf allen Plattformen gleich aussehenden Look-and-Feels verzichtet. Es wurde das dem systemüblichen angepasste `SystemLookAndFeel` verwendet, dabei wurde die Gestaltung der Benutzungsoberfläche hauptsächlich für das Betriebssystem Windows optimiert.

In der aktuellen Implementierung von *Tulip* ist der Datenaustausch mit anderen Systemen nur mittels Import- und Exportfunktionalität gegeben. Somit ist die *Abgeschlossenheit* der Software garantiert.

7 Bewertung des Konzeptes „Usability Patterns“

Im Rahmen dieser Arbeit wurde das Konzept der Usability Patterns bei der Entwicklung eines Softwareproduktes eingesetzt. In diesem Kapitel wird der Einsatz des Konzeptes „Usability Patterns“ in einzelnen Phasen des Softwareentstehungsprozesses einer Bewertung unterzogen.

7.1 Anforderungsanalyse

Bei der Anforderungsanalyse werden Usability-Anforderungen oft vergessen oder zusammen mit anderen nichtfunktionalen Anforderungen nur am Rande berücksichtigt. Der Usability-Patterns-Katalog bietet eine Sammlung der strukturiert beschriebenen Anforderungen, die die Usability der Software steigern. Dadurch, dass diese Anforderungen bereits beschrieben und zusammengefasst sind, wird der Prozess der Erhebung von Usability-Anforderungen deutlich vereinfacht.

Bei den Usability-Anforderungen handelt es sich meistens um so genannte weiche Anforderungen, da sie nicht quantifiziert werden können. Weiche Anforderungen sind dadurch gekennzeichnet, dass es einen fließenden Übergang zwischen richtig und falsch gibt [LL10]. Solche Anforderungen sind schwer zu erheben und zu formulieren. An dieser Stelle bietet der *Usability Patterns Katalog* [Röd11a] eine geeignete Grundlage, um die Usability-Anforderungen mit dem Kunden zu diskutieren und diese dank den Schablonen aus dem Katalog zu quantifizieren.

Auch ist es dem Kunden oft nicht bewusst, welche Usability-Merkmale die Software aufweisen soll. Diese Anforderungen bleiben latent, bis der Kunde gezielt danach gefragt wird oder bis er die fertige Software sieht. In diesem Fall kann der Katalog mit zahlreichen Beispielen für den Einsatz jedes Patterns dazu verhelfen, die latenten Kundenwünsche zu identifizieren und diese zu beschreiben.

Allerdings muss man sich bewusst sein, dass nicht alle Usability-Anforderungen mit Hilfe von Usability Patterns erhoben werden können. Bei Usability Patterns handelt es sich um Usability-Merkmale, die einen erheblichen Einfluss auf die Funktionalität der Software haben und daher als eine Ergänzung zu den funktionalen Anforderungen formuliert werden können. Für die Erhebung der Anforderungen bezüglich der Gestaltung der UI sollen entsprechende Techniken, wie z.B. *UI-Prototyping* (vergleiche Abschnitt 2.1.2) eingesetzt werden.

7.2 Spezifikation

Bei der Spezifikation der Anforderungen stehen funktionale Merkmale im Vordergrund, da Erfüllung dieser den Nutzen der Software ausmachen. Funktionale Anforderungen lassen sich recht präzise fassen und werden z.B. mittels Use Cases in einer semiformalen Form festgehalten. Die nichtfunktionalen Anforderungen sind dagegen oft weich und vage und werden in natürlicher Sprache formuliert. Usability-Merkmale, die im Grenzbereich zwischen funktionalen und nichtfunktionalen Anforderungen liegen, werden üblicherweise zu den nichtfunktionalen gezählt und entsprechend nicht strukturiert beschrieben.

Das Konzept der Usability Patterns hilft dabei, die Usability-Merkmale, welche die Funktionalität der Software betreffen, zu identifizieren und bieten ein Modell für die Integrierung dieser in Use Cases. Somit werden mittels Use Cases beschriebene funktionale Anforderungen um dazugehörige Usability-Merkmale erweitert. Über Usability Patterns definierte Merkmale werden strukturiert und konsistent mittels erweiterter Use-Cases beschrieben. Dies erhöht die Qualität der Anforderungsspezifikation als Referenz für weitere Artefakte, da diese jetzt auch quantitative Vorgaben für Usability-Anforderungen beinhaltet.

Ein Nachteil der erweiterten Spezifikation ist, dass diese mühsam zu pflegen ist. Die Erweiterungen erhöhen die Komplexität der Use Cases, die üblicherweise in Form von Tabellen mit Hilfe eines Textverarbeitungsprogramms erstellt werden. Use-Case-Erweiterungen benötigen syntaxische Validierung und Konsistenzprüfung, somit ist eine Werkzeugunterstützung für die Erstellung einer erweiterten Use-Case-Spezifikation wünschenswert. Der Einsatz des im Rahmen dieser Arbeit entwickelten Werkzeug *Tulip* für die Erstellung einer erweiterten Use-Case-Spezifikation hat gute Ergebnisse gezeigt (vergleiche Abschnitt 6.2), allerdings gibt es zurzeit kein Werkzeug für die Erstellung der kompletten Anforderungsspezifikation, welches die Integration der Usability Patterns unterstützt.

7.3 Entwurf und Implementierung

Die annotierte Use-Case-Spezifikation beschreibt Usability-Anforderungen eng verzahnt mit den Funktionalitäten, die sie betreffen. Dies versetzt den Entwickler in die Lage, die Usability-Merkmale bereits bei der Realisierung der entsprechenden Funktionalität zu berücksichtigen und nicht erst später, in der UI-Design-Phase. Dadurch wird die Anpassung der Architektur und Datenstruktur einer Software an diese Anforderungen in frühen Phasen des Entwicklungsprozesses sichergestellt.

Allerdings muss man beachten, dass die erweiterte Use-Case-Spezifikation nur eine Referenz für den Entwickler darstellt, es gibt keine Techniken, die Use Cases in Bausteine eines Softwareentwurfs umwandeln. Der Entwickler muss in einem iterativen Prozess den Entwurf ausarbeiten und die Erfüllung der Anforderungen aus der Spezifikation überprüfen.

Bei der Implementierung erwiesen sich die detaillierten Vorgaben zu den Usability-Merkmalen als durchaus hilfreich. Üblicherweise wird der Entwickler auf sich alleine gestellt,

was die Usability der implementierten Funktionalität angeht. Im besten Fall werden allgemeingültige Vorgaben in Form einer Richtlinie oder eines Styleguides gemacht. Die Vorgaben für die konkrete Funktionalität müssen dann von dem Entwickler selber abgeleitet werden. Dies kann zur Inkonsistenz im Erscheinungsbild führen, wenn z.B. mehrere Entwickler an der Implementierung arbeiten. Mit dem Konzept der Usability Patterns wird sichergestellt, dass die allgemeingültige globale Vorgaben eines Merkmals sowie die Vorgaben für einzelne Funktionalitäten stets konsistent bleiben.

7.4 Testphase

In der Testphase wird die Erfüllung der an die Software gestellten Anforderungen durchgeführt. Für weiche Anforderungen, die keine klaren Vorgaben definieren, können auch keine aussagekräftige Testdaten erstellt werden. Dank Usability Patterns können die in Form der Annotationen in Use Cases enthaltene Usability-Merkmale, wie im Abschnitt 5.4 beschrieben, direkt in Testfälle für einen Systemtest überführt werden. Somit kann die Erfüllung der über Usability-Patterns definierten Usability-Anforderungen zusammen mit anderen funktionalen Anforderungen an die Software in einem Systemtest getestet werden.

7.5 Zusammenfassung

Die Anwendung des Konzeptes „Usability Patterns“ in einem Softwareentwicklungsprozess bringt einige Vorteile für die Entwickler mit sich. In der Anforderungsanalysephase bietet der Usability-Pattern-Katalog eine Unterstützung bei der Identifizierung und Abstimmung bestimmter Usability-Anforderungen. In der Spezifikationsphase werden diese Anforderungen nicht mehr als nichtfunktionale behandelt, sondern strukturiert und konsistent zusammen mit funktionalen Anforderungen beschrieben. Eine solche Anforderungsspezifikation kann als Referenz bei der Realisierung und beim Testen der Usability-Anforderungen verwendet werden.

Man muss allerdings beachten, dass nicht alle Usability-Anforderungen als ein Usability-Pattern beschrieben werden können. Dieses Konzept kann nur für die Merkmale verwendet werden, welche die Funktionalität der Software betreffen. Auch ist das Konzept noch relativ neu und umfasst nur einige wenige Usability-Probleme, die am häufigsten auftreten. Es gibt auch zurzeit keine ausreichende Werkzeugunterstützung für die Anwendung des Konzeptes, wobei die Evaluation

8 Zusammenfassung und Ausblick

Im Rahmen dieser Arbeit wurde das Konzept der Usability Patterns an einem praktischen Beispiel angewendet, evaluiert und verbessert. Dabei entstand ein Werkzeug, welches die Entwickler bei der Anwendung des Konzeptes unterstützt.

Als Erstes wurde die Use-Case-Struktur um zusätzliche Elemente, so genannte *Annotationen*, erweitert, welche die Spezifizierung der Usability-Merkmale ermöglichen. Eine erweiterte Use-Case-Spezifikation beschreibt sowohl funktionale als auch mittels Usability Patterns beschriebene Usability-Merkmale des Systems und trägt dazu bei, dass die Entwickler bei der Erstellung des Entwurfs und des Codes auch Usability-Aspekte berücksichtigen.

Das Konzept der Usability Patterns wurde bei der Entwicklung eines Use-Case-Editors *Tulip* angewendet und evaluiert. Während der Anforderungsanalyse wurden passende Usability Patterns identifiziert, woraufhin eine Use-Case-Spezifikation für *Tulip* erstellt und mit den ausgewählten Patterns erweitert wurde. Die spezifizierten Usability Patterns konnten somit beim Entwurf und der Implementierung der Software sowie bei der Erstellung der Testdaten berücksichtigt werden.

Im Laufe der Arbeit entstand ein Werkzeug zur Erstellung und Bearbeitung einer Use-Case-Spezifikation mit Unterstützung für erweiterte Use Cases und Usability Patterns. Das Werkzeug bietet eine Komponente zur Anzeige der vorhandenen Patterns aus dem Katalog und die Funktionalität für die Übernahme dieser in die Use-Case-Spezifikation. Desweiteren ist es möglich, Use Cases zu erstellen, die Anwendung der Usability Patterns zu spezifizieren und Use-Case-Elemente mit den Patterns zu annotieren. Die erweiterte Use-Case-Spezifikation kann in PDF- und RTF-Format exportiert und in andere Dokumente, wie z.B. in die Anforderungsspezifikation, integriert werden.

Das erstellte Werkzeug *Tulip* wurde an einem praktischen Beispiel evaluiert, indem die eigene Spezifikation nochmal erstellt wurde. Dabei ließ sich erkennen, dass der Prozess der Erstellung einer erweiterten Use-Case-Spezifikation mit einer Werkzeugunterstützung deutlich optimiert werden kann, indem die Strukturierung und die syntaktische Validierung von dem Werkzeug übernommen werden.

Ausblick

Der im Rahmen dieser Arbeit durchgeführte Einsatz der Usability Patterns im Entwicklungsprozess konnte erste positive Erkenntnisse über die Brauchbarkeit und Praxistauglichkeit des

Konzeptes liefern. Weitere Schritte können und sollen in mehrere Richtungen unternommen werden.

Der aktuelle Pattern-Katalog beschreibt nur einige geläufige Usability-Merkmale. Um ein breiteres Spektrum an Anforderungen zu unterstützen, können weitere spezifische Usability Patterns ausgearbeitet werden. Für eine bessere Übersichtlichkeit kann eine Kategorisierung der Patterns oder sogar die Verwendung mehrerer Pattern-Kataloge in Betracht gezogen werden.

Die Werkzeugunterstützung durch *Tulip* trägt zur Optimierung des Prozesses der Erstellung einer mit Usability Patterns erweiterten funktionalen Use-Case-Spezifikation bei. Im Rahmen dieser Arbeit wurden die Grundfunktionen für die Verwaltung der Use Cases und Annotierung dieser mit Usability Patterns realisiert. Eine Weiterentwicklung von *Tulip* wäre für die Unterstützung der Entwickler bei der Anwendung des Konzeptes von Vorteil. Es sind folgende funktionale Erweiterungen zu empfehlen:

- Erstellung von Use-Case-Diagrammen zusätzlich zu den Use-Case-Beschreibungen
- Erfassung zusätzlicher Elemente, wie z.B. nichtfunktionalen Anforderungen
- Pflege eines Glossars
- Bereitstellung der Arbeitsumgebung für mehrere Benutzer
- Anpassungsmöglichkeit der Struktur der exportierten Dokumente
- Rechtschreibprüfung

Literaturverzeichnis

[ABC03] S. Adolph, P. Bramble, A. Cockburn. *Patterns for effective use cases*. Addison-Wesley, Boston, 2003. (Zitiert auf Seite 31)

[Cas11] Use Cases and Requirements Management - CaseComplete, 2011. URL http://www.casecomplete.com/. (Zitiert auf Seite 44)

[Coc07] A. Cockburn. *Writing effective use cases*. Addison-Wesley, Boston, 18. print. edition, 2007. (Zitiert auf den Seiten 29 und 66)

[DIN06] DIN EN ISO 9241-110: Ergonomische Gestaltung von Benutzungsschnittstellen : Kommentar zur Grundsatznorm, 2006. (Zitiert auf Seite 19)

[GHJV04] E. Gamma, R. Helm, R. Johnson, J. M. Vlissides. *Design patterns: Elements of reusable object-oriented software*. Addison-Wesley, Reading, Mass, 31. printing. edition, 2004. (Zitiert auf den Seiten 60, 61 und 62)

[GR01] E. Gamma, D. Riehle. *Entwurfsmuster: Elemente wiederverwendbarer objektorientierter Software*. Addison-Wesley, München ;, Boston [u.a.], 5., korrigierter nachdr. edition, 2001. (Zitiert auf Seite 61)

[Hec09] G. Hecke. Bachelorarbeit. Semantische Erweiterung von Glossareinträgen zur automatisierten Überprüfung der korrekten Benutzung. *Leibnitz Universität Hannover*, 2009. (Zitiert auf Seite 46)

[HeR11] HEuristic Requirements Assistant, 2011. URL https://trac.se.uni-hannover.de/trac/hera/wiki. (Zitiert auf Seite 46)

[ite11a] iText ® - Free / Open Source PDF Library for Java and C#, 2011. URL http://itextpdf.com/. (Zitiert auf Seite 63)

[ite11b] iText RTF library | Download iText RTF library software for free at SourceForge.net, 2011. URL http://sourceforge.net/projects/itextrtf/. (Zitiert auf Seite 63)

[Jac92] I. Jacobson. *Object-oriented software engineering: A use case approach*. ACM Press, Harlow, Essex, 1992. URL http://www.worldcat.org/oclc/638307356. (Zitiert auf Seite 29)

[JMSS07] N. Juristo, A. M. Moreno, M.-I. Sanchez-Segura. Guidelines for Eliciting Usability Functionalities. *IEEE Transactions on Software Engineering*, 33(11), 2007. (Zitiert auf den Seiten 25 und 26)

[Kar90] C.-M. Karat. *Cost-benefit analysis of usability engineering techniques.* Proceedings of the Human Factors Society, Orlando, Florida, 1990. (Zitiert auf Seite 19)

[LL10] J. Ludewig, H. Lichter. *Software Engineering: Grundlagen, Menschen, Prozesse, Techniken*. dpunkt-Verl, 2010. (Zitiert auf den Seiten 11, 21, 25, 29, 63, 71, 72 und 75)

[RB11] H. Röder, R. Brull. Erweiterte Use-Case-Spezifikation für Tulip, 2011. (Zitiert auf den Seiten 53, 63, 69, 71 und 73)

[RE-07] Portal für Anforderungsmanagement, 2007. URL http://re-wissen.de/Wissen/. (Zitiert auf den Seiten 28 und 32)

[rem11] remasystem - Requirements management system - Google Project Hosting, 2011. URL http://code.google.com/p/remasystem/. (Zitiert auf Seite 48)

[RF07] M. Richter, M. Flückiger. *Usability Engineering kompakt: Benutzbare Software gezielt entwickeln*. Elsevier, München, 1 edition, 2007. (Zitiert auf Seite 20)

[Röd10] H. Röder. Using Interaction Requirements to Operationalize Usability. *Proceedings of the 25th ACM Symposium on Applied Computing*, 2010. (Zitiert auf Seite 25)

[Röd11a] H. Röder. Katalog "Usability Patterns", 2011. (Zitiert auf den Seiten 27, 34, 43, 67, 72 und 75)

[Röd11b] H. Röder. A Pattern Approach to Specifying Usability Features in Use Cases. *Proceedings of the 2nd Int'l Workshop on Pattern-Driven Engineering of Interactive Computing Systems*, 2011. (Zitiert auf den Seiten 11, 26, 35 und 36)

[she09] SHEF - Swing HTML Editor Framework, 2009. URL http://shef.sourceforge.net/. (Zitiert auf Seite 62)

[Som07] S. Somé. UCEd Use Cases development approach, 2007. (Zitiert auf Seite 48)

[Tul11] Tulip. Generierte erweiterte Use-Case-Spezifikation für Tulip, 2011. (Zitiert auf Seite 69)

[UCE11] Use Case Editor, 2011. URL http://sourceforge.net/projects/uced/. (Zitiert auf Seite 47)

[xom11] XOM - XML object model, 2011. URL http://www.xom.nu/. (Zitiert auf Seite 63)

Alle URLs wurden zuletzt am 1.08.2011 geprüft.

Erklärung

Hiermit versichere ich, diese Arbeit selbständig verfasst und nur die angegebenen Quellen benutzt zu haben.

Stuttgart, den 8. August 2011

(Ruslana Brull)