1 Einleitung und Überblick

Das einleitende Kapitel gibt einen Überblick über den Inhalt dieser Arbeit. Zunächst wird die Motivation vorgestellt, die der Arbeit zugrunde liegt. Im anschließenden Abschnitt 1.2 werden das Ziel der Arbeit erläutert und das prinzipielle Vorgehen sowie die dabei erzielten Ergebnisse skizziert. Abschnitt 1.3 beschreibt den Aufbau der Arbeit und den Inhalt der einzelnen Kapitel.

1.1 Motivation

Software ist aus der heutigen Welt nicht mehr wegzudenken. Software ist allgegenwärtig, ist in Flugzeugen und Mobiltelefonen zu finden, steuert Handelsplattformen und Waschmaschinen, verwaltet betriebliche Informationen und zeigt virtuelle Welten auf. Mit der zunehmenden Ubiquität hat sich das Profil der Benutzer von Software-Systemen gewandelt. Waren Benutzer in den Anfängen der Software-Entwicklung in den 1960er Jahren zumeist geschulte Experten, so wird Software heute von einem breitem Personenkreis eingesetzt, der keine spezifische Ausbildung oder Expertise besitzt. *Usability* als Qualität der Benutzung von Software-Systemen hat an Bedeutung gewonnen. Heutige Software-Systeme sollen Benutzer möglichst gut dabei unterstützen, ihre Aufgaben effektiv, mit geringem Aufwand und mit Zufriedenheit zu erfüllen und auf diese Weise ihre Ziele zu erreichen. Das Ideal stellt ein System dar, das seine Benutzer begeistert. Usability erscheint somit als eine maßgebliche Qualität bei der Entwicklung und beim Einsatz von Software-Systemen.

Unzureichende Usability wirkt sich auf unterschiedliche Weise negativ aus. Benutzer des Software-Systems arbeiten weniger effizient, machen häufiger Fehler, nutzen aus Unkenntnis oder Unwillen nicht alle Möglichkeiten des Systems und sind in der Konsequenz unzufrieden oder gar verärgert. Die Produktivität sinkt, die Akzeptanz des Systems auf Benutzer- und Kundenseite schwindet, höhere Kosten für Schulungen und Support sind die Folge. Auf Herstellerseite können Usability-Probleme zu hohen Fehlerbehebungskosten und Zeitverzug im Projekt

führen. Auch in der Wartung, auf die im Softwarelebenszyklus oftmals der größte Kostenanteil entfällt, kann die Behebung von Usability-Problemen erhebliche Aufwände verursachen. Nicht zuletzt leidet bei Entwicklungs- und Auftragsprojekten der Ruf des Herstellers, wenn sich die Usability des entwickelten Software-Systems als unzureichend erweist [Nielsen, 2004]. Gute Usability hingegen führt auf Herstellerseite zu geringeren Kosten für Entwicklung und Wartung der Software und zu positiven Wettbewerbseffekten, auf Kundenseite zu geringeren Dokumentations-, Schulungs- und Supportkosten und zu einer höheren Produktivität der Benutzer [Harrison u. a., 1994; Mayhew u. Mantei, 1994; Donahue, 2001].

Viele Eigenschaften gut benutzbarer Software sind seit langem bekannt und beschrieben. Ein rascher Blick auf manch existierende Software zeigt, dass gute Usability dennoch nicht selbstverständlich ist. Auch in heutigen Software-Projekten entsteht nicht zwangsläufig gut benutzbare Software, wenn dies nicht bewusst und mit Hilfe geeigneter Maßnahmen angestrebt wird. Wie kann nun also zuverlässig Software mit guter Usability entwickelt werden? Im Kontext der Mensch-Rechner-Interaktion wurde hierzu eine Reihe von Ansätzen entwickelt, die die Gestaltung ergonomischer Systeme durch konstruktive, analytische und organisatorische Maßnahmen ermöglichen sollen. Sie werden unter dem Begriff *Usability Engineering* zusammengefasst. Die systematische Berücksichtigung dieser Ansätze in der Praxis wird jedoch dadurch erschwert, dass häufig unklar bleibt, wie Usability-bezogene Aktivitäten (und die resultierenden Artefakte) in existierende Software-Entwicklungsprozesse und etablierte Vorgehensweisen des Software Engineerings eingebunden werden können. Wirkung und Akzeptanz von Usability-Maßnahmen, die inhaltlich oder personell losgelöst von den „klassischen" Entwicklungsschritten in der Software-Entwicklung durchgeführt werden, sind jedoch oftmals gering. Hinzu kommt, dass viele Software-Ingenieure in der Praxis nicht über ausreichende Erfahrung und Expertise verfügen, die für die effektive Anwendung der Ansätze erforderlich sind – und auch nicht über die Zeit, sich in komplexe Regelwerke oder Notationen einzuarbeiten. In der Praxis der Software-Entwicklung sind systematische Ansätze zur Verbesserung der Usability deshalb in vielen Fällen nicht etabliert. Vorhandenes Wissen über die ergonomische Gestaltung von Software-Systemen bleibt so ungenutzt.

Besonders bei frühen Entwicklungsaktivitäten wie der Analyse und Spezifizierung der Anforderungen und der Konzeption der Systemarchitektur werden Usability-Aspekte in Software-Projekten kaum berücksichtigt. Dort richten Software-Ingenieure ihr Augenmerk noch immer vor allem auf die fachliche Funktionalität der zu entwickelnden Systeme. Etablierte Spezifikationsmethoden und -techniken des Requirements Engineerings sehen keine spezifische Berücksich-

tigung von Usability-Anforderungen vor, die über die generische Behandlung als Qualitätsanforderungen hinaus geht. Die unzureichende Betrachtung von Usability bei der Definition der Anforderungen kann jedoch dazu führen, dass gewünschte Usability-Merkmale der Software, die die Qualität der Benutzung positiv beeinflussen können, nicht oder nur unzureichend spezifiziert werden, aus diesem Grund erst deutlich später im Entwicklungsprozess erkannt werden (z. B. bei einem Usability-Test) und mit entsprechend höherem Aufwand nachträglich umgesetzt werden müssen oder letztlich ganz entfallen. Dies resultiert in höheren Kosten für die Entwicklung der Software, schlechter Usability bei der Benutzung – oder beidem.

Aus der beschriebenen Situation ergibt sich die Motivation für diese Arbeit und das Ziel, ein pragmatisches Verfahren zu entwickeln, mit dem Software-Ingenieure auch ohne weitreichende Usability-Expertise wichtige Usability-Aspekte bei der Analyse und Spezifizierung der Anforderungen an die Software besser berücksichtigen können. Der folgende Abschnitt erläutert dieses Ziel näher und skizziert die wesentlichen Ergebnisse dieser Arbeit.

1.2 Ziel, Vorgehen und Ergebnisse

Das Ziel dieser Arbeit lässt sich wie folgt formulieren:

> Ziel ist die Entwicklung und Validierung eines konstruktiven Ansatzes, mit dem unter den Bedingungen heutiger Software-Entwicklung für ein Software-System relevante Usability-Merkmale in frühen Entwicklungsphasen systematisch betrachtet und spezifiziert werden können.

Der Zielsetzung liegen die Annahmen zugrunde, dass spezifizierte Vorgaben die Umsetzung der Usability-Merkmale im Entwicklungsverlauf unterstützen und auf diese Weise zu einer Verbesserung der Usability beitragen. Die explizite Bezugnahme auf die Bedingungen heutiger Software-Entwicklung drückt aus, dass sich der zu entwickelnde Ansatz an den Gegebenheiten in der Praxis und den dort etablierten Vorgehensweisen und vorhandenen Kenntnissen orientieren soll. Ein formaler Ansatz, den Software-Ingenieure in typischen kleinen oder mittelgroßen Software-Projekten nicht einsetzen können, findet erfahrungsgemäß keine Akzeptanz.

Im Folgenden werden das Vorgehen bei der Entwicklung des Ansatzes und die in der Arbeit erzielten Resultate knapp erläutert:

- Zunächst wurden der **Stand der Praxis** und der **Stand der Forschung** betrachtet. Existierende konstruktive Ansätze des Usability Engineerings wurden dazu analysiert und bewertet. Auf Grundlage der kritischen Diskussion und Bewertung der Ansätze ergaben sich **Anforderungen an einen neuen konstruktiven Ansatz**, die sich in drei Stichworten zusammenfassen lassen. Erstens: Konzeptionelle Wiederverwendung bewährter Lösungen. Zweitens: Spezifizierung konstruktiver Vorgaben in frühen Entwicklungsphasen. Und drittens: Methodisches, ingenieurmäßiges Vorgehen.

- Anschließend wurde untersucht, wie Wissen um bewährte und für die jeweiligen Software-Systeme relevante ergonomische Lösungen den Software-Ingenieuren in der Praxis zugänglich gemacht werden kann. Muster wurden dabei als geeignetes Konzept zur Wissensbeschreibung und -vermittlung bewertet und als Grundlage für diese Arbeit ausgewählt; zu der Entscheidung trug auch bei, dass musterbasierte Ansätze in anderen Gebieten des Software Engineerings bereits erfolgreich angewendet werden (z. B. Entwurfsmuster für den Software-Entwurf). Die Entscheidung für einen musterbasierten Ansatz führte zur Entwicklung von **Usability Patterns**. Diese greifen das universelle Musterprinzip auf, das Muster als bewährte Lösungen für Probleme in einem Kontext versteht. In dieser Arbeit wurden als Lösungen speziell *funktionale Usability-Merkmale* betrachtet, also Funktionen der Software, die die Usability verbessern. Diese Merkmale werden in existierenden Ansätzen nur unzureichend berücksichtigt, obwohl ihr positiver Nutzen bekannt ist. Zudem haben sie oft spürbare Auswirkungen auf Software-Architektur und -Implementierung, so dass ein Ansatz zur frühen und systematischen Berücksichtigung auch aus Kostenperspektive relevant ist. Insgesamt 20 bewährte funktionale Lösungsmuster wurden identifiziert und in einem **Katalog von Usability Patterns** in einem einheitlichen Format beschrieben. Der Musterkatalog dient Software-Ingenieuren bei der frühen Auswahl angemessener Usability-Merkmale für ein Software-System als Nachschlagewerk und Diskussionsgrundlage.

- Um geforderte funktionale Usability-Merkmale im Entwicklungsverlauf umzusetzen, müssen entsprechende Vorgaben dokumentiert werden. Als Dokument bietet sich hierfür die Anforderungsspezifikation des Software-Systems an, in der auch die anderen Anforderungen an das System spezifiziert sind. Da existierende Techniken und Notationen die Spezifizierung der Merkmale nicht angemessen unterstützen, wurde eine spezielle

Spezifikationstechnik entwickelt. Diese wurde bewusst so gestaltet, dass sie mit Use Cases eine verbreitete Technik als Grundlage nutzt. Erweiterungen erlauben die Spezifizierung funktionaler Usability-Merkmale in knapper und konsistenter Form. Software-Ingenieure werden beim Einsatz der Spezifikationstechnik durch Spezifikationsschablonen unterstützt.

- Musterkatalog und Spezifikationstechnik alleine implizieren kein methodisches Vorgehen. In einem **Prozessmodell** als Ergänzung eines phasenorientierten Software-Entwicklungsprozesses wurden deshalb anschließend Rollen, Aktivitäten und Artefakte definiert, die die systematische Anwendung von Usability Patterns erlauben. Zentrale Rolle ist dabei der *Usability-Ingenieur*, der bei der Auswahl und Spezifizierung federführend ist. Das Prozessmodell verankert Usability Patterns als Konzept in bewährten Vorgehensweisen der Software-Entwicklung.

- Begleitend entstanden in der Arbeit zwei **Werkzeuge**, die das Vorgehen bei Beschreibung und der Auswahl von Usability Patterns und der Spezifizierung funktionaler Usability-Merkmale erleichtern. Diese wurden auch bei der **Validierung** des Ansatzes in verschiedenen studentischen Software-Projekten eingesetzt. Die erfolgreiche Validierung zeigte, dass der neue Ansatz *Usability Patterns* gute Ergebnisse liefert und somit das eingangs formulierte Ziel der Arbeit erreicht wurde.

Einzelne Zwischenergebnisse und Überlegungen wurden bereits an anderer Stelle vorgestellt [Röder, 2010a,b,c, 2011, 2012]. In dieser Arbeit wird das Konzept *Usability Patterns* zum ersten Mal vollständig beschrieben.

1.3 Aufbau der Arbeit

Die Arbeit ist in vier Teile untergliedert. Die Kapitel 2 und 3 beschreiben Grundlagen der ergonomischen Gestaltung von Software-Systemen und geben einen Überblick über den Stand des Usability Engineerings in der Praxis und in der Forschung.

In den Kapiteln 4 und 5 wird das Konzept der Usability Patterns entwickelt. Zunächst wird in Kapitel 4 erläutert, welche Bedeutung funktionale Merkmale für die Usability haben und inwieweit sich hierbei bewährte Lösungsmuster identifizieren und als Usability Patterns beschreiben lassen. Anschließend werden ein geeignetes Beschreibungsformat und eine Mustersprache für Usability Patterns entworfen.

Kapitel 5 beschreibt dann, wie das Konzept mit Leben gefüllt wird: Entwicklung und Inhalt des in der Arbeit entstandenen Musterkatalogs werden vorgestellt.

Die Kapitel 6 bis 8 widmen sich der praktischen Anwendung von Usability Patterns in der Software-Entwicklung. In Kapitel 6 wird eine Spezifikationstechnik für *erweiterte Use-Case-Spezifikationen* entwickelt, mit der funktionale Usability-Merkmale spezifiziert werden können. Kapitel 7 bettet den Einsatz dieser Spezifikationstechnik und die vorangegangene Auswahl zu spezifizierender Merkmale auf Grundlage des Musterkatalogs in den Software-Entwicklungsprozess ein. Die Entwicklung prototypischer Werkzeuge, die die Anwendung von Usability Patterns und insbesondere die Erstellung erweiterter Use-Case-Spezifikationen unterstützen sollen, wird in Kapitel 8 dokumentiert.

Der abschließende Teil der Arbeit umfasst die Kapitel 9 bis 11. Kapitel 9 beschreibt die Validierung des entwickelten Ansatzes in verschiedenen Software-Projekten. Im folgenden Kapitel 10 wird der Ansatz diskutiert und unter verschiedenen Gesichtspunkten, z. B. Kosten, Nutzen und Grenzen, bewertet. Die Arbeit schließt mit einer Zusammenfassung der erzielten Ergebnisse und einem Ausblick auf mögliche Weiterentwicklungen in Kapitel 11.

2
Grundlagen

Dieses Kapitel behandelt die thematischen Grundlagen der Arbeit und stellt Begriffe vor, die für das Verständnis der nachfolgenden Kapitel notwendig sind. Zunächst wird *Usability* als Begriff und Konzept erläutert. Anschließend werden die Forschungs- und Arbeitsgebiete *Mensch-Rechner-Interaktion* (Abschnitt 2.2) und *Usability Engineering* (Abschnitt 2.3) vorgestellt und abgegrenzt, die sich mit Usability im Kontext der Entwicklung interaktiver Software-Systeme befassen. Die Rolle von Usability im *Software Engineering* und die gegenseitige Integration von Usability Engineering und Software Engineering werden in Abschnitt 2.4 beschrieben. Schließlich geht Abschnitt 2.5 darauf ein, wie Usability-Anforderungen im *Requirements Engineering* berücksichtigt werden.

2.1 Usability

Der Begriff *Usability* (Gebrauchstauglichkeit[1]) ist im Zusammenhang mit der Entwicklung und Nutzung von Software-Systemen vielfach definiert und diskutiert worden. Der Begriff findet sich bereits in frühen Software-Qualitätsmodellen [Boehm u. a., 1976; Cavano u. McCall, 1978]. So betrachten etwa Cavano und McCall in ihrem FCM-Qualitätsmodell (*Factors, Criteria, Metrics*) Usability als einen von elf Qualitätsfaktoren von Software-Systemen und definieren den Begriff als „benötigten Aufwand, um ein Programm zu erlernen, zu bedienen, Eingaben vorzunehmen und Ausgaben zu interpretieren"; je geringer dieser Aufwand ist, desto besser ist die Usability. In der Folge wurde das Verständnis des Begriffs weiterentwickelt. Nach Nielsen ist Usability ein Qualitätsmerkmal, dass sich aus der Erlernbarkeit (*learnability*), der Einprägsamkeit (*memorability*) und der Effizienz der Benutzung (*efficiency*), dem Umgang mit Fehlern (*errors*) sowie der Zufriedenheit der Benutzer (*satisfaction*) zusammensetzt [Nielsen, 1993]. Nielsen stellt Usability als Qualitätsmerkmal neben *Utility* (Zweckmäßigkeit). Utility bezieht

[1] Auf den sperrigen deutschen Begriff *Gebrauchstauglichkeit* wird im Folgenden verzichtet. Der Begriff *Usability* ist auch im deutschen Sprachraum etabliert.

sich dabei auf die Eignung des Software-Systems für die Aufgaben der Benutzer (also auf die Frage, ob das System überhaupt die dafür benötigte Funktionalität bietet). Usability interpretiert Nielsen im Sinne von Benutzerfreundlichkeit (*ease of use*) und Bedienkomfort und betrachtet sie als Produkteigenschaft des Software-Systems. Ein solches Verständnis des Begriffs findet sich auch im Qualitätsmodell der Norm ISO/IEC 9126-1 [ISO9126-1, 2001] wieder.

Dem engen Verständnis von Usability steht eine weiter gefasste Interpretation des Begriffs entgegen, die Usability als umfassende Qualität der Benutzung eines Software-Systems versteht, also als Maß dafür, ob, wie und unter welchen Bedingungen Benutzer mit dem Software-System ihre Ziele erreichen können [Bevan, 1995, 2001]. Benutzer setzen interaktive Software-Systeme ein, um mit ihnen bestimmte Aufgaben zu erfüllen und bestimmte Ziele zu erreichen. Als Usability wird hierbei die Qualität der Benutzung betrachtet, die sich in der Effizienz, Effektivität und Zufriedenheit ausdrückt, mit der Benutzer ihre Aufgaben mit dem Software-System erledigen. Verschiedene Normen definieren den Begriff Usability entsprechend, etwa die Ergonomie-Norm DIN EN ISO 9241-11 [ISO9241-11, 1998] und die Software-Qualitätsnorm ISO/IEC 25010 [ISO25010, 2011] (als Nachfolger der oben zitierten Norm ISO/IEC 9126-1). Auch diese Arbeit verwendet den Begriff Usability in diesem Sinne. In Anlehnung an das in der Norm DIN EN ISO 9241-11 definierte Verständnis der Gebrauchstauglichkeit wird Usability in dieser Arbeit wie folgt verstanden:

Usability — *Usability ist das Ausmaß, in dem ein Software-System durch bestimmte Benutzer in einem bestimmten Nutzungskontext genutzt werden kann, um bestimmte Ziele effektiv, effizient und zufriedenstellend zu erreichen.*

Diese Definition impliziert, dass Usability nicht allein eine Qualitätseigenschaft des Software-Systems, also kein reines Produktmerkmal, ist. Stattdessen bezieht sich Usability als Qualität stets auf die Benutzung des Software-Systems durch Benutzer in einem Kontext und hängt somit von verschiedenen Rahmenbedingungen ab: neben den Eigenschaften des Software-Systems selbst u. a. auch von den Fähigkeiten der Benutzer, den Zielen und Aufgaben sowie dem Nutzungskontext, also der technischen, sozialen und organisatorischen Umgebung, in der das Software-System benutzt wird. Abbildung 2.1 skizziert diesen Zusammenhang.

Bei der Entwicklung von Software-Systemen steht die (voraussichtliche) Usability bei der späteren Benutzung der Systeme naturgemäß in Konkurrenz zu anderen Entwicklungsparametern, etwa Aufwands- oder Terminzielen. Dennoch wird im Allgemeinen, so die Annahme in dieser Arbeit, ein hohes Maß an Usability

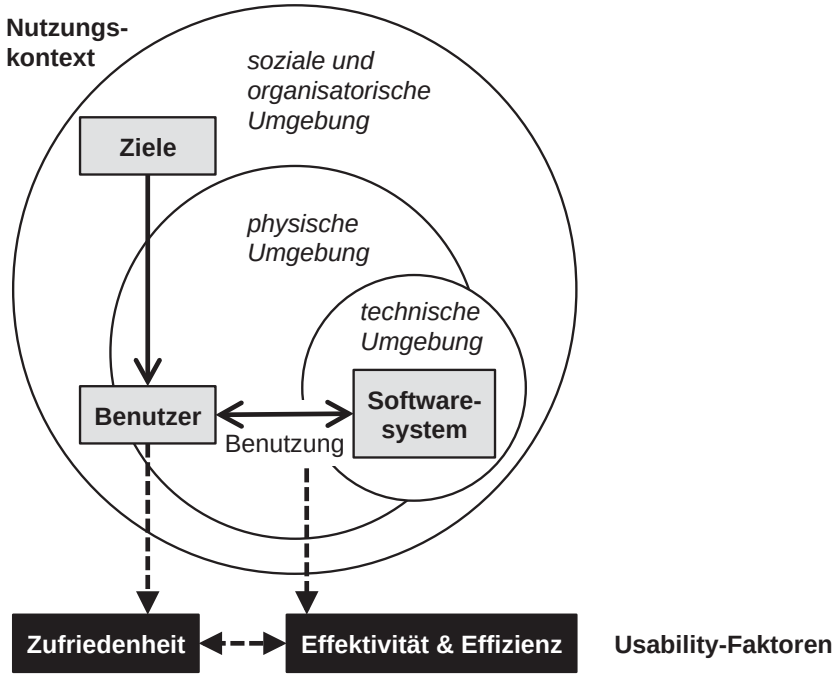

Abbildung 2.1: Usability-Ziele und Nutzungskontext (nach [Bevan, 1995])

angestrebt – Benutzer sollen mit dem System möglichst erfolgreich und fehlerfrei (effektiv), mit geringem Aufwand (effizient) und auf zufriedenstellende Weise arbeiten können. Dahinter steht die Überlegung, dass sich aus hoher Usability eine hohe Produktivität und Motivation der Benutzer sowie geringe Schulungs- und Supportkosten ergeben, hohe Usability letztlich also zu niedrigeren Gesamtkosten führt [Mayhew u. Mantei, 1994].

Um das angestrebte Maß an Usability zu erreichen, können – zumindest theoretisch – alle Rahmenbedingungen variiert und mit Blick auf die Usability optimiert werden. Beispiele sind Schulungen der Benutzer, die deren Fähigkeit zur effektiven und effizienten Nutzung des Systems erhöhen, oder Anpassungen der Geschäftsprozesse (und damit der Aufgaben und Ziele der Benutzer) an die Möglichkeiten des Systems. Software-Ingenieure können jedoch in der Regel nur das Software-System als solches direkt beeinflussen und gestalten. Die Usability-Faktoren Effektivität, Effizienz und Zufriedenheit der Benutzer lassen sich zwar operationalisieren und bei der Systembenutzung messen [Hornbæk, 2006]; in der Entwicklung bleiben sie jedoch zunächst abstrakte Ziele, die sich nicht direkt durch konstruktive Lösungen erfüllen lassen. An ihrer Stelle werden deshalb Eigenschaften des Software-Systems betrachtet, die sich erfahrungsgemäß auf die Qualität der Benutzung (d. h. auf die genannten Usability-Faktoren) auswirken, die zumindest mittelbar bei der Entwicklung konstruktiv beeinflusst werden können und die als Kriterien für die

Bewertung des Systems aus Usability-Sicht herangezogen werden können. In welchem Ausmaß sich Usability-Eigenschaften des Systems auf die Usability-Faktoren auswirken, hängt dabei vom Nutzungskontext ab. DIN EN ISO 9241-110 beschreibt sieben *Grundsätze der Dialoggestaltung*, die als Usability-Eigenschaften sowohl konstruktiv als Gestaltungsprinzipien als auch analytisch als Bewertungskriterien herangezogen werden können: Aufgabenangemessenheit, Selbstbeschreibungsfähigkeit, Erwartungskonformität, Lernförderlichkeit, Steuerbarkeit, Fehlertoleranz und Individualisierbarkeit[2]. Daneben existieren detailliertere Modelle, die eine größere Zahl von Usability-Eigenschaften unterscheiden. So wurden etwa im QUIM-Modell (*Quality in Use Integrated Measurement*) 26 Usability-Eigenschaften (dort als *criteria* bezeichnet) sowie 127 zugeordnete Metriken identifiziert [Seffah u. a., 2006].

Die Usability-Eigenschaften eines Software-Systems werden durch die Benutzungsschnittstelle (*User Interface*, UI) bestimmt. Diese Arbeit lehnt sich beim Verständnis des Begriffs an der Definition der Norm DIN EN ISO 9241-110 an, erweitert diese aber um die explizite Einbeziehung logischer Elemente.

Benutzungsschnittstelle — *Die Benutzungsschnittstelle eines interaktiven Software-Systems besteht aus den logischen und physischen Bestandteilen des Systems, die Informationen und Steuerungsmöglichkeiten zur Verfügung stellen, die für den Benutzer notwendig sind, um eine bestimmte Arbeitsaufgabe mit dem System zu erledigen.*

Die Interaktionsabläufe, die sich für Benutzer bei der Interaktion mit dem System ergeben und die durch eine entsprechende Gestaltung des Software-Systems festgelegt werden, bilden die *logische* Benutzungsschnittstelle. Zur *physischen* Benutzungsschnittstelle zählen in dieser Arbeit im Sinne der Definition insbesondere die wahrnehmbaren Gestaltungsmerkmale grafischer Benutzungsschnittstellen (z. B. Menüs, Fenster, Eingabefelder, Schriftarten, Farben). Zwar sind grundsätzlich auch Hardware-Bestandteile (z. B. die Eingabetastatur des Computers, mit dem ein Benutzer das Software-System benutzt) Teil der physischen Benutzungsschnittstelle; sie werden jedoch in dieser Arbeit nicht betrachtet. Die gedankliche Unterteilung von logischer und physischer Benutzungsschnittstelle kann anhand eines Beispiels verdeutlicht werden: Soll ein System Benutzern die Möglichkeit bieten, Aktionen rückgängig zu machen (*Undo*), so eröffnet dies zusätzliche mögliche Interaktionsabläufe bei der Benutzung des Systems, erfordert also eine entsprechende

[2] Auf diese Gestaltungsgrundsätze wird im Zusammenhang mit der Kategorisierung von Usability Patterns in Abschnitt 5.3.1 nochmals eingegangen.

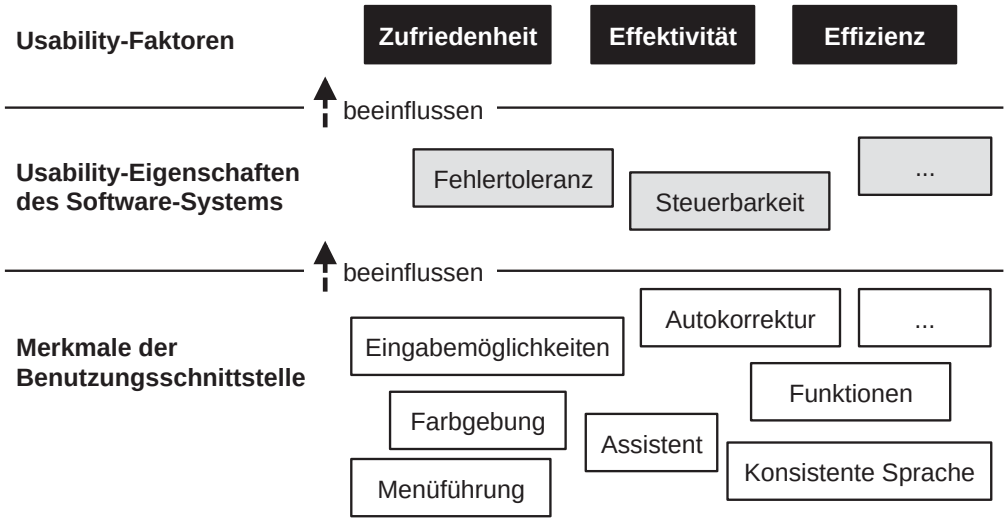

Abbildung 2.2: Usability-Schichtenmodell

Gestaltung der logischen Benutzungsschnittstelle. Gleichzeitig muss die physische Benutzungsschnittstelle so gestaltet werden, dass Benutzer die Undo-Möglichkeit auswählen können; dies kann etwa durch einen zusätzlichen Menüeintrag erfolgen. Die Gestaltung der logischen Benutzungsschnittstelle geht somit zumeist mit der Gestaltung der physischen Benutzungsschnittstelle einher (und umgekehrt).

Die Beziehungen zwischen (logischen und physischen) Bestandteilen der Benutzungsschnittstelle, Usability-Eigenschaften und Usability-Faktoren können als konzeptionelles Schichtenmodell verstanden werden (Abbildung 2.2, in Anlehnung an die schichtenorientierte Sichtweise in [Van Welie u. a., 1999; Folmer u. Bosch, 2004]). In einem solchen Modell bilden die Bestandteile der Benutzungsschnittstelle eines Software-Systems die unterste Schicht. Sie wirken sich in verschiedener Weise auf die darüber angeordneten Usability-Eigenschaften des Systems aus. Die Usability-Eigenschaften des Software-Systems wiederum beeinflussen die Usability-Faktoren, die die oberste Schicht des Modells bilden und sich erst bei der Benutzung des Software-Systems zeigen.

Das vorhandene (oder zumindest vermutete) Wissen um die Zusammenhänge von Benutzungsschnittstelle, Usability-Eigenschaften und Usability-Faktoren ist in Form von Gestaltungsprinzipien, Richtlinien und Heuristiken beschrieben. Es spiegelt sich auch in Methoden und Techniken zur ergonomische Gestaltung von Software-Systemen wider. Diese Ansätze begründen das Gebiet der *Mensch-Rechner-Interaktion*, das im nächsten Abschnitt vorgestellt wird.

2.2 Mensch-Rechner-Interaktion

Als *Mensch-Rechner-Interaktion* (*Human-Computer Interaction*, HCI) wird seit Beginn der 1980er Jahre das interdisziplinäre Arbeitsgebiet bezeichnet, das sich mit dem Verständnis und der Gestaltung der Interaktion zwischen Menschen und Computersystemen befasst [Card u. a., 1983]. Das Gebiet vereint Aspekte der Informatik, der Psychologie (insbesondere der Verhaltens- und Kognitionswissenschaft), der Soziologie, der Arbeitswissenschaft, der Ergonomie und weiterer wissenschaftlicher Disziplinen. Der übergreifende Anspruch wird in der Definition der *Special Interest Group on Computer-Human Interaction* (SIGCHI) der *Association for Computing Machinery* (ACM) deutlich, die Mensch-Rechner-Interaktion als „Fachrichtung, die mit der Gestaltung, Bewertung und Implementierung interaktiver Computersysteme zur menschlichen Benutzung und mit dem Studium der sie umgebenden Vorgänge befasst ist“ beschreibt [Hewett u. a., 1992].

Ziel bei der Anwendung von HCI-Ansätzen ist die Verbesserung der Usability. Der Benutzungsschnittstelle des Software-Systems kommt dabei eine wesentliche Bedeutung zu: eine *ergonomische*, also den Fähigkeiten und Zielen der Benutzer angemessene, einfach bedienbare und leicht erlernbare Benutzungsschnittstelle ist Voraussetzung für gute Usability. Bei der Entwicklung und Gestaltung der Benutzungsschnittstelle können vier prinzipielle Herangehensweisen unterschieden werden [Wallace u. Anderson, 1993; Reiterer, 2001]. Tabelle 2.1 zeigt einen Überblick.

Die handwerkliche Betrachtungsweise (*craft approach*) sieht die individuelle Handwerkskunst und Urteilskraft des UI-Designers als maßgeblich für die Gestaltung an [Laurel, 1990; Wroblewski, 1991]. Nach diesem Verständnis folgt die Qualität der Benutzungsschnittstelle im Wesentlichen aus dem Talent, der Kreativität und der Erfahrung der verantwortlichen Designer, weniger aus den eingesetzten Methoden.

Der kognitionspsychologische Ansatz (*cognitive psychology approach*) stützt sich stärker auf wissenschaftliche Erkenntnisse und wendet Ergebnisse aus der Verhaltens- und Kognitionswissenschaft systematisch bei der Gestaltung der Benutzungsschnittstelle an. Das Benutzerverhalten bei der Interaktion mit Computersystemen und die dabei ablaufenden mentalen Prozesse werden von Experten modelliert, analysiert, bewertet und optimiert [John, 1995; Diaper u. Stanton, 2004].

Der softwareergonomische Ansatz (*enhanced software engineering approach*) versucht, Aspekte der Mensch-Rechner-Interaktion in existierenden Software-Entwicklungsprozessen besser zu berücksichtigen [Nielsen, 1993; Shneiderman, 1998;

Ansatz	Gestaltungsphilosophie, Schwerpunkt	Qualität durch …	Maßgebliche Rollen
Handwerklicher Ansatz	Nutzung der individuellen Fähigkeiten und Erfahrung des UI-Designers (Handwerkskunst)	Talent, Können, Erfahrung	UI-Designer als Künstler und Handwerker
Kognitionspsychologischer Ansatz	Anwendung wissenschaftlicher Theorien über Informationsverarbeitung und Problemlösung, Evaluierung und Modellbildung	Theorie	Psychologe, Ergonom, HCI-Experte
Softwareergonomischer Ansatz (Usability Engineering)	Integration von HCI-Methoden in Software-Entwicklungsprozesse	Methoden	Usability-Ingenieur, Software-Ingenieur
Technologiegetriebener Ansatz	Werkzzeuggestützte, automatisierte Gestaltung von Benutzungsschnittstellen	Werkzeuge	Werkzeugentwickler

Tabelle 2.1: Ansätze zur Gestaltung der Benutzungsschnittstelle (nach [Reiterer, 2001])

Richter u. Flückinger, 2010]. Dazu werden pragmatische Methoden und Techniken bereitgestellt, die z. B. auf kognitionspsychologischen Erkenntnissen beruhen, aber den Anspruch haben, auch von Software-Ingenieuren ohne HCI-Expertise genutzt werden zu können und sich in etablierte Entwicklungsabläufe einbinden zu lassen. Entsprechende Ansätze werden unter dem Schlagwort *Usability Engineering (UE)* zusammengefasst.

Der technologiegetriebene Ansatz (*technologist approach*) schließlich konzentriert sich auf die Entwicklung und Bereitstellung von Software-Werkzeugen für die Gestaltung der Benutzungsschnittstelle [Myers u. a., 2000]. Dabei wird versucht, Gestaltungswissen und -kompetenz in entsprechenden Werkzeugen zu verankern und so die Abhängigkeit der Qualität der Benutzungsschnittstelle vom individuellen Talent der Beteiligten zu vermindern.

Die genannten Ansätze sind nicht unabhängig voneinander, auch sind sie in der Praxis nicht in Reinform anzutreffen. Sie charakterisieren vielmehr unterschiedliche Strömungen innerhalb des Gebiets Mensch-Rechner-Interaktion. Usability Engineering kann dabei am ehesten als Versuch betrachtet werden, eine Brücke zwischen Mensch-Rechner-Interaktion und klassischen, traditionell systemzentrierten Vorgehensweisen der Software-Entwicklung zu schlagen und die Gestaltung ergonomischer Benutzungsschnittstellen unter den Bedingungen heutiger

Entwicklungspraxis zu ermöglichen. Der folgende Abschnitt gibt einen knappen Überblick über das Gebiet.

2.3 Usability Engineering

Als Teilgebiet der Mensch-Rechner-Interaktion befasst sich Usability Engineering mit der zielgerichteten, methodischen Entwicklung ergonomischer Software-Systeme durch Anwendung geeigneter Verfahren. Methoden und Techniken des Usability Engineerings stützen sich auf kognitionspsychologische und arbeitswissenschaftliche Erkenntnisse der Mensch-Rechner-Interaktion und wenden diese praktisch auf verschiedene Aspekte der Software-Entwicklung an. Ansätze sind z. B. Nutzungskontextanalysen (*Usability Context Analysis*) [Thomas u. Bevan, 1996] zur Bestimmung des Nutzungskontexts der Software, Personas [Cooper, 1999] und Szenarien [Carroll, 2000b] zur anschaulichen Modellierung der Systembenutzung[3], UI-Prototyping [Bäumer u. a., 1996] zur explorativen Entwicklung der Benutzungsschnittstelle, heuristische Evaluationen der Benutzungsschnittstelle [Nielsen u. Molich, 1990], kognitive Durchgänge (*Cognitive Walkthroughs*) [Polson u. a., 1992] zur Identifizierung von Usability-Problemen und formale Usability-Tests mit Benutzern zur Bewertung der Usability [Potosnak, 1988].

Als Ergänzung dieser Ansätze wurde eine Reihe von Prozessmodellen des Usability Engineerings entwickelt, die die einzelnen Methoden und Techniken kombinieren und in umfassende Software-Entwicklungsprozesse einordnen. Auf diese Weise soll die durchgängige Berücksichtigung von Usability in allen Phasen der Software-Entwicklung erreicht werden. Nach dem *Usability Engineering Lifecycle* von Mayhew [Mayhew, 1999] etwa beginnt die Entwicklung der Software mit der Anforderungsanalyse, bei der anhand von Nutzungskontext und Benutzerprofilen messbare Usability-Ziele festgelegt werden. Anschließend werden iterativ Methoden des Usability Engineerings, u. a. UI-Prototyping und Usability-Tests, angewandt und Zwischenergebnisse evaluiert. Der Prozess endet, wenn die Usability-Ziele erreicht sind. Weitere bekannte Prozessmodelle sind das namensgebende *Usability Engineering* von Nielsen [Nielsen, 1993], *Star Lifecycle* von Hix und Hartson [Hix u. Hartson, 1993] und *Usage-Centered Design* von Constantine und Lockwood [Constantine u. Lockwood, 1999].

Charakteristische Prinzipien der Prozessmodelle im Usability Engineering sind eine iterative Vorgehensweise bei der Entwicklung, die regelmäßige Evaluation von

[3] Szenarien, UI-Prototypen und weitere konstruktive Ansätze des Usability Engineerings werden in Kapitel 3 näher beschrieben.

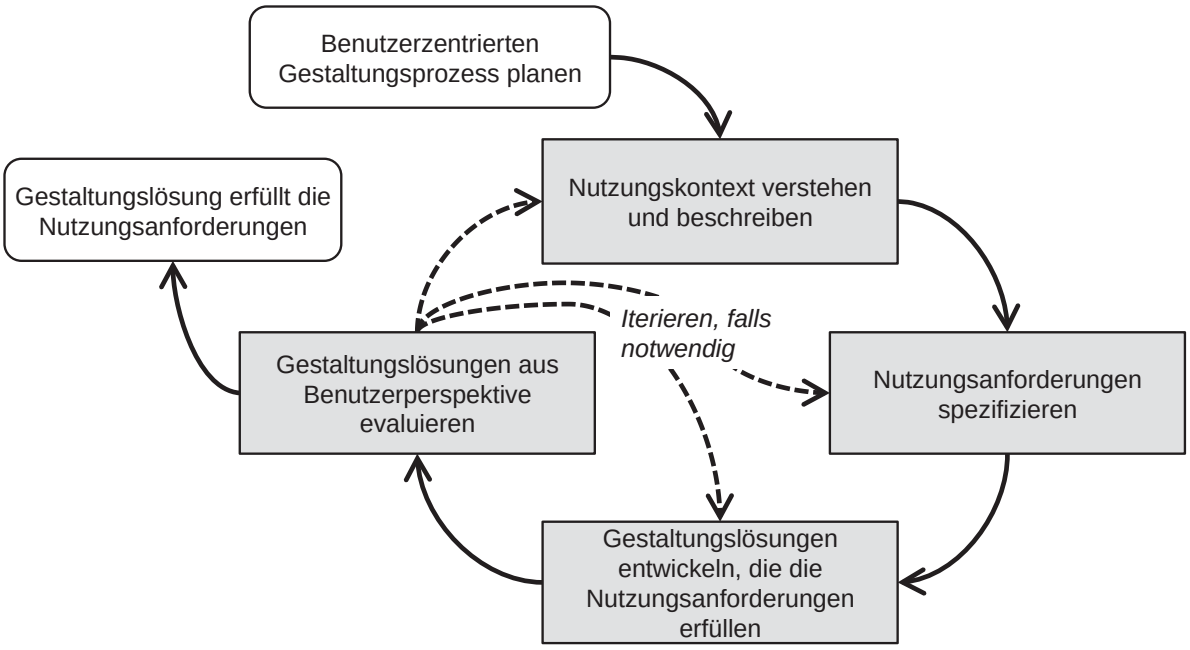

Abbildung 2.3: Iteratives Vorgehen bei der Benutzer-orientierten Gestaltung [ISO9241-210, 2011]

Zwischenprodukten (z. B. Prototypen) sowie die systematische, frühzeitige und intensive Einbeziehung der Benutzer [Gould u. Lewis, 1985]. Die Norm DIN EN ISO 9241-210 greift diese Prinzipien auf und definiert abstrakte Anforderungen an einen „Prozess zur Gestaltung gebrauchstauglicher interaktiver Systeme“ [ISO9241-210, 2011]. Abbildung 2.3 zeigt das prinzipielle Vorgehen, das durch die Norm festgelegt wird.

2.4 Usability im Software Engineering

Software Engineering (SE) beschäftigt sich als Forschungs- und Arbeitsgebiet mit der ingenieurmäßigen Entwicklung von Software-Systemen. Software Engineering umfasst dabei (nach dem weiten und pragmatischen Verständnis von Ludewig und Lichter) „jede Aktivität, bei der es um die Erstellung oder Veränderung von Software geht, soweit mit der Software Ziele verfolgt werden, die über die Software selbst hinausgehen“ [Ludewig u. Lichter, 2010]. Als Forschungsdisziplin reicht das Software Engineering bis in die 1960er Jahre zurück, als es als Reaktion auf unbefriedigende Resultate, Termin- und Kostenüberschreitungen der zunehmend komplexer werdenden Software-Projekte entstand.

Die Sichtweise im Software Engineering ist traditionell systemzentriert, nicht – wie in der Mensch-Rechner-Interaktion und im Usability Engineering – benutzerzentriert. Die technische Konstruktion des Software-Systems steht im Mittelpunkt, nicht die spätere Benutzung des Systems. Zwar wird Usability als Teilqualität der Software betrachtet, die im Qualitätsmodell auf gleicher Stufe neben anderen Teilqualitäten wie Funktionalität, Zuverlässigkeit, Effizienz, Wartbarkeit und Portierbarkeit steht [ISO9126-1, 2001]. Erfahrungsgemäß räumen Software-Ingenieure jedoch der Funktionalität des Software-Systems den weitaus größten Stellenwert bei der Entwicklung ein. In der Konsequenz wird die Verbesserung der Usability nicht als Kernaufgabe des Software Engineerings betrachtet. Charakteristisch ist die Darstellung im SWEBOK (*Software Engineering Body of Knowledge*) der IEEE Computer Society, die die ergonomische Gestaltung der Benutzungsschnittstelle und allgemein Methoden und Techniken zur Verbesserung der Usability von Software nicht dem Software Engineering zurechnet, sondern lediglich unter dem Oberbegriff *Software Ergonomics* als verwandte Disziplin aufführt [Abran u. a., 2004]. Lewis und Rieman bezeichnen eine solche vereinfachende Betrachtungsweise ironisch als „Erdnussbutter-Theorie der Usability“ (benannt nach der Art, wie Erdnussbutter auf ein amerikanisches Pausenbrot gestrichen wird): Die Benutzungsschnittstelle wird in dieser Theorie als konzeptionell losgelöste Schicht angesehen, die unabhängig von anderen Komponenten der Software entwickelt und zum Schluss über diesen ausgebreitet werden kann [Lewis u. Rieman, 1994]. Eine solche Trennung auf architektonischer und zumeist auch organisatorischer Ebene ist jedoch weder möglich noch sinnvoll [Bass u. John, 2001].

Obwohl Software Engineering und Usability Engineering mit der Entwicklung von Software-Systemen befasst sind, besteht zwischen beiden Disziplinen eine Kluft, die sich sowohl in der Forschung als auch in der Praxis auftut. Verschiedene Autoren thematisieren die ungenügende Integration von benutzerzentrierten (UE) und systemzentrierten (SE) Entwicklungsansätzen. Heiskari u. a. kritisieren die ungenügende Berücksichtigung von Usability-Aspekten im Requirements Engineering und führen diese darauf zurück, dass viele Software-Ingenieure die Gestaltung der Benutzungsschnittstelle noch immer als isolierte Aktivität ansehen, die im Rahmen der Anforderungsanalyse nicht betrachtet werden muss [Heiskari u. a., 2009]. Seffah u. a. nennen eine Reihe von Gründen für die unzureichende Verknüpfung der beiden Disziplinen [Seffah u. Metzker, 2004; Seffah u. a., 2005]. Aus ihrer Sicht tragen das noch immer verschiedene Verständnis von Usability als Benutzungsqualität (UE) beziehungsweise als Produktqualität (SE), die Entkopplung von UE-Prozessen von in der Praxis etablierten SE-Entwicklungsprozessen, die geringe Werkzeugunterstützung für UE-Ansätze sowie fehlende Kenntnisse und

Wertschätzung der jeweils anderen Disziplin auf Seiten der Beteiligten maßgeblich dazu bei, dass UE-Ansätze in der Praxis der Software-Entwicklung oftmals nicht eingesetzt werden.

Um die unzureichende Integration zu verbessern, wurden verschiedene Vorschläge zu einer stärkeren Verzahnung von UE- und SE-Aktivitäten im Entwicklungsprozess formuliert. So empfehlen etwa Constantine u. a. die Nutzung einheitlicher, untereinander verknüpfter Modelle, die die beiden Sichtweisen auf das zu entwickelnde Software-System und die spätere Benutzung des Systems aneinander koppeln [Constantine u. a., 2003]. Ferre stellt in einem inkrementellen Vorgehensmodell SE-Aktivitäten in den Mittelpunkt und ordnet diesen die jeweils ergänzend durchzuführenden UE-Aktivitäten zu [Ferre, 2003]. Metzker beschreibt ein Verfahren, mit dem Organisationen kontextspezifisch geeignete UE-Ansätze identifizieren und in den eigenen SE-Prozess integrieren können [Metzker, 2005]. Diese Vorschläge blieben nicht ohne Wirkung. Auch eher traditionell orientierte SE-Prozessmodelle wie der *Rational Unified Process* [RUP, 2003] oder das *V-Modell XT* [V-Modell, 2009] sehen in ihren aktuellen Fassungen explizit Usability-bezogene Prozesselemente (Rollen, Aktivitäten, Artefakte) vor. Hierzu gehören z. B. Rollen wie die des *Ergonomieverantwortlichen* im V-Modell XT oder Aktivitäten wie die Bestimmung der Aufgaben und Ziele der Benutzer, die Entwicklung und Einführung von UI-Styleguides und die Durchführung von Usability-Tests. Im Kontext des Requirements Engineerings erscheint die Berücksichtigung von Usability, insbesondere die Spezifikation entsprechender Vorgaben in geeigneter Form, jedoch trotz dieser Anpassungen noch immer unzureichend.

2.5 Usability-Anforderungen im Requirements Engineering

Wie im Software Engineering spielt Usability auch im *Requirements Engineering (RE)*, das sich als Teildisziplin mit der Bestimmung, Spezifikation und Verwaltung der Anforderungen an Software befasst, traditionell nur eine untergeordnete Rolle. Usability-Anforderungen werden zumeist als nichtfunktionale Anforderungen oder Qualitätsanforderungen klassifiziert [Glinz, 2007]. Sie können prinzipiell in unterschiedlicher Form (quantitativ oder qualitativ, produkt- oder prozessbezogen) formuliert werden [Lauesen u. Younessi, 1998].

Als Usability-Anforderungen im engeren Sinn werden quantitative Anforderungen verstanden, die Zielwerte für Usability-Metriken spezifizieren. Dabei wird versucht, Usability-Faktoren und -Eigenschaften zu operationalisieren und quantitativ messbar zu machen. Ein Beispiel ist die Usability-Metrik *Erlernbarkeit einer*

Funktion (*ease of function learning*), die misst, wie lange Benutzer durchschnittlich benötigen, um die korrekte Verwendung einer Funktion zu erlernen [ISO9126-1, 2001]. Eine quantitative Usability-Anforderung könnte auf Basis dieser Metrik eine maximale Zeitdauer spezifizieren, die Benutzer für das Erlernen einer bestimmten Funktion durchschnittlich benötigen dürfen. Das Beispiel weist bereits auf die grundlegenden Schwierigkeiten hin, die quantitative Usability-Anforderungen aus Sicht vieler Software-Ingenieure mit sich bringen. Zum einen sind angemessene Zielwerte in vielen Fällen nicht bekannt. Zum anderen enthalten die Anforderungen keinerlei Hinweis auf die Konstruktion der Lösung – die Erfüllung der Anforderungen bleibt meist bis zum Abschluss der Entwicklung unklar. Diese Schwierigkeiten mögen ein Grund sein, weshalb quantitative Usability-Anforderungen in der Praxis des Requirements Engineerings kaum formuliert werden [Breitfeld, 2009].

Auch spezifische Vorgaben für die Gestaltung ergonomischer Benutzungsschnittstellen, die über allgemeine Forderungen, z. B. nach Einhaltung bestimmter UI-Styleguides, hinausgehen, werden in existierenden RE-Ansätzen üblicherweise nicht explizit erhoben oder spezifiziert. Dazu mag beitragen, dass keine Spezifikationstechniken oder -notationen etabliert sind, die sich speziell auf Usability-Anforderungen beziehen. Stattdessen finden sich in Anforderungsspezifikationen natürlichsprachige Formulierungen wie die Forderung nach „guter Bedienbarkeit“ und „leichter Erlernbarkeit“, die weder präzise noch überprüfbar sind. In der Konsequenz findet die Software-Entwicklung somit bezüglich Usability zumeist ohne oder nur mit vagen dokumentierten Vorgaben statt [Svensson u. a., 2009].