Universität Hamburg

Fachbereich Informatik

Arbeitsbereich Softwaretechnik (SWT)
Arbeitsbereich Angewandte und Sozialorientierte Informatik (ASI)

Diplomarbeit

Vergegenständlichung von flexiblen Arbeitszusammenhängen in WAM-Anwendungssystemen

Erstbetreuer: Prof. Dr. Heinz Züllighoven (SWT)
Zweitbetreuer: Prof. Dr. Horst Oberquelle (ASI)

Vorgelegt von:
Niels Kausche
Dorotheenstr. 145
22299 Hamburg
Niels@Kausche-online.de
Matrikelnummer: 491 0 430

Erklärung

Diese Arbeit habe ich selbstständig und ausschließlich unter Zuhilfenahme der angegebenen Quellen und Hilfsmittel erstellt.

Hamburg, den 21. März 2005

Anmerkungen

Alle enthaltenen Bezeichnungen und Anreden werden im Sinne funktionaler Rollen (siehe [Floyd 1997]), die menschliche Aktivitäten und Aufgaben bezeichnen, verwendet und sind damit geschlechtsneutral.

In dieser Arbeit werden eingetragene Warenzeichen, Handelsnamen und Gebrauchsnamen verwendet. Auch wenn diese nicht als solche gekennzeichnet sind, gelten die entsprechenden Schutzbestimmungen.

Es liegt die neue deutsche Rechtschreibung zugrunde.

Danksagungen

Für ihre Übernahme der Betreuung dieser Arbeit danke ich Herrn Prof. Dr. Heinz Züllighoven und Herrn Prof. Dr. Horst Oberquelle.

Besonders bedanken möchte ich mich bei Petra Becker-Pechau, die jederzeit ein offenes Ohr für mich hatte.

Katharina Klink, Rocío Millan, Inga Matthiesen und Axel Schmolitzky sei als meinem persönlichen Lektorenteam von tiefstem Herzen gedankt.

Darüber hinaus möchte ich mich bei folgenden Menschen bedanken: Andreas Börnsen, Matthias Eichhorn, Kerstin Heider, Shani Kim, Diana Laugisch, Catrin Peters und Reinhard Wunsch. Jeder von ihnen hat auf seine Weise zum Gelingen dieser Arbeit beigetragen. Danke!

Meinen Eltern

Inhaltsverzeichnis

Kapitel 1: Einleitung

1.1 Motivation

Computer als Arbeitsmittel haben in den letzten Jahren die Gesellschaft durchdrungen. In der Arbeitswelt und auch privat nutzen Menschen, vom Geschäftsführer bis hin zum Studenten, die Möglichkeiten der elektronischen Datenverarbeitung. Es gibt kaum noch Aufgabengebiete, für die keine Software-Lösungen angeboten werden. Das Internet bildet darüber hinaus eine weltweite Infrastruktur für den Austausch von Daten. Daten sind überall und zu jeder Zeit verfügbar. Auch dies trägt dazu bei, dass der Computer immer stärker unser Arbeiten bestimmt.

Betrachtet man die technische Entwicklung aus Anwender-Perspektive, so kann man davon sprechen, dass nahezu ein Plateau erreicht ist. Für die meisten Aufgabengebiete ist die Geschwindigkeit heutiger Computer mehr als ausreichend. „Bürorechner" ist ein Synonym für günstige Rechner geworden. Auch aufseiten der Software ist die Entwicklung langsamer geworden. Vergleicht man heutige Rechner, z.B. unter dem Betriebssystem Microsoft Windows XP und Microsoft Office XP mit einer alten Installation unter Microsoft Windows 95 und ähnlichen alten Office-Version, so gibt es einige Detailverbesserungen und viele optische Verbesserungen, jedoch keine wirklich großen qualitativen Unterschiede. Dies ist bemerkenswert; waren doch die zehn Jahre davor (von 1985-1995) sowohl in Hinsicht auf Hardware (Siegeszug des PCs) als auch Software (Einführung von Microsoft Windows) von starken Veränderungen geprägt. Man könnte sagen, dass die Innovationsphase in eine Phase der stetigen Weiterentwicklung übergegangen ist. Die meisten Innovationen sind eher technischer Art (Flachbildschirme ersetzen Röhrenmonitore) bzw. organisatorischer Art (OpenSource-Bewegung, Digitales Rechtemanagement), die aus Sicht des Anwenders jedoch eher als stetige Entwicklung denn als Innovationen gesehen werden können.

Diese Verlangsamung erscheint für die Anwender von Software aus zwei Gründen vorteilhaft. Einerseits scheint es eine Konsolidierung und damit eine ausreichende Unterstützung in bestimmten Kernbereichen, d.h. in typischen Aufgabenbereichen wie z.B. der Textverarbeitung, zu geben. Andererseits bleibt bei einer langsameren Entwicklung einmal erworbenes Wissen auch längere Zeit gültig. Beides legt den Gedanken nahe, dass nur wenig Verbesserungsbedarf im Hinblick auf eine Computer-unterstützte Arbeitserledigung besteht. Dem ist nicht so.

Häufig gleicht auch noch heute das Arbeiten mit dem Computer eher einem täglichen Ringen als einem souveränen Beherrschen des Arbeitsmittels. Insbesondere als Software-Entwickler vollzieht man häufig einen interessanten Rollentausch. Obwohl man mit seinem technischen Wissen über die Prinzipien von Anwendungen normalen Anwendern voraus ist, kommt es häufig vor, dass man in Anwendungen Funktionalitäten vergeblich sucht, von denen man weiß, dass sie existieren müssen. Oder sie werden nur auf eine bestimmte Art und Weise angeboten, die der gesetzten Aufgabe nicht angemessen ist. Selbst wenn es technisch kein weiter Weg zu einer Lösung wäre, ist man den Entscheidungen seiner Kollegen hilflos ausgeliefert. Das Anpassen einer unbekannten Software –

selbst wenn der Source-Code verfügbar ist (OpenSource) – ist jenseits jeder Verhältnismäßigkeit.

Normale Anwender sind im Prinzip in der gleichen Lage, allerdings wissen sie weit weniger um die Ursachen ihrer Probleme und mögliche Abhilfen. Man kann hier durchaus von einer gewissen Hilflosigkeit und einem „Ausgeliefertsein" sprechen. Dies ist besonders dann bedenklich, wenn der Anwender seine Arbeit mit der Software erledigen muss.

Eine Ursache mag darin liegen, dass sich Software grundlegend von Phänomenen der physischen Welt unterscheidet: Es gibt keine Naturgesetze, keine fundamentalen Regeln, denen eine Anwendung gehorchen muss. Vielmehr stellt jede Software mehr oder weniger eine Welt für sich dar, die der Anwender von Grund auf neu lernen muss. Zwar gibt es Konventionen bei der Benutzung von Anwendungen, allerdings ist ihre Umsetzung freiwillig. Ein Übertragen von Erfahrungen auf andere Anwendungen durch den Anwender ist deshalb nur begrenzt möglich.

1.2 Thema

In dieser Arbeit möchte ich untersuchen, wie ein Anwender Einfluss auf eine Anwendung nehmen kann, damit sie seinen Aufgaben und Bedürfnissen gerecht wird. Dabei möchte ich insbesondere das Spannungsfeld zwischen Anwender und Entwickler berücksichtigen. Entwickler sind dazu ausgebildet, eine Anwendung zu konstruieren und zu gestalten. Allerdings müssen sie dabei konkrete Arbeitssituationen vorwegnehmen. In manchen Arbeitsfeldern können die Aufgabestellungen jedoch stark variieren. Dabei kann es vorkommen, dass die Anwendung für eine spezielle Aufgabe nicht mehr aufgabengerecht ist. Sinnvollerweise sollte der Anwender sein Software-Arbeitsmittel entsprechend umgestalten können. Dies steht allerdings in einem gewissen Konflikt zu den Aufgaben und den Möglichkeiten der Entwickler. Einerseits müssen die Entwickler das korrekte Funktionieren der Anwendung sicherstellen, andererseits stehen auch nur begrenzte Mittel zur Verfügung, um eine mögliche Anpassbarkeit umzusetzen.

Hieraus folgt unmittelbar die Frage, wie dieser Konflikt aufgelöst werden kann. Wie kann eine Anwendung so werden, dass der Anwender sie an geänderte Aufgabenstellungen anpassen kann und die Entwickler zugleich das Funktionieren sicherstellen können? Ein wichtiger Aspekt ist dabei, dass der Anwender die Gestaltungsmöglichkeiten so weit verstehen können muss, dass er Umgestaltungen gezielt planen und durchführen kann. Wie kann dies geeignet unterstützt werden?

Ich möchte mich dabei auf WAM-Anwendungen fokussieren, d.h. Anwendungen, die nach dem WAM-Ansatz entwickelt wurden (siehe [Züllighoven et al. 1998]).

1.3 Überblick

Bevor ich inhaltlich beginne, möchte ich einen kurzen Überblick über die einzelnen Kapitel geben, um dem Leser eine grobe Orientierung zu ermöglichen.

Kapitel 2 wirft einen arbeitswissenschaftlichen Blick auf die Gestaltung von Software. Ich konzentriere mich dabei auf Software zur Unterstützung von *Büroarbeit*. Es wird das Spannungsfeld zwischen den am Software-

Gestaltungsprozess beteiligten Parteien beleuchtet. Neben Anwendern und Entwicklern betrachte ich auch den Arbeitgeber als beteiligte Partei. Anschließend möchte ich auf einen negativen Aspekt der Unterstützung durch Software hinweisen: den Verlust von Gestaltungsspielraum. Diesen Aspekt möchte dann anhand des Kontrollbegriffs näher untersuchen. Eine differenzierte Betrachtung des Kontrollbegriffs führt schließlich zur Forderung nach *Gestaltbarkeit*.

Im Kapitel 3 fasse ich zusammen, wie die aktuelle Software-ergonomische Literatur Gestaltbarkeit diskutiert. Damit eine Anwendung gestaltbar ist, muss sie Spielräume bei der Nutzung anbieten. Der Mechanismus zum Schaffen solcher Spielräume wird in der Literatur mit *Anpassbarkeit* bezeichnet. In diesem Kapitel stelle ich die Motivation für Anpassbarkeit vor und welche Dimensionen der Begriff umfasst, insbesondere welche Gegenstände einer Anwendung typischerweise angepasst werden können. Ich gehe auch auf praktische Hürden beim Einsatz von anpassbaren Systemen ein. Schließlich arbeite ich einen für Gestaltbarkeit wesentlichen Aspekt heraus: *nicht-vorweggenommene Spielräume*.

Gestaltbarkeit setzt Verstehen, genauer *Orientierung,* voraus. Im Kapitel 4 stelle ich deshalb den Begriff der *mentalen Modelle* vor. Mentale Modelle sind ein ganzheitlicher Ansatz, um Verstehen und Orientierung zu erklären. Unter dem Begriff lassen sich viele verschiedene Ansätze zusammenfassen. Letztendlich präsentiert sich so ein Querschnitt durch die aktuelle wissenschaftliche Diskussion über das Verstehen von Sachverhalten allgemein und das Verstehen von Anwendungssystemen im Speziellen. Vor diesem Hintergrund stelle ich vier Instrumente vor, um den Aufbau mentaler Modelle zu unterstützen: *konzeptionelle Modelle*, *Metaphern*, *Konsistenz* und *aktives Lernen.* Zum Abschluss des Kapitels beschreibe ich unterschiedliche Perspektiven auf Software und ziehe eine Verbindung zwischen existierenden Modellen über Software und den vorgestellten Instrumenten.

Das Kapitel 5 stellt meine Lösungsidee zum Konstruieren von gestaltbarer Software vor. Kern der Idee ist ein neues Instrument: *Konzepte*. Sie vergegenständlichen allgemeine Prinzipien einer Anwendung für den Anwender. Dabei können sie sowohl *Orientierung* als auch *Gestaltbarkeit* unterstützen.

Nachdem ich den Begriff *Konzept* eingeführt habe, stelle ich im Kapitel 6 verschiedene Konzepte von WAM-Anwendungen vor. Den Konzepten liegen größtenteils bekannte Konzeptions- und Entwurfsmuster aus dem WAM-Ansatz zugrunde (siehe [Züllighoven et al. 1998]). Eine herausragende Bedeutung kommt dabei dem Konzept des *Arbeitszusammenhangs* zu. Er repräsentiert eine konkrete Arbeitssituation und koordiniert die dafür benötigten Arbeitsmittel. Damit ist er aber auch ein wesentliches Konzept für Anpassungen an andere Arbeitssituationen.

Im Kapitel 7 behandelt die praktische Umsetzung von Anpassungen im Arbeitszusammenhang.

Kapitel 2: Software als Arbeitsmittel

Software als Arbeitsmittel zu bezeichnen, ist sowohl für die Software-Ergonomie als auch die Softwaretechnik nicht neu ([Hacker 1994]). Software zu entwickeln heißt, mit dieser Sichtweise, Arbeit zu gestalten ([Ulich 2001], [Oberquelle 1997]). Mir ist diese Sichtweise deshalb so wichtig, weil sie die Verantwortung deutlich macht, die Software-Entwickler gegenüber den Anwendern ihrer Software tragen. Arbeit nimmt einen großen Teil unserer Lebenszeit in Anspruch und hat einen bedeutenden Einfluss auf die Lebensqualität. Entsprechend verantwortlich muss Software-Entwicklung gestaltet werden.

In diesem Kapitel möchte ich deutlich machen, dass Software-unterstützte Arbeit gegenüber Arbeit mit rein materiellen Arbeitsmitteln nicht nur eine Verbesserung darstellt, sondern auch immer eine Verschlechterung bestimmter Arbeitsbedingungen mit sich bringt. Software ist in vielerlei Hinsicht unverständlich und schränkt den Anwender in seinem persönlichen Gestaltungsspielraum bei der Ausführung der Arbeit ein.

Um mich diesem Themenkomplex zu nähern, möchte ich zunächst ein Bewertungsschema für Arbeit vorstellen. Danach führe ich den Begriff *Büroarbeit* ein. Auf diesen speziellen Typ von Arbeit möchte ich mich in dieser Diplomarbeit konzentrieren. Schließlich möchte ich die Konsequenzen für den Arbeitnehmer herausarbeiten, wenn seine Arbeit durch Software unterstützt wird. Das Einführen des arbeitspsychologischen Begriffs der Kontrolle ermöglicht es, all diese Konsequenzen in einem Wort zusammenzufassen: Kontrollverlust. Möglichkeiten herauszuarbeiten, diesem Kontrollverlust entgegenzuwirken, ist das Ziel dieser Arbeit. Am Ende dieses Kapitels stelle ich deshalb einen ausdifferenzierten Kontrollbegriff vor, der hierfür als Ausgangspunkt dienen soll.

2.1 Ein Bewertungsschema für Arbeitsgestaltung

Etwas zu gestalten bedeutet entweder, etwas *um*zugestalten mit dem Ziel, den Gegenstand in einer bestimmten Hinsicht zu verbessern, oder neu zu gestalten, wobei i.d.R. ein bestimmtes Güte-Maß erreicht werden soll. Gestalten heißt somit immer auch bewerten und erfordert entsprechende Bewertungskriterien. In diesem Abschnitt möchte ich deshalb Bewertungskriterien für Arbeit vorstellen.

In [Luczak et al. 1989] haben die Autoren ein Bewertungsschema für Arbeitsbedingungen aufgestellt, das inzwischen allgemeine Anerkennung findet (siehe auch [Arbeitshandbuch 1997]). Die einzelnen Kriterien sind der Nummerierung folgend priorisiert. Zunächst ist die Schädigungslosigkeit der Arbeitsbedingungen sicherzustellen, darauf folgend die Ausführbarkeit und so fort.

1. Aushaltbarkeit, Erträglichkeit und Schädigungslosigkeit
2. Ausführbarkeit
3. Beeinträchtigungsfreiheit und Handlungsspielraum
4. Arbeitszufriedenheit und Persönlichkeitsentfaltung
5. Sozialverträglichkeit

Schädigungslosigkeit und Erträglichkeit

Der arbeitende Mensch ist in seiner Arbeitswelt häufig künstlichen Bedingungen ausgesetzt, die weit von den natürlichen Bedingungen abweichen können. Es ist wichtig, dass diese Bedingungen innerhalb der Beanspruchungskompensation des Organismus Mensch liegen. Dabei wird zeitlich nach kurzfristiger Beanspruchung (Aushaltbarkeit), mittelfristiger Beanspruchung im Stunden- und Tagesbereich (Erträglichkeit) als auch langfristiger Beanspruchung im Bereich von Monaten und Jahren (Schädigungslosigkeit) unterschieden.

Ausführbarkeit

Zur Sicherung der Ausführbarkeit ist es erforderlich, dass sich die Anforderungen innerhalb der Grenzwerte menschlicher Leistungsfähigkeit bewegen, etwa hinsichtlich der Erreichbarkeit von Bedienteilen, erforderlicher Körperkräfte, Erkennbarkeit von Anzeigen und Wahrnehmbarkeit von Signalen.

Beeinträchtigungsfreiheit und Handlungsspielraum

Zu achten ist auch auf die Einhaltung kollektiver Normen (z.B. gesetzlicher und tarifvertraglicher Art). Neben diesen soziokulturellen Aspekten muss der Tätigkeitsumfang individuell auf Basis handlungstheoretischer Begründung bewertet werden, beispielsweise ob der Grad der Arbeitsteilung den Handlungsspielraum einer Person beeinträchtigt.

Arbeitszufriedenheit und Persönlichkeitsentfaltung

Die Begriffe der Zufriedenheit und Persönlichkeitsentfaltung schließlich heben stärker auf eine Bewertung der Arbeitssituation ab, die nicht in Richtung auf das Erreichen von Mindestbedingungen ausgelegt sind. Hier wird Bezug auf die individuelle Arbeitsperson genommen. Einen objektiv beschreibbaren Ziel-Gestaltungszustand von Arbeit kann es deshalb für dieses Kriterium nicht geben.

Sozialverträglichkeit

Sozialverträglichkeit meint die Partizipation in der Gestaltung von Arbeitsbedingungen, z.B. Gruppenarbeitsansätze. Als mündige Arbeitsperson sollte der Arbeitnehmer durch Mitwirkung und Mitsprache die Arbeitswelt, genau wie die persönliche Lebenswelt ohnehin, nach seinen Bedürfnissen und Interessen mitgestalten können.

Dieser Kriterienkatalog erlaubt nun eine Bewertung eines Arbeitsgestaltungs-prozesses. Bevor ich jedoch auf einen Arbeitsgestaltungsprozess eingehe, möchte ich näher auf die Klasse der von mir betrachteten Arbeit eingehen: die Büroarbeit.

2.2 Büroarbeit - eine besondere Form von Arbeit

In dieser Diplomarbeit möchte ich mich auf die Betrachtung von *Büroarbeit* beschränken. Aufgrund ihrer besonderen Merkmale ist eine geeignete Unterstützung durch Software besonders schwierig. Der Begriff *Büroarbeit* ist durchaus intuitiv zugänglich. Es fällt nicht schwer, ihn mit konkreten Vorstellungen zu verbinden. Die dieser Arbeit zugrunde liegende Bedeutung des Begriffs

Büroarbeit möchte ich deshalb an dieser Stelle anhand von Merkmalen konkretisieren. Dazu greife ich auf Merkmale zurück, die in [Friedrich 1995] aufgeführt sind.

Allen Tätigkeiten im Büro ist gemeinsam, dass es sich im Wesentlichen um *geistige Arbeit* handelt. Es gibt dabei eine „natürliche" Nähe zur Informatik, die sich u.a. darin begründet, dass die Gegenstände und Produkte dieser Arbeitsform „Informationen" sind.

Ein weiteres Merkmal von Büroarbeit ist ein breites Spektrum von Tätigkeiten *unterschiedlicher Formalisierbarkeit*. An einem Ende des Spektrums befinden sich Routinetätigkeiten, zu deren Lösung klare Regeln existieren und die der „informationellen Massenverarbeitung" zugänglich sind. Am anderen Ende des Spektrums befinden sich Einzelfälle, die häufig nur vage zu beschreiben sind.

Darüber hinaus ist Büroarbeit häufig arbeitsteilig organisiert. Hier ist eine Form von *Koordination* bzw. *Kooperation* mit den Kollegen erforderlich.

Über die von Jürgen Friedrich genannten Merkmale hinaus, möchte ich noch weitere Aspekte benennen. Zum einen verfügen Mitarbeiter im Rahmen von Büroarbeit über ein gewisses Maß an Zeitautonomie. Zeitliche Vorgaben werden, wenn überhaupt, nur individuell oder für eine konkrete Aufgabe gesetzt. Eine explizite zeitliche Taktung, wie z.B. bei der Arbeit am Fließband, gibt es nicht. Diesen Aspekt möchte ich als *zeitliche Teilautonomie* bezeichnen.

Die zeitliche Teilautonomie ist die Voraussetzung für das nächste Merkmal, den *Gestaltungsfreiräumen in der Ausführung*. Eine zeitliche Teilautonomie ermöglicht es dem Mitarbeiter, Einfluss auf seine Arbeitserledigung zu nehmen. Er hat dann z.B. die Zeit für arbeitsvorbereitende Tätigkeiten. Die eigentliche Ausführung der Arbeit ist damit durch ihn selbst gestaltbar. Man spricht auch von Handlungsspielräumen ([Ulich 2001]). Zwar gibt es in vielen Fällen Regelungen, die bei der Arbeitserledigung beachtet werden müssen, aber bei der konkreten Ausführung, der konkreten Überführung der Eingabe zum Arbeitsergebnis (vergleiche auch Abbildung 2 auf Seite 12), hat jeder Mitarbeiter Gestaltungsfreiräume, die er individuell nutzen kann. Man könnte auch sagen, der *Inhalt* der Arbeit wird durch den Arbeitgeber bestimmt, die *Form* kann durch den Mitarbeiter frei gestaltet werden.

Darüber hinaus gibt es zwei Merkmale, die für die Software-Entwicklung besonders wichtig sind und die ich abschließend aufgreifen möchte: *Stabilität* und *Häufigkeit* von Tätigkeiten.

Typisch für Büroarbeit ist es, dass sich Anforderungen ändern, wie z.B. interne Regelungen oder zu berücksichtigende Gesetze. Infolge dessen werden neue Tätigkeiten Teil der Arbeit, bestehende Tätigkeiten ändern sich oder entfallen sogar ganz. Auch hier gibt es ein breites Spektrum. Es reicht von Tätigkeiten, die lange Zeit stabil bleiben, bis hin zu Tätigkeiten, die sich beinahe täglich ändern. Diesen Aspekt möchte ich als *Stabilität* von Tätigkeiten bezeichnen.

Daneben gibt es ein breites Spektrum im Hinblick auf die *Häufigkeit* bestimmter Tätigkeiten. Manche Tätigkeiten fallen mehrmals am Tag an, andere nur einmal im Jahr oder sogar, in einer ganz besonderen Ausprägung, nur ein einziges Mal.

Büroarbeit ist einer der zentralen Begriffe dieser Arbeit. An dieser Stelle fasse ich deshalb die eben genannten Kriterien zusammen.

Büroarbeit zeichnet sich aus durch:

- überwiegend geistige Arbeit
- Informationen als zentrale Gegenstände und Produkte
- Tätigkeiten unterschiedlicher Formalisierbarkeit
- Koordination und Kooperation mit Kollegen
- zeitliche Teilautonomie
- Gestaltungsfreiräume in der Ausführung
- Tätigkeiten unterschiedlicher Stabilität
- Tätigkeiten unterschiedlicher Häufigkeit

Mit diesem Merkmalskatalog als Definitionsgrundlage hebt sich Büroarbeit von anderen Arbeitsklassen, wie z.B. Fließbandarbeit, dadurch ab, dass ihre Inhalte und damit die Arbeitsprozesse stark variieren können. Büroarbeit ist deshalb ein Bereich, in dem der Einsatz von Rechnern in Form von Automatisierung i.d.R. weder möglich noch sinnvoll wäre. Vielmehr zeichnet sich Büroarbeit durch ein hohes Maß an notwendiger Flexibilität und situativem Handeln aufseiten der Mitarbeiter aus. Gestaltungsfreiräume in der Ausführung bieten den dafür notwendigen Raum.

Mit Bezug auf den Arbeitsgegenstand „Informationen" könnte man Büroarbeit als das informationsbezogene Spiegelbild des materiell bezogenen Handwerks bezeichnen. Diese Analogie liegt z.B. dem WAM-Ansatz (**W**erkzeug-, **A**utomat-, **M**aterial-Metapher) zugrunde (siehe [Züllighoven et al. 1998]).

2.3 Gestaltung von Software-unterstützter Büroarbeit

Nachdem ich im vorhergehenden Abschnitt den Begriff „Büroarbeit" anhand von Merkmalen genauer definiert habe, möchte ich in diesem Abschnitt auf die Eigenschaften der Gestaltung von Software-unterstützter Büroarbeit und ihren Konsequenzen für die betroffenen Mitarbeiter eingehen.

Zunächst einmal schränken die besonderen Merkmale von Büroarbeit die Klasse der sinnvollerweise einsetzbaren Software ein. Diese Software-Klasse möchte ich in diesem Abschnitt skizzieren. Am eigentlichen Gestaltungsprozess der Software-Entwicklung sind viele verschiedene Parteien beteiligt. Um die damit verbundene Dynamik besser verstehen zu können, ist es sinnvoll, die Akteure und ihre Ziele näher zu betrachten. Tendenziell ist der Mitarbeiter dabei der schwächste Akteur. Im Folgenden möchte ich auf diese Punkte genauer eingehen.

2.3.1 Software für Büroarbeit

Der Zweck „Unterstützung von Büroarbeit" setzt bereits bestimmte Rahmenbedingungen für eine Software. Man könnte von einer Software-Klasse sprechen. In diesem Abschnitt möchte ich diese Software-Klasse herausarbeiten, auf die ich mich dann im Weiteren beziehe.

Büroarbeit zeichnet sich, wie im Abschnitt 2.2 beschrieben, durch ein hohes Maß an notwendiger Flexibilität und situativem Handeln aus. Deshalb ist der Ansatz der Automatisierung für die Unterstützung von Büroarbeit nur in Teilbereichen sinnvoll. Viel bedeutender für die Gestaltung von Software-unterstützter Büroarbeit ist *interaktive Software*. Bei ihr findet die Arbeitserledigung in stetigem Wechselspiel zwischen Anwender und Software statt.

Bei interaktiver Software sind zwei Sichtweisen zu unterscheiden: die ablaufsteuernde und die unterstützende Sichtweise (siehe [Gryczan 1995]). Die ablaufsteuernde Sichtweise basiert auf Automatisierung. Sie versucht den Menschen in die Automatisierung zu integrieren, indem sie menschliches Arbeitshandeln regelt und kontrolliert. Die unterstützende Sichtweise hingegen sieht den Anwender als Experten seines Arbeitsgebietes, der in seinem Arbeitshandeln Computer als Arbeitsmittel einsetzt. Wie Guido Gryczan feststellt, wird die letztere Sichtweise den Anforderungen menschlichen Handelns (und damit auch denen von Büroarbeit) eher gerecht.

Ein weiterer, die Software charakterisierender Aspekt begründet sich im Merkmal *Informationen als Gegenstand* von Büroarbeit. Im Gegensatz zur Steuerung von Maschinen sind bei der Modellierung des Problembereichs kaum technischen Aspekte zu berücksichtigen. Wichtig ist dabei „lediglich" eine geeignete Modellierung der zentralen Arbeits-Konzepte und Gegenstände, sodass dem Anwender die Aufgabenerledigung ermöglicht wird. Hier hat der Software-Entwickler einen großen Gestaltungsspielraum. Die Frage, was eine geeignete Modellierung ausmacht, ist alles andere als trivial und wird im Verlauf der Arbeit noch aufgegriffen werden.

Unter den formalisierbaren Tätigkeiten im Bereich Büroarbeit gibt es solche, die Organisations-übergreifend formalisiert werden können, und solche, die nur Organisations-individuell formalisiert werden können. Dies führt zu mindestens zwei Kategorien von Software innerhalb einer Organisation: Software-Produkte, die auch in anderen Organisationen eingesetzt werden, und Individual-Software, die speziell für diese Organisation (oder wenige andere) erstellt wurde[1]. Software-Produkte sind dabei wesentlich günstiger zu erhalten als Individual-Software, da die Kosten von mehreren Kunden-Organisationen getragen werden. Für die Entwicklung von Software-Produkten stehen aus dem gleichen Grund wesentlich höhere Geldbeträge zur Verfügung als für Individual-Software. Gegenüber Software-Produkten hat die Entwicklung von Individual-Software allerdings den Vorteil, dass für eine kleinere Anwendergruppe entwickelt wird. Die Aufgaben und Arbeitsplätze lassen sich deshalb spezifischer untersuchen und unterstützen.

Im Prinzip ist bei der Entwicklung von Software aus beiden Kategorien Verständlichkeit und Flexibilität eine Herausforderung, wenn auch aus verschiedenen Gründen. Bei Software-Produkten durch die größere Distanz zum konkreten Arbeitsplatz, bei Individual-Software durch den größeren Aufwand und die dadurch wesentlich höheren Kosten. Die in dieser Arbeit aufgestellten Thesen lassen sich prinzipiell auf beide Kategorien anwenden. Der Einfachheit halber möchte ich mich im Weiteren jedoch auf Individual-Software konzentrieren.

Ich fasse zusammen: Bei Software für die Unterstützung von Büroarbeit handelt

1 System-Software wie z.B. SAP, die für die Kunden-Organisation individualisiert werden kann, liegt im Prinzip zwischen Software-Produkt und Individual-Software. Ich möchte diese Art von Software im Weiteren nicht betrachten.

es sich um unterstützend, interaktive Software, deren zentralen Konzepte unabhängig von technischen Aspekten modelliert werden können.

Darüber hinaus möchte ich diese Arbeit speziell Systeme betrachten, die nach dem WAM-Ansatz entwickelt wurden (siehe [Züllighoven et al. 1998]). Der WAM-Ansatz nutzt zur Implementierung Methoden der objektorientierten Programmierung. Zur Unterstützung der Konstruktion nach WAM wurde das objektorientierte Rahmenwerk JWAM ([JWAM 2005]) entwickelt, auf das ich in dieser Arbeit nicht explizit eingehen, aber an einigen Stellen verweisen werde.

2.3.2 Die beteiligten Parteien und ihre Ziele

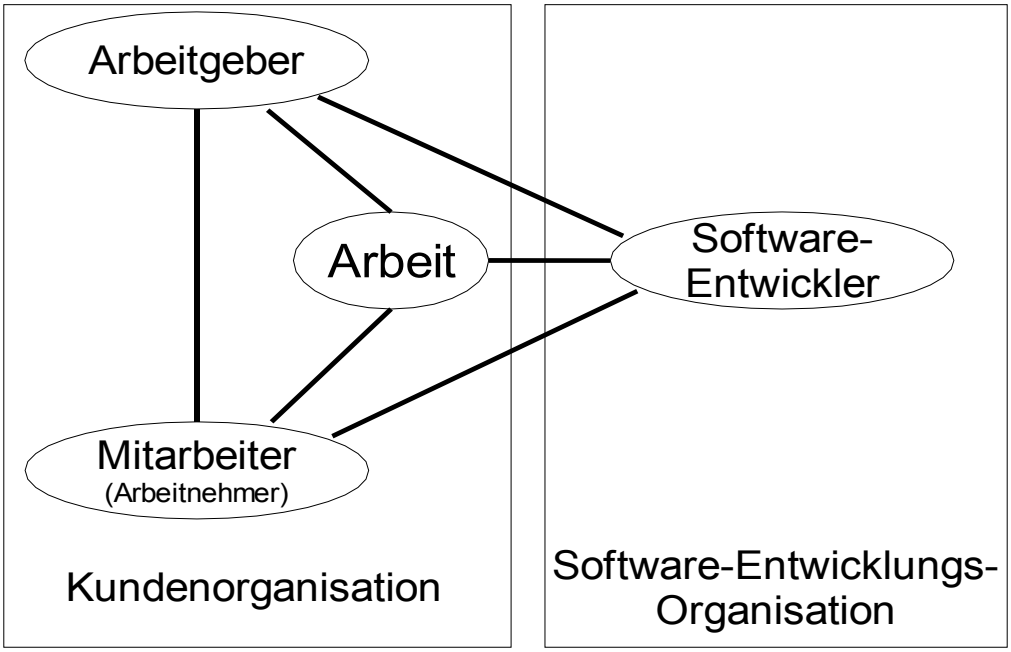

Abbildung 1: Die bei der Software-Entwicklung beteiligten Parteien gestalten Arbeit in der Kundenorganisation.

Außer von den Bewertungskriterien ist ein Gestaltungsprozess von den beteiligten Parteien und ihren Zielen abhängig. Ich möchte in diesem Abschnitt auf die an Software-Entwicklung von Individual-Software beteiligten Parteien und ihre Ziele eingehen. Abbildung 1 zeigt eine Übersicht der in diesem Abschnitt beschriebenen Zusammenhänge.

Der Arbeitgeber[2]

Diese Partei nimmt zwei verschiedene Rollen im Gestaltungsprozess wahr. Zum einen ist sie Auftraggeber gegenüber der Organisation, die mit der Software-Entwicklung betraut wird. Zum anderen sind ihre Mitglieder häufig direkte Vorgesetzte der betroffenen Mitarbeiter. Der Arbeitgeber beurteilt den Gestaltungsprozess und vertritt als Entscheidungsträger die Interessen der Kundenorganisation als Auftraggeber gegenüber der Software-Entwicklungs-Organisation als Auftragnehmer.

2 Ich verwende an dieser Stelle den Singular, weil diese Gruppe oft aus nur einer Person bzw., im Vergleich mit den anderen beteiligten Akteuren, aus wenigen Personen besteht.

Der Arbeitgeber hat i.d.R. den Gestaltungsprozess angestoßen. Dafür sind meist betriebswirtschaftliche Gründe ausschlaggebend. Arbeitsprozesse sollen produktiver und damit kostengünstiger gestaltet werden. Oder aber es sollen neue Arbeitsprozesse ermöglicht werden, um den eigenen Kunden bessere Leistungen anbieten zu können.

Wesentlich bei den Zielen des Arbeitgebers ist der Fokus auf die Gesamtwirkung des Umgestaltungsprozesses. Aufgaben dürfen z.B. durchaus komplizierter gestaltet werden, wenn letztendlich die Gesamtkosten sinken, z.B. durch das Wegfallen anderer Aufgaben.

Um die betriebswirtschaftlichen Ziele zu erreichen, müssen die betroffenen Mitarbeiter nach dem Umgestaltungsprozess die Funktionalität der Software auch nutzen können. Dafür müssen die Mitarbeiter wissen, wie sie ihre Aufgaben überhaupt mit der entwickelten Software bearbeiten können und wie sie die Möglichkeiten der Software auf eine möglichst effektive Art und Weise nutzen können. Darüber hinaus kann sich die Arbeitsgestaltung auch auf die Motivation auswirken, die sich über den Faktor des persönlichen Einsatzes ebenfalls auf die Produktivität auswirken kann.

Aus Sicht des Arbeitgebers ist allerdings häufig die reine Funktionalität des Anwendungssystems ausschlaggebend. Meist ist er vom Umgestaltungsprozess nicht direkt betroffen, also kein Mitarbeiter. Außerdem ist in dem an sich recht abstrakten Software-Entwicklungsprozess „Funktionalität" ein konkreter Begriff, dessen Kosten und Nutzen sich relativ leicht nachvollziehen lassen.

Die Software-Entwickler

Die Ziele der Software-Entwickler lassen sich relativ einfach zusammenfassen. Sie wollen den Kunden (den Arbeitgeber) mit ihrer Dienstleistung zufrieden stellen. Dabei müssen sie die betriebswirtschaftliche Ziele der eigenen Organisation beachten. Sie müssen ihre Dienstleistung günstig anbieten, um sich gegen Konkurrenten bei der Auftragsvergabe durchsetzen zu können, und sie möchten dabei einen möglichst großen Gewinn erwirtschaften. Diese Ziele ändern sich nicht wesentlich, auch wenn die Software-Entwicklungs-Organisation in der Kundenorganisation integriert ist, z.B. als EDV-Abteilung.

Dabei ist zu beachten, dass die Planung eines Software-Entwicklungs-Projekts besonders komplex ist. Die tatsächlich entstehenden Kosten sind am Anfang, bei der Angebotserstellung, nur schwer zu schätzen. Gleichzeitig muss das Angebot mit denen anderer Software-Enwicklungs-Organisationen konkurrieren. Dies führt in der Praxis häufig dazu, dass im späteren Projektverlauf entweder das Budget erhöht oder auf Funktionalität verzichtet werden muss. Letzteres ist ein Kerninstrument agiler Methoden der Software-Entwicklung, die aktuell gegenüber nicht-agilen Methoden eine bessere Planbarkeit von Software-Entwicklungs-Projekten versprechen (siehe auch [Beck und Andres 2004]).

Die Software-Entwickler stehen also tendenziell unter dem Druck, geforderte Funktionalität schnell und kostengünstig umzusetzen.

Die Mitarbeiter

Diese Partei umfasst alle Mitarbeiter der Kundenorganisation, die durch den Umgestaltungsprozess unmittelbar in ihrer Arbeit betroffen sind.

Widersinnigerweise bilden sie tendenziell die schwächste Partei. Widersinnig deshalb, weil ihnen später die Aufgabe zufällt, die Nutzen-Seite der Kosten-Nutzen-Rechnung des Umgestaltungsprozesses zu realisieren. Selbst wenn durch die Umgestaltung Arbeitsplätze rationalisiert werden, muss der Arbeitsprozess durch die verbleibenden Mitarbeiter in Interaktion mit der Software getragen werden. Ihre Schwäche begründet sich dadurch, dass sie auf die Kosten-Seite, die den Prozess im Wesentlichen prägt, keinen Einfluss haben. Sowohl das zur Verfügung stehende Budget als auch die entstehenden Kosten entziehen sich ihrem Einflussbereich.

Zum Arbeitgeber besteht ein Arbeitsverhältnis. Die Mitarbeiter stellen dem Arbeitgeber im Tausch für ein Gehalt ein vereinbartes Kontingent ihrer Zeit zur Verfügung, die Arbeitszeit. Elementares Ziel des Mitarbeiters ist es dabei, diese fest vereinbarte Zeit für ihn möglichst angenehm zu gestalten. Arbeit ist ein Mittel der Persönlichkeitsentfaltung. Es darf daher unterstellt werden, dass „angenehm" nicht mit Untätigkeit gleichzusetzen ist, sondern mit einem möglichst produktivem Arbeiten bei einer nicht als ungangenehm empfundenen Belastung.

Mit dieser Sichtweise haben Arbeitgeber und Arbeitnehmer durchaus die gleichen Ziele. Der Unterschied findet sich jedoch im jeweiligen Fokus. Der Arbeitgeber akzeptiert Arbeitsbedingungen, die vom individuellen Mitarbeiter als unproduktiv und unangenehm empfunden werden, solange die Kosten-Nutzen-Rechnung auf Organisationsebene die gewünschten Produktivitätzuwachs erzielt. Dabei muss dem Arbeitgeber keine böse Absicht unterstellt werden. Aus Sicht der Kundenorganisation ist es sogar seine Pflicht, auf das Erreichen des Kosten-Nutzen-Verhältnisses zu achten, um langfristig das betriebswirtschaftliche Überleben der Organisation zu sichern.

Damit sind die Ziele und Interessen der am Software-Entwicklungs-Prozess beteiligten Parteien erläutert. Wesentlich dabei ist das Spannungsfeld um das Kosten-Nutzen-Verhältnis, das den Arbeitgeber umgibt. Bei der Realisierung von Funktionalität ist der Nutzen sofort erkennbar und der Arbeitgeber kann die entstehende Kosten gegen den Nutzen abwägen. Alle weiteren Tätigkeiten im Software-Entwicklungs-Prozess, wie z.B. das Überarbeiten der Benutzungsschnittstelle für eine bessere Verständlichkeit, müssen ihren Nutzen gegenüber dem Imperativ der Funktionalität erst erstreiten. Der Prozess tendiert somit dahin, sich auf reine, unmittelbar notwendige Funktionalität zu konzentrieren.

Besonders wichtig für die Motivation der vorliegenden Arbeit ist die schwache Position des Mitarbeiters. Ziel dieser Arbeit ist es, einen praktikablen Weg aufzuzeigen, die Folgen des Umgestaltungsprozesses für ihn zu verbessern. Praktikabel muss dabei letztendlich heißen, die zusätzlichen Kosten möglichst gering zu halten.

2.3.3 Verlust des Gestaltungsfreiraums

Um die negativen Folgen eines Arbeitsumgestaltungsprozesses mit Ziel des Einsatzes von Software klarer hervorzuheben, möchte auf den Begriff des *Arbeitssystems* zurückgreifen. In der arbeitswissenschaftlichen Literatur hat sich dieser Begriff durchgesetzt. Er ermöglicht eine systematische Betrachtung der

Elemente von Arbeit und ihren Zusammenhängen. Abbildung 2 zeigt eine Darstellung des Arbeitsystems aus [Arbeitshandbuch 1997].

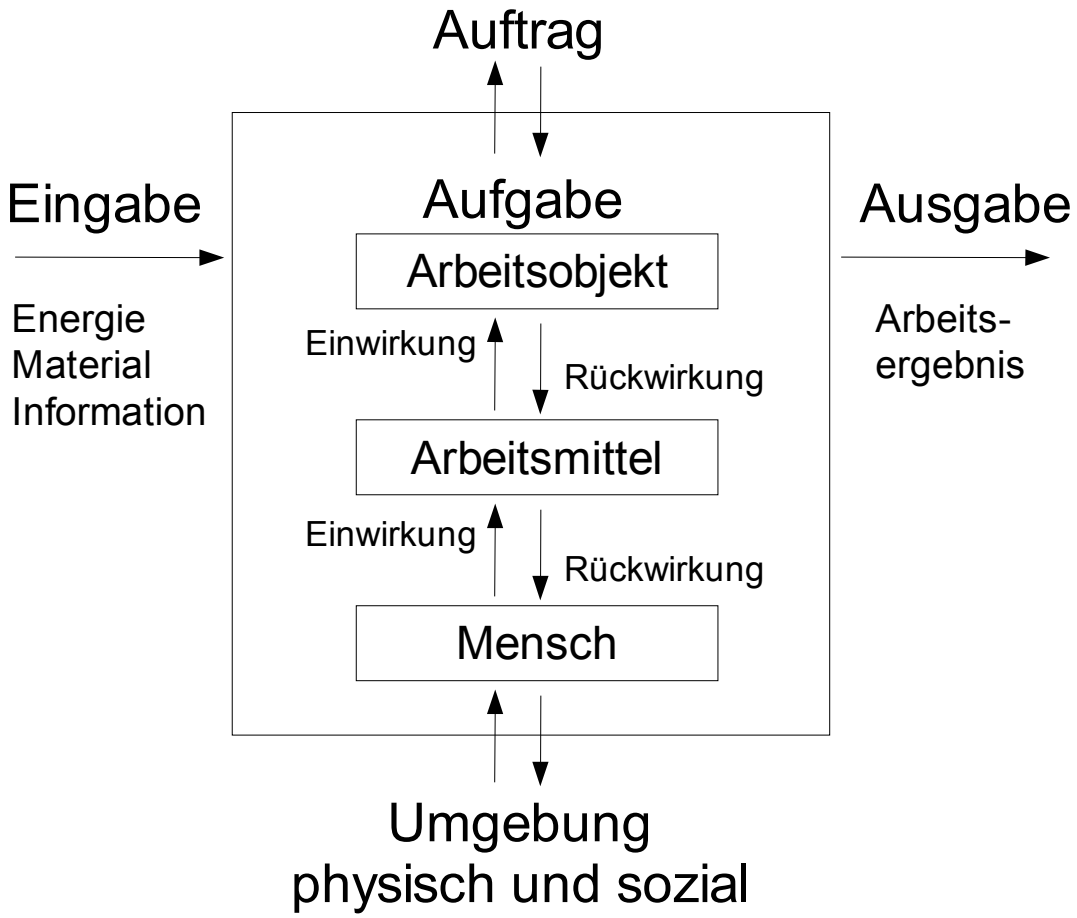

Abbildung 2: das Arbeitssystem (aus [Arbeitshandbuch 1997])

Aus Gründen der Verständlichkeit greife ich zum Erläutern der Grafik auf ein Beispiel aus dem Handwerk zurück: Der Mensch (ein Zimmerer) greift über das Arbeitsmittel auf das Arbeitsobjekt zu, z.B. schlägt er mit einem Hammer (Arbeitsmittel) einen Nagel in eine Holzlatte (Arbeitsobjekt). Die durch einen Auftrag von außen (dem Arbeitgeber) gestellte Aufgabe (z.B. einen Dachstuhl bauen) wird vom Arbeitenden dadurch erledigt, dass er die Eingabe (Nägel und Holzlatten) durch Einwirken über das Arbeitsmittel (Hammer) zur Ausgabe, dem Arbeitsergebnis (dem Dachstuhl), transformiert. Dabei steht der Mensch mit seiner physikalischen (Baustelle) und sozialen Umwelt (Bauherr, Meister) im Kontakt.

Werden sowohl das Arbeitsmittel als auch das Arbeitsobjekt in Software realisiert, wie es bei Software-unterstützter Büroarbeit typischerweise der Fall ist, hat dies unmittelbare Auswirkungen auf die Arbeitsbedingungen. Die Distanz von Mensch zu Arbeitsmittel und Arbeitsobjekt erhöht sich merklich. Beide sind nicht mehr unmittelbar erfahrbar, sondern nur noch über die Oberfläche der Software. Beispielsweise kann ein elektronisches Formular bereits unmittelbar nach der Eingabe für Kollegen verfügbar sein oder aber der Anwender muss dies explizit freischalten. Bei einem materiellen Formular, das er z.B. in einer Registratur ablegt, ist dies unmittelbar erfahr- und nachvollziehbar.

Wesentliche Teile der Arbeitserledigung werden bei Softwareunterstützung in den Rechner verlagert und damit in einen Raum, der dem Arbeitenden nicht unmittelbar zugänglich ist, über den er so gut wie keine Erfahrungen hat und dessen zugrundeliegenden Regeln ihm nicht bekannt sind.

Aus softwaretechnischer Sicht kann für eine gegebene Oberfläche ein beliebiges Verhalten implementiert werden. Die eigentliche Arbeitserledigung findet also in einer Black Box, einem schwarzen Raum oder, in Anlehnung an Arthur C. Clarke[3], einem *magischen* Raum statt. Dieser Raum ist so unverständlich, dass viele Menschen, die tagtäglich mit einem Computer arbeiten, ihm einen eigenen Willen zugestehen, um sein Verhalten zu erklären (wenn auch nur scherzhaft).

Auf Büroarbeit bezogen ist dies mit einer Beschränkung, wenn nicht sogar dem Verlust des Gestaltungsfreiraums in der Ausführung (siehe Abschitt 2.2) gleichzusetzen. Bei einem Umgestaltungsprozess ohne Software-Unterstützung hat der Mitarbeiter noch die Möglichkeit, die für ihn negativen Folgen über die Umgestaltung des Arbeitsprozesses im Kleinen[4] abzufedern. Er kann die konkreten Ausführung bzw. die Selbstorganisation an seine Bedürfnisse anpassen. Bei einem Umgestaltungsprozess unter Einsatz von Software ist er jedoch dem Erzeugnis der Software-Entwickler ausgeliefert.

Selbst ein partizipativer Entwicklungsprozess (siehe „partizipative Systementwicklung" in [Duden Informatik 2001]) kann hier nur bedingt Abhilfe schaffen. Die Software-Entwickler müssen konkrete Arbeitssituationen vorwegnehmen, die sie dann mit Software unterstützen können. Dies kann einen situativen Gestaltungsfreiraum, bei dem die Tätigkeiten an die Aufgabe angepasst werden können, nicht ersetzen.

Man kann dieses Anpassen der Arbeitsmittel an die konkrete Arbeitssituation als eine Kernkompetenz des Menschen sehen, die im Bereich Büroarbeit aufgrund der wenig formalisierbaren Tätigkeiten besonders wichtig ist. Um Software-unterstützte Büroarbeit menschengerechter aber auch produktiver zu gestalten, muss Software deshalb so gestaltet werden, dass der Mitarbeiter genau diese Form der Kontrolle wiedererlangt.

Hier wird ein weiterer Aspekt der von mir geforderten „praktikablen" Lösung erkennbar. Nicht nur muss dieser Gestaltungsspielraum mit vertretbarem Aufwand realisiert werden können, der einzelne Software-Entwickler muss auch in der Lage sein, diesen Gestaltungsspielraum bei der Entwicklung umzusetzen. Dies ist für Arbeitssituationen, die der Entwickler prinzipiell nicht vorweggenommen hat, nicht selbstverständlich.

Darüber hinaus müssen auch die Schlüsselqualifikationen von Software-Entwicklern berücksichtigt werden, die in erster Linie technischer Natur sind. Es ist z.B. wenig realistisch, von Software-Entwicklern kognitionspsychologisches

3 Folgendes Zitat wird Arthur C. Clarke zugeschrieben: „Jede hinreichend fortschrittliche Technologie ist von Magie nicht zu unterscheiden." Dabei ist die Technologie einer möglichen außerirdischen Intelligenz gemeint. In Bezug auf die heutige Computertechnologie könnte man diese Aussage abwandeln: „Jede hinreichend *unverständliche* Technologie ist von Magie nicht zu unterscheiden." Eine Technologie wird dann wie Magie wahrgenommen, wenn ihre inneren Zusammenhänge für Nicht-Eingeweihte undurchschaubar erscheinen.

4 Solche Umgestaltung im Kleinen sind ein häufig anzutreffendes Phänomen im Büroalltag. Die offiziellen Regelungen (Geschäftprozesse) und die tatsächlich gelebte bzw. gearbeitete Realität („Wie wir das machen.") können erheblich voneinander abweichen.

Wissen über ein Maß hinaus zu fordern, das nicht mithilfe üblicher Weiterbildungsmaßnahmen vermittelt werden kann.

2.4 Kontrolle

Ziel dieser Diplomarbeit ist es, nicht nur die negativen Folgen des Umgestaltungsprozesses für den einzelnen Mitarbeiter zu beschreiben, sondern auch einen Weg aufzuzeigen, diese zu verringern. Positive Forderungen aufzustellen hilft dabei, diesen Weg leichter nachzuvollziehen. Deshalb möchte ich an dieser Stelle den Kontrollbegriff aus den kognitiven (Stress-)Theorien der Arbeitspsychologie aufgreifen (siehe [Troy 1981], [Ulich 2001]).

Es wird unterstellt ([Heider 1958]), dass das Bedürfnis, Kontrolle über relevante Umweltbedingungen ausüben zu können, ein Grundmotiv des Menschen darstellt. Für die Arbeitswelt hieße das Kontrolle über Arbeitsbedingungen. Ist dies nicht gegeben (Nichtkontrolle), so wird dies als eine *aversive* Situation wahrgenommen. Aversiv meint dabei, dass sich eine Person dieser Situation unter freien Bedingungen entziehen würde.

Die Folgen von Nichtkontrolle oder Kontrollverlust umfassen dabei einen großen Bereich: von Demotivation bis hin zu dem, was im Allgemeinen Sprachgebrauch als Stress bezeichnet wird. Umgekehrt kann das Vorhandensein von Kontrolle motivierend wirken.

Die wiederholte Erfahrung von Kontrolle oder Nichtkontrolle in vergleichbaren Situationen kann dazu führen, dass sich bestimmte, generalisierte Erwartungen bezüglich der Kontrolle bilden (*Locus of Control*). Dies kann bei häufig wiederkehrenden Erfahrungen auf die Persönlichkeit rückwirken, im Fall von Nichtkontrolle bis hin zu Depressionen (siehe [Seligman 1974], „Theorie der erlernten Hilflosigkeit"). Ein wichtiger Faktor hierfür ist jedoch auch die Ursachenzuschreibung, d.h. die Frage, ob die Person die Nichtkontrolle auf äußere Faktoren zurückführt oder auf das eigene Unvermögen.

Interessant ist, dass die positiven Folgen von Kontrolle unabhängig von dem tatsächlichen Ausüben der Kontrolle auftreten. Norbert Troy hat Versuche durchgeführt ([Troy 1981]), in denen Probanden Aufgaben neben einem Lautsprecher lösen mussten, der einen unangenehmen Ton produzierte. Wurde Probanden ein Knopf angeboten, der diesen Ton kurzzeitig unterband, schnitten sie in den Aufgaben signifikant besser ab als Probanden, denen kein Knopf angeboten wurde. Der Effekt trat auch dann auf, wenn der Knopf während des Versuchs überhaupt nicht genutzt wurde.

Der hier angesprochene Kontrollbegriff beschränkt sich also nicht auf ausgeübte Kontrolle, sondern umfasst bereits das *subjektiv wahrgenommene Kontrollgefühl*. Kontrolle muss also nicht unbedingt ausgeübt werden und kann dabei trotzdem positiv wirken.

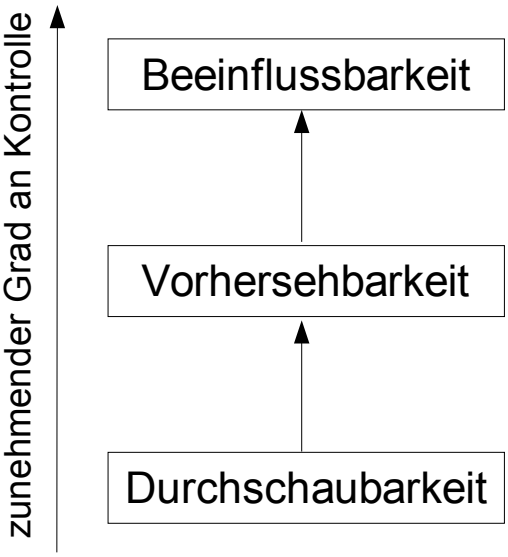

Abbildung 3: Komponenten der Kontrolle nach [Troy 1981]

Kontrolle setzt sich dabei laut Norbert Troy ([Troy 1981]) aus den Kompontenten *Durchschaubarkeit*, *Vorhersehbarkeit* und *Beeinflussbarkeit* zusammen (siehe Abbildung 3):

Durchschaubarkeit

Das Verständnis von für das Individuum bedeutsamen Zusammenhängen und die Erklärung für Ereignisse, Handlungsergebnisse usw.

Vorhersehbarkeit

Das Bilden von Hypothesen und Prognosen.

Beeinflussbarkeit

Die aktive und in der gewünschten Richtung wirksame Einflussnahme auf die Umwelt.

Norbert Troy geht dabei von einer hierarchischen Anordnung dieser Komponenten aus. *Durschaubarkeit* ist eine Voraussetzung für *Vorhersehbarkeit*: Ereignisse können erst dann vorhergesehen werden, wenn die Ursachen aufgetretener Ereignisse verstanden worden sind. *Vorhersehbarkeit* ist wiederum Voraussetzung für *Beeinflussbarkeit*. Die gewünschte Einflussnahme setzt Hypothesen über die Ergebnisse bestimmter Handlungsweisen voraus.

Susanne Maaß weist darauf hin ([Maaß 1994]), dass Kontrolle immer vor dem individuellen Hintergrund einer Person gesehen werden muss. Im Einzelnen heißt das:

- Kontrolle ist immer vor dem Hintergrund der indiviudell verfolgten Ziele zu sehen. Nur Entscheidungsmöglichkeiten bzgl. Ziele, die ein Handelnder tatsächlich verfolgt, wirken sich auf das Kontroll-Erleben

aus.

- (Fehlende) Kontrolle wirkt sich unterschiedlich aus, je nachdem wo die betreffenden Ziele in der persönlichen Zielhierarchie der handelnden Person angesiedelt sind. Wichtige Ziele wiegen schwerer.
- Entscheidungsspielräume sind nur dann als positiv zu bewerten, wenn die einzelnen Entscheidungen keine hohen Risiken bergen. Riskante Entscheidungen werden vermieden und stellen daher keine Erweiterung der Handlungsmöglichkeiten dar.
- Je länger sich eine Person in Situationen der Kontrolle oder Nicht-Kontrolle befindet, desto stärker ist der Einfluss auf das erlebte Maß an Kontrolle. Kontrollerfahrungen führen zu entsprechenden Kontrollerwartungen für die Zukunft.

Die Folgen von Kontrollempfinden bzw. Kontrollverlust betreffen nahezu alle im Abschnitt 2.1 aufgeführten Punkte für die Bewertung von Arbeit: von der Schädigungsfreiheit, die beim Auftreten von Depressionen definitiv nicht mehr gegeben ist, bis hin zur Forderung der Ausführbarkeit, die ein Mindestmaß an Durchschaubarkeit (Verständnis) voraussetzt. Auch kann Kontrolle als Voraussetzung für eine ungehinderte Persönlichkeitsentfaltung bei der Arbeit betrachtet werden. Der Kontrollbegriff fasst damit die für die Gestaltung von Software-unterstützter Büroarbeit wesentlichen Ziele zusammen und macht sie dadurch besser handhabbar.

Transparenz (im Sinne von Durchschaubarkeit) ist eine häufig anzutreffende Forderung in der Software-Ergonomie. In [Maaß 1994] greift Susanne Maaß den Kontrollbegriff aus [Troy 1981] auf, um den Begriff der Transparenz auszudifferenzieren. Unter Rückgriff auf ([Koch, Reiterer und Tjoa]) beschreibt sie eine Einbettung der in der ISO 9241 – Part 10 ([ISO 9241-10] festgehaltenen „Grundsätze ergonomischer Dialoggestaltung” in das Kontrollkonzept. Abbildung 4 zeigt diese Einbettung im Überblick.

Die einzelnen Grundsätze *Aufgabenangemessenheit*, *Erlernbarkeit*, *Individualisierbarkeit*, *Steuerbarkeit*, *Selbstbeschreibungsfähigkeit*, *Erwartungskonformität* und *Übersichtlichkeit* sind dabei kursiv dargestellt. Kontrolle wird hier in erster Linie dem Kriterium *Persönlichkeitsförderlichkeit* des in 2.1 beschriebenen Bewertungsschemas für Arbeit zugeordnet. Wie ich allerdings im vorgehenden Abschnitt gezeigt habe, hat Kontrolle auch Einfluss auf weitere Kriterien. In *[Koch, Reiterer und Tjoa]* und [Maaß 1994] werden die beiden unteren Stufen von Kontrolle, *Durchschaubarkeit* und *Vorhersehbarkeit*, zum Begriff *Orientierung* zusammengefasst.

Kontrolle wird von guter Erlernbarkeit gefördert, die wiederum von Orientierung und Beeinflussung unterstützt wird. Ein hoher Grad an Kontrolle, insbesonders im Hinblick auf die Arbeitserledigung, kann nur eine aufgabenangemessene Software bieten. Ansonsten stellt sich evtl. zwar Kontrolle über die Anwendung, aber nicht über die Arbeitserledigung ein. Diesen Aspekt beschreibt Susanne Maaß nicht explizit, jedoch möchte ich auf ihn weiter unten näher eingehen.

2.5 Transparenz und Gestaltbarkeit

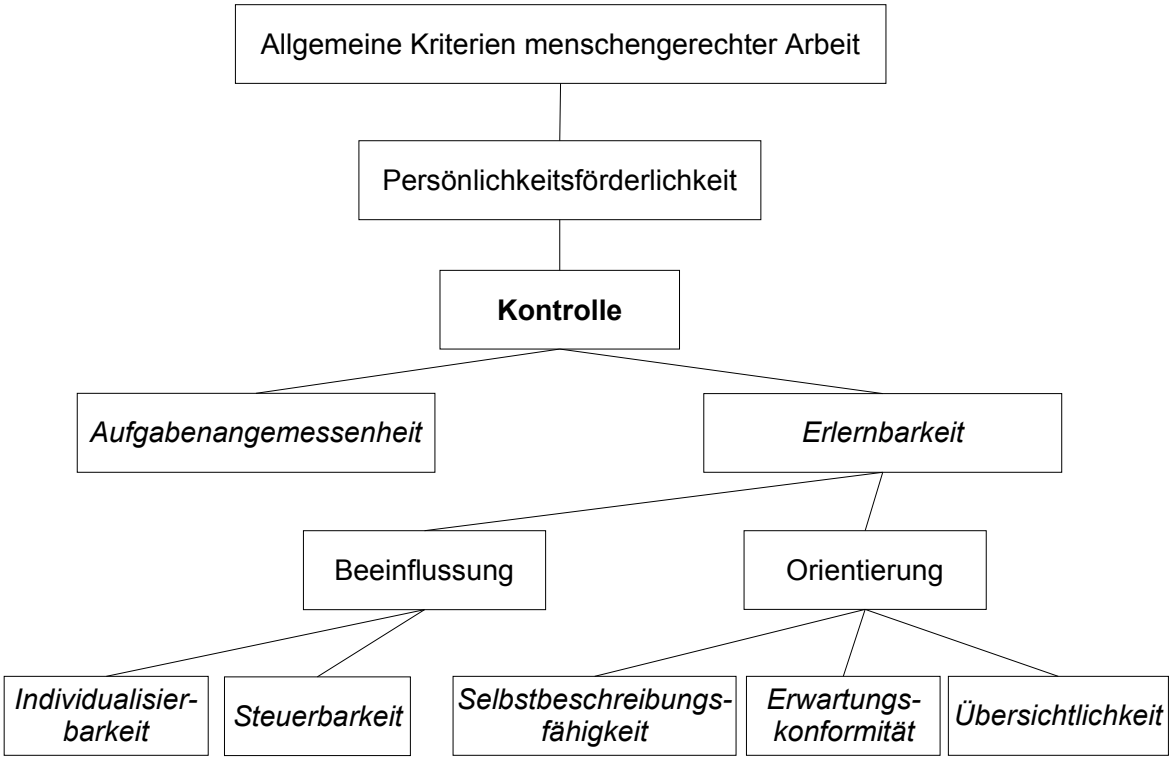

Abbildung 4: Einbettung Software-ergonomischer Kriterien in das Kontrollkonzept (aus [Maaß 1994])

Das Kontrollkonzept schlägt somit die Brücke von den allgemeinen Kriterien der Gestaltung von menschengerechter Arbeit zu den konkreten Software-ergonomischen Forderungen der ISO 9241 – Part 10. Der Kontrollbegriff bietet mit seiner Hierarchie von Durchschaubarkeit, Vohersehbarkeit und Beeinflussbarkeit und seinem theoretischen Hintergrund jedoch den besseren Ausgangspunkt für eine ganzheitliche Betrachtung. Aus diesem Grund möchte ich in mich in dieser Arbeit im Wesentlichen auf den Kontrollbegriff berufen.

Gestaltbarkeit

Insbesondere möchte ich das obere Ende der Kontrollhierarchie betrachten. Dazu möchte ich bestimmte Aspekte des Begriffs *Beeinflussbarkeit* zu einem neuen Begriff *Gestaltbarkeit* zusammenführen.

> **Def.: Gestaltbarkeit**
>
> Gestaltbarkeit beschreibt die höchste Ebene im Kontrollkonzept (siehe Abbildung 5). Gestaltbarkeit ist dann erreicht, wenn der Anwender qualitativen Einfluss auf seine Arbeitserledigung hat. Bespielsweise wenn er die Anwendung so verändern kann, dass er mit ihr neue, so von den Entwicklern nicht explizit vorgesehene Aufgaben erledigen kann.

Michael Frese versteht unter dem Begriff *Einfluss* (und damit Beeinflussbarkeit) nicht nur, dass sich der Benutzer mit dem System auskennt, Eingaben machen kann und somit die Abläufe des Systems steuert. Er meint damit ausdrücklich

auch Möglichkeiten zur Veränderung der Systemfunktionalität und -handhabung durch die Benutzer. Damit handelt es sich nicht um Veränderung des Systemcodes, sondern um eine Ausnutzung von vorgegebenen Varianten oder Modifikationsoptionen.

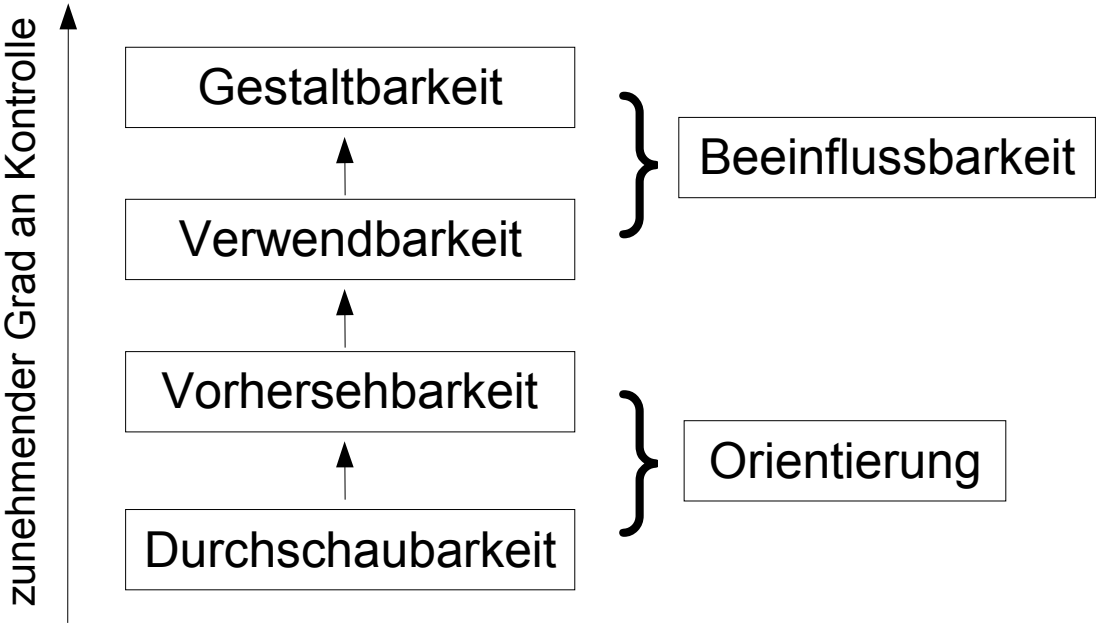

Abbildung 5: Komponenten des ausdifferenzierten Kontrollbegriffs

Zur Verdeutlichung möchte ich den in Abschnitt 2.3.3 beschriebenen Kontrollverlust heranziehen. Mit Gestaltbarkeit soll dieser Kontrollverlust weitestgehend kompensiert werden. Darunter verstehe ich Möglichkeiten zur Anpassung der Anwendung an die konkrete Arbeitssituation, auch wenn diese von den Entwicklern so nicht vorweggenommen wurde. *Gestaltbarkeit* heisst somit Beeinflussung der *Aufgabenangemessenheit* einer Anwendung (siehe Abbildung 6).

Als *Verwendbarkeit* möchte ich den Teil von *Beeinflussbarkeit* benennen, der die Steuerung der Anwendung umfasst in der Art, wie es die Entwickler vorgesehen haben. Allerdings auch „kleinere Anpassungen" der Handhabbarkeit, die aber noch keine qualitativen Veränderungen der Aufgabenangemessenheit mit sich bringen, z.B. Änderungen von Tastaturkürzel. Diese Unterscheidung zwischen Gestaltbarkeit und Verwendbarkeit ist nicht scharf, sondern basiert auf einem qualitativen Kriterium.

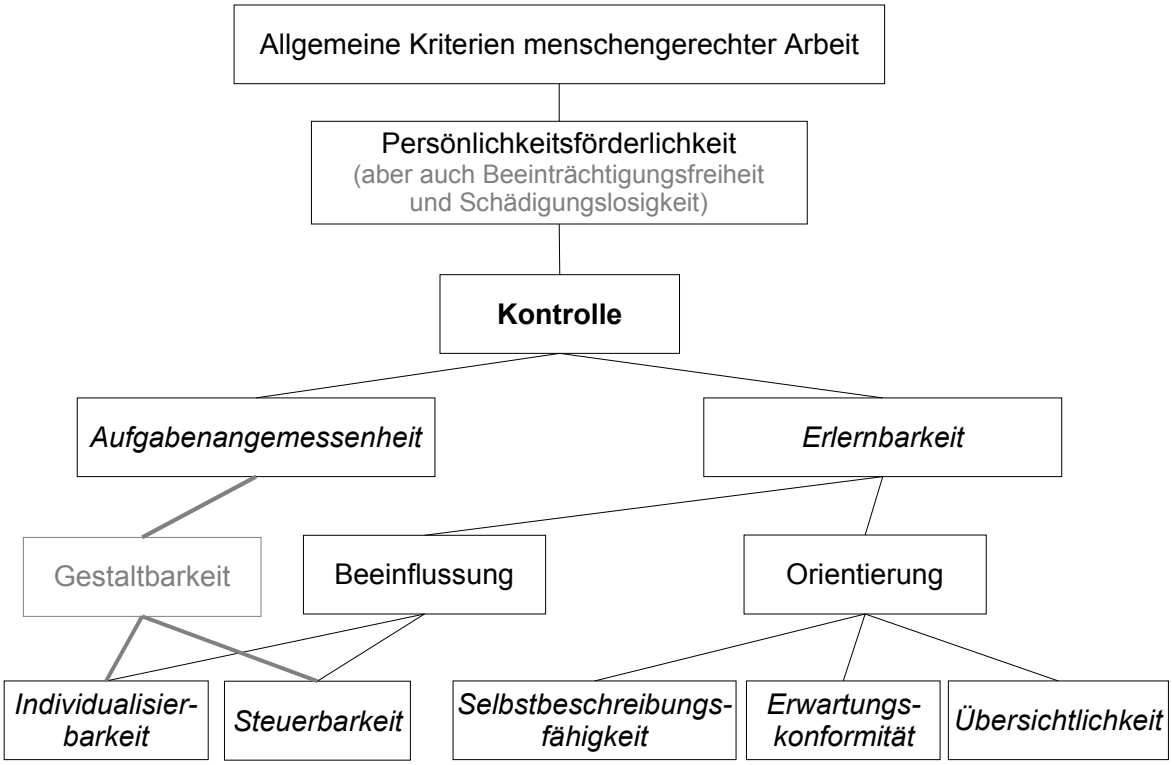

Abbildung 6: Einordnung des Begriffs Gestaltbarkeit *zwischen Software-ergonomischen Kriterien und Kontrollkonzept (in Anlehnung an [Maaß 1994])*

Externe und interne Voraussetzungen von Kontrolle

Somit sind wir bei der zentralen Frage dieser Arbeit angelangt: Wie kann Kontrolle allgemein und Gestaltbarkeit im Besonderen von Anwendungssystemen unterstützt werden? Michael Frese ([Frese 1987]) und Susanne Maaß ([Maaß 1994]) sehen übereinstimmend zwei Arten von Voraussetzungen bezüglich des Anwenders: externe und interne (siehe Abbildung 7).

Zum einen muss ein Anwendungssystem *Orientierung* (interne Voraussetzungen) unterstützen. Susanne Maaß sieht in [Maaß 1994] die Stufe *Orientierung* dann erreicht, wenn sich der Anwender ein *zweckmäßiges mentales Modell* vom System gebildet hat. Nach Michael Frese hat der Anwender dann Kenntnis von den Entscheidungsnotwendigkeiten. Kurz: der Benutzer ist in der Lage, die Anwendung wie vorgesehen zu verwenden und zu bedienen.

Zum anderen muss ein Anwendungssystem aber auch (Entscheidungs-) Spielräume anbieten (externe Voraussetzungen). Kann der Anwender diese nutzen, so hat er vollen Einfluss auf die Anwendung und man kann von Kontrolle bzw. Transparenz sprechen.

Den aktuellen Stand der Forschung dieser beiden Aspekte möchte ich in den beiden folgenden Kapiteln vorstellen. Im Kapitel 3 „Gestaltbarkeit: Bereitstellen von Spielräumen" möchte ich die Möglichkeiten darstellen, wie Spielräume in Anwendungen geschaffen werden können. Orientierung und den Aufbau mentaler Modelle behandle ich im Kapitel 4 „Orientierung: Aufbau mentaler Modelle".

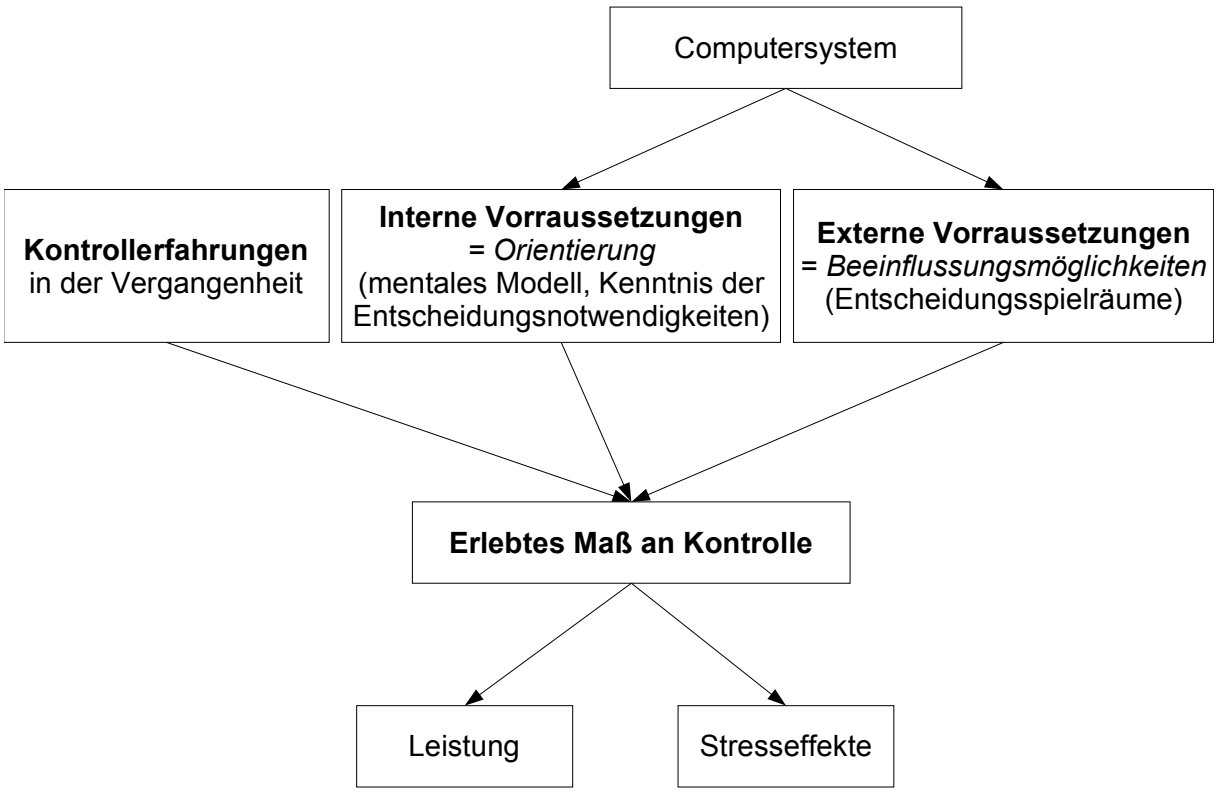

Abbildung 7: Faktoren des subjektiven Kontrollerlebens bei computergestützer Arbeit (aus [Maaß 1994])

2.6 Zusammenfassung

Ausgangspunkt dieser Arbeit ist die Sicht auf Software-Entwicklung als Arbeitsgestaltung. Gestalten heißt immer auch bewerten, deshalb habe ich am Anfang dieses Kapitels einen Katalog von anerkannten Kriterien aus den Arbeitswissenschaften vorgestellt. Ich konzentriere mich dabei auf die Gestaltung von Büroarbeit. Die im Rahmen von Büroarbeit anfallenden Tätigkeiten weisen eine große Bandbreite im Bezug auf ihre Formalisierbarkeit, ihre Stabilität und ihre Häufigkeit auf. Büroarbeit verlangt deshalb von den Mitarbeitern ein hohes Maß an Flexibilität und situativem Handeln. Gestaltungfreiräume in der Ausführung sind dafür ein wichtiges Instrument.

Aus diesem Grund kann Software, die Tätigkeiten vollständig automatisiert, im Bereich Büroarbeit nur in bestimmten Teilbereichen eingesetzt werden. Flexibilität und situatives Handeln sind Kompetenzen, in denen der Mensch stark und Software eher schwach ist. Software für Büroarbeit unterstützt deshalb sinnvollerweise den Menschen in seinen Aufgaben und versucht nicht, ihn in das Korsett einer Ablaufsteuerung zu zwängen. In der Interaktion zwischen Mitarbeiter und Software können so auch wenig formalisierbare Aufgaben bearbeitet werden.

Trotz der Möglichkeiten von unterstützender Software führt ihre Einführung zur Verschlechterung bestimmter Arbeitsbedingungen. Gegenüber der Arbeitserledigung mit rein materiellen Mitteln findet sie mit Software in einem *magischen Raum* statt. Hier sind im Prinzip beliebige Regeln denkbar, sodass letztendlich keine Regeln unbeschränkt Gültigkeit besitzt. Mit dem von mir eingeführten Begriff der *Kontrolle* lässt sich dies als Kontrollverlust beschreiben,

genauer gesagt, einem Kontrolltransfer vom einzelnen Mitarbeiter zu den Software-Entwicklern. Zwangsläufig geht dabei für den Mitarbeiter der unmittelbare Gestaltungsfreiraum verloren.

Software kann schließlich nur durch Software-Entwickler geändert werden, sie haben die Möglichkeit zu gestalten. Da sie bei der Entwicklung die konkreten Arbeitssituationen vorweg nehmen müssen, kann Software nur eingeschränkt Tätigkeiten unterstützen, die ein hohes Maß an Flexibilität und situativem Handeln erfordern. Wegen des fehlenden Gestaltungsfreiraums bleibt dem Mitarbeiter keine Möglichkeit dies auszugleichen.

Dabei verstärkt der Software-Entwicklungsprozess mit seinen verschiedenen Akteuren und ihren Zielen tendenziell diesen negativen Effekt. Im Spannungsfeld zwischen Kosten und Nutzen werden Aufwände vermieden, die nicht unmittelbar Funktionalität erzeugen. Dies ist umso bedauerlicher, als dass dies dem betroffenen Mitarbeiter die Möglichkeit nimmt, seine Kernkompetenz als Mensch zu entfalten. Auch aus betriebswirtschaftlicher Sicht ist dies zu bedauern. Mögliche Produktivität für nicht vorweggenommene oder vorwegnehmbare Tätigkeiten bleibt ungenutzt.

Das Ziel dieser Arbeit ist es deshalb, Möglichkeiten zu untersuchen, wie dem einzelnen Mitarbeiter Kontrolle bei dem Arbeiten mit Software zurückgegeben werden kann. Dazu stellen sich die Fragen, was Kontrolle im Detail bedeutet und wie das Produkt Software praktikabel gestaltet werden sollte, damit das Erreichen von Kontrolle beim Anwender gefördert wird.

Das Adjektiv *praktikabel* soll dabei zwei Dinge betonen: Einerseits darf der Entwicklungsprozess nicht über das für den Arbeitgeber wahrgenommene Kosten-Nutzen-Verhältnis hinaus verteuert werden. Andererseits muss auch der einzelne Software-Entwickler in die Lage versetzt werden, Software entsprechend der Aufforderung von Gestaltbarkeit zu entwickeln.

Kapitel 3: Gestaltbarkeit: Bereitstellen von Spielräumen

Wenn es um Spielräume beim Nutzen von Software geht, so werden in der Literatur viele verschiedene Begriffe synonym verwendet. Susanne Maaß merkt in [Maaß 1994] an, dass das Bereitstellen von (Handlungs-)Spielräumen auch als *Anpassbarkeit* oder *Individualisierung* bezeichnet wird. Detlef Haaks spricht in [Haaks 1991] von *Adaption*, *Adaptierbarkeit* und *Adaptivität*. Reinhard Oppermann schreibt in [Oppermann 1994], dass Tätigkeitsspielräume durch die *Flexibilität des Systems* eingelöst werden. *Individualisierung* hilft dabei, dem Benutzer die wünschenswerte Vielfalt von Systemfunktionalitäten zu vermitteln. Horst Oberquelle ([Oberquelle 1994a]) nennt weitere in der Literatur gebräuchliche Synonyme für *Anpassbarkeit* bzw. *anpassbare Software*: plastic software, malleable systems, tailorable systems, radically tailorable systems, adaptable systems, flexible Systeme, individualisierbare Systeme, anpassbare Systeme.

Ich möchte in diesem Kapitel zunächst einmal eine Definition von Anpassbarkeit vorstellen, die der weiteren Diskussion zugrunde liegt. Anschließend möchte eine systematische Betrachtung der Motivation für Anpassbarkeit vorstellen und in Bezug zum Kapitel 2 setzen. Darauf folgt eine eingehende Betrachtung der Aspekte von Anpassbarkeit, als „Dimensionen der Anpassbarkeit" bezeichnet. Schließlich möchte ich einige Probleme und damit zusammenhängende Forderungen im Zusammenhang mit anpassbaren Anwendungssystemen skizzieren.

3.1 Anpassbarkeit

Horst Oberquelle definiert *Anpassbarkeit* folgendermaßen:

> ***Anpassbarkeit ([Oberquelle 1994a])***
>
> *Unter Anpassbarkeit verstehe ich ganz allgemein die Veränderbarkeit von während der Aufgabenerledigung stabilen Aspekten von Hilfsmitteln (Räume, Verbindungen, Werkzeuge, Steuerungen) als Reaktion auf erkannte lokale Bedürfnisse* während *ihrer Nutzung.*

Er bezieht sich bei der Definition von Anpassbarkeit indirekt auch auf das Arbeitssystem (siehe Abschnitt 2.3.3). Arbeitsgegenstände, die bei der Erledigung einer Aufgabe normalerweise verändert werden, bezeichnet er als „primäre Arbeitsgegenstände". Veränderungen durch Anpassbarkeit beziehen sich im Gegensatz dazu auf die Hilfsmittel. Er unterscheidet also ebenfalls Arbeitgegenstände, die während der Aufgabenerledigung typischerweise verändert werden (Arbeitsobjekte), und Arbeitsgegenstände, die normalerweise stabil bleiben (Arbeitsmittel). Anpassbarkeit bezieht sich auf die Veränderbarkeit der Arbeitsmittel.

3.2 Motivation für Anpassbarkeit

Im Kapitel 2 habe ich die Notwendigkeit von Spielräumen begründet. Der Vollständigkeit halber möchte ich an dieser Stelle eine systematische Betrachtung der Gründe für Anpassbarkeit von Anwendungssystemen vorstellen, wie sie Detlef Haaks in [Haaks 1991] zusammengestellt hat.

Detlef Haaks stellt individuelle Unterschiede zwischen Benutzern einer Anwendung fest. Er identifiziert dabei zwei Kategorien von Unterschiedsdimensionen. Erstens die Unterschiede zwischen verschiedenen Individuen (interindividuelle Unterschiede) und die Unterschiede eines Individuums im Verlauf der Zeit (intraindividuelle Unterschiede). Im Folgenden möchte ich näher auf die einzelnen Unterschiedsdimensionen eingehen:

- **Interindividuelle Unterschiede**
  - *Systemerfahrung*
  - *Nutzungshäufigkeit*
  - *Präferenzen bei Interaktionsformen*
  - *Interaktions- und Handlungsstruktur*
  - *Aufgabenstellung und Aufgabenkontext*
- **Intraindividuelle Unterschiede**
  - *Dynamik der Benutzer*
  - *Dynamik der Aufgabenstellung*

Interindividuelle Unterschiede

Systemerfahrung

Einzelne Benutzer unterscheiden sich durch die Breite ihrer Systemkenntnisse, insbesondere bzgl. ihrer Erfahrungen mit anderen Informationssystemen. Erfahrungen im Umgang mit anderen Informationssystemen sind auch dann bedeutsam, wenn keine oder nur wenig Gemeinsamkeiten mit diesen anderen Systemen vorhanden sind.

Aus dem unterschiedlichen Umfang an Systemerfahrung resultiert die häufig vorgefundene Einteilung von Benutzern vom Neuling bis zum Experten. Sie begründet sich dadurch, dass unerfahrene Benutzer ein hohes Maß an Unterstützung benötigen und vor folgenschweren Fehlern geschützt werden müssen. Erfahrene Systemanwender bevorzugen jedoch eine flexible, effektive Kontrolle des Systems.

Für die Feinheiten gibt es kein allgemein anerkanntes Maß. Häufig ist eine dreistufige Einteilung anzutreffen (Anfänger, Fortgeschrittener und Experte), es werden aber auch bis zu zwölf Benutzerklassen unterschieden. Manche Autoren sind sogar der Meinung, dass feste Kategorien hier nicht angemessen sind.

Detlef Haaks fügt diesen Betrachtungen hinzu, dass die Erfahrung bezüglich eines Gesamtsystems aus der Erfahrung mit den Teilfunktionen resultiert. Der Grad der Systemerfahrung kann also nur in Bezug auf Systemkomponenten bzw. Funktionen sinnvoll bestimmt werden.

Darüber hinaus ist anzumerken, dass sich die Systemerfahrung von Benutzern nicht proportional entwickelt. Sie ist, bedingt durch die Intensität der Systemnutzung, ständigen Schwankungen unterworfen.

Nutzungshäufigkeit

Die Häufigkeit (bzw. Regelmäßigkeit) der Systembenutzung hat einen direkten Einfluss auf die Erinnerungen eines Benutzers. Häufig benutzte Kommandos einer Kommandosprache z.B. werden behalten, während für selten verwendete Kommandos eine Unterstützung notwendig wird (z.B. Online-Hilfe, Handbuch oder ein erfahrener Kollege).

Es gibt verschiedene Arten, die Nutzungshäufigkeit zu beschreiben, z.B. die Dauer und der Abstand einzelner Sitzungen oder aber der Anteil an der Gesamtarbeitszeit.

Die Systemerfahrung ist von der Nutzungshäufigkeit abhängig. Detlef Haaks weist aber darauf hin, dass die nahe liegende Relation

„Systemerfahrung = Nutzungshäufigkeit * Zeit”

nicht zutrifft.

Präferenz bei Interaktionsformen

Das Spektrum an Interaktionsformen ist sehr vielfältig: von den bekannten Techniken der direkten Manipulation, über Kommandosprachen, Menüs, Formulare bis hin zu natürlichsprachlicher Interaktion. Individuelle Präferenzen bezüglich verschiedener Interaktionsformen sind häufig ein Grund für die Bereitstellung mehrerer Interaktionsformen, aus denen der Benutzer wählen kann. Die Präferenzen sind sicherlich auch von der Erfahrung im Umgang mit den einzelnen Interaktionsformen abhängig. Allerdings ist auch das Abweichen von der Präferenz in Abhängigkeit von der Aufgabenstellung vorstellbar.

Interaktions- und Handlungsstruktur

Als Interaktionsstruktur bezeichnet Detlef Haaks den Vorrat an elementaren Interaktionen und ihre Kombinierbarkeit. Handlungsstrukturen resultieren aus der mentalen Repräsentation der Strukturierung von Aufgaben in Teilaufgaben. Einzelne Handlungsschritte und deren Kombination zu Handlungsfolgen werden vom Benutzer ausgeführt, um Teilaufgaben (und damit die Gesamtaufgabe) zu erfüllen.

Benutzer unterscheiden sich bezüglich der Strukturierung von Aufgabenstellungen. Sie verfolgen unterschiedliche Handlungsstrukturen. Bei Nicht-Übereinstimmung der Handlungsstruktur eines Benutzers mit der vom Informationssystem bereitgestellten Interaktionsstruktur ist entweder ein zusätzlicher Transformationsaufwand notwendig oder es treten Handlungsfehler auf, die einen erhöhten kognitiven Regulationsaufwand bewirken.

Zahlreiche arbeitspsychologische Untersuchungen haben nachgewiesen, dass persönlichkeitsspezifische Differenzen in der Aufgabenbewältigung vorliegen. Insbesondere gibt es keinen „optimalen” Weg der Aufgabenbearbeitung und keine „optimale” Interaktionsstruktur.

Aufgabenstellung und Aufgabenkontext

Detlef Haaks unterscheidet Benutzer, die sich hauptsächlich auf ihre eigentliche Aufgabe konzentrieren wollen (qualifizierte Fachkräfte, z.B. Manager oder Wissenschaftler), und Benutzer, deren eigentliche Aufgabenstellung eher in

direktem Zusammenhang mit dem Informationssystem steht.

Qualifizierte Fachkräfte haben häufig wechselnde Fragestellungen zu bearbeiten und kaum Zeit, die Nutzung des Systems zu erlernen. Daher benötigen sie ein hohes Maß an Unterstützung.

Die zweite Gruppe von Benutzern hat einen ungleich höheren Bedarf an Handlungsspielraum, Vollständigkeit und Qualifizierung bei der rechnergestützten geistigen Arbeit. Der Aufgabenbereich eines Benutzers hat auch direkten Einfluss auf seine Erfahrungen im Umgang mit dem Gesamtsystem. Bei einer streng arbeitsteiligen Organisation lernt der Benutzer lediglich eine begrenzte Teilfunktionalität des Informationssystems kennen.

Insofern existiert ein Kontinuum der Beziehung der Aufgabenstellung zum Werkzeug (Arbeitsmittel), das zur Durchführung der Aufgabe eingesetzt wird. Mit einer steigenden Vielseitigkeit der Gesamtaufgabe sinkt die Nutzungshäufigkeit und damit die Notwendigkeit, den rechnergestützten Teil der Aufgabenstellung vielfältig und anspruchsvoll zu gestalten.

Intraindividuelle Unterschiede

Dynamik der Benutzer

Bereits bei der Darstellung der interindividuellen Unterschiede wurde deren Dynamik deutlich. Ein statisches System kann den Anforderungen an die vielfältigen, sich ändernden Benutzeranforderungen nicht gerecht werden. Im Extremfall werden Benutzer anfänglich überfordert, während sie mit der Zeit ein derartiges System als zu umständlich bzw. zu eingeschränkt empfinden. Das System sollte mit dem Benutzer wachsen können.

Insbesondere steigt die Erfahrung eines Benutzers im Umgang mt dem System. Diese Erfahrungen sind im Allgemeinen keine isolierten Systemkenntnisse, sondern entstehen im Kontext der bearbeiteten Aufgabenstellung. Es werden komplexe Probleme bearbeitet, und es bilden sich wiederkehrende Routinearbeiten heraus, die eine „Teilautomatisierung” rechtfertigen. Ein Benutzer entwickelt mit der Zeit einen subjektiven besten Weg der Aufgabenbearbeitung mithilfe des Informationssystems.

In diesem Zusammenhang ist die Flexibilität und die Anpassbarkeit des Dialogsystems sehr wichtig. Eine flexible Systemgestaltung mit mehreren Handlungsalternativen führt insbesondere zu keiner erhöhten Belastung der Benutzer und bedingt eine Erhöhung der Effizienz. Die Handlungsfreiräume werden benutzt und stimulieren innovatives Verhalten. Die Anpassung der Handlungsstrukturen an die individuellen Ziele und die mentale Aufgabenrepräsentation führt zu einer erhöhten Effizienz der Systemnutzung.

Dynamik der Aufgabenstellung

Der Tatsache, dass auch die Aufgabenstellungen, die mit einem Informationssystem bearbeitet werden sollen, Änderungen unterworfen sind, wird im Zusammenhang mit der Anpassbarkeit von Informationssystemen nur wenig Beachtung geschenkt.

Oft liegt der Grund darin, dass von einer festen „Spezifikation” ausgegangen wird,

was sich wiederum aus dem traditionellen linearen Phasen-Modell der Software-Entwicklung begründet. Detlef Haaks stützt sich auf [Floyd 1987], wenn er feststellt, dass diesem Modell ein technischer Blickwinkel zugrunde liegt, mit dem Ziel, eine bestehende Organisation in einem Softwaresystem einzufrieren. Dabei werden mögliche Änderungen der Organisation, die aufgrund des Einsatzes des Informationssystems (als Wirkung) eintreten können, nicht betrachtet. Als Alternative schlägt er evolutionär partizipative Ansätze, wie z.B. STEPS ([Floyd 1987]), vor. WAM ist ebenfalls solch ein evolutionär partizipativer Ansatz ([Züllighoven et al. 1998]).

Insgesamt wird eine Abkehr vom traditionellen Ansatz der Konzeption von Informationssystemen – der Automatisierung häufig vorkommender Arbeiten – in Richtung einer höheren Flexibilisierung gefordert. Das bedeutet, dass die Anpassung des Systems an veränderte Aufgabenstellungen bereits im Systemdesign berücksichtigt werden muss.

Zum Abschluss dieses Abschnitts möchte ich einen Bezug zum Kapitel 2 herstellen. Dort habe ich die Forderung nach Gestaltbarkeit im Wesentlichen mit den hier vorgestellten Punkten *Aufgabenstellung und Aufgabenkontext, sowie Dynamik der Aufgabenstellung* argumentiert. Dies ist aufgrund der Nähe des Begriffs Gestaltbarkeit zur Aufgabenangemessenheit einer Anwendung begründet. Gestaltbarkeit als höchste Stufe der Kontrolle muss aber letztendlich auch Kontrolle über die anderen hier angesprochenen Aspekte bieten.

3.3 Dimensionen der Anpassbarkeit

Nachdem wir einen systematischen Blick auf die Gründe für anpassbare Anwendungen geworfen haben, möchte ich nun die Bedeutung des Begriffs *Anpassbarkeit* in ihren verschiedenen Dimensionen beleuchten. Dabei stütze ich mich im Wesentlichen auf Detlef Haaks, der diese Dimensionen in [Haaks 1991] vorgestellt hat.

Detlef Haaks sieht *Anpassbarkeit* als Oberbegriff für Adaptierbarkeit und Adaptivität, spricht aber von *Adaption*. Diese beiden Begriffe sehe ich synonym. Aus Gründen der Konsistenz werde ich aber weiterhin den Begriff *Anpassbarkeit* verwenden.

Folgende Dimensionen prägen den Begriff *Anpassbarkeit*:

- Gegenstand
- Initiator und Akteur
- Ziel
- Zeitpunkt
- Geltungsbereich

3.3.1 Gegenstand

Am deutlichsten wird der Begriff Anpassbarkeit, wenn man seine konkreten Erscheinungsformen betrachtet. Was kann angepasst werden, was sind also die

Gegenstände der Anpassung? [Haaks 1991] gibt einen Überblick, der in [Oppermann 1994] aufgegriffen und dort im Detail ergänzt wird. Dabei erheben beide Autoren keinen Anspruch auf Vollständigkeit.

a) Synonyme

b) Defaultwerte

c) Hilfestellung

d) Interaktionsformen

e) Interaktionsinitiative

f) Handlungspläne

g) Einschränkung des verfügbaren Funktionsvorrats

h) Modifikation und Erweiterung der Funktionalität

a) Synonyme

Bestimmte Funktionsbezeichnungen, wie z.B. Kommandonamen, lösen bei Benutzern gelegentlich falsche Assoziationen aus. Kann der Benutzer selbst Synonyme definieren, so kann dies Missverständnisse beseitigen und Klarheit schaffen.

b) Defaultwerte

Das Anbieten von Standard-Werten, z.B. für Formulare, dient dem Vereinfachen der Handhabung und kann die Effizienz erhöhen. Diese können von den Entwicklern während der Entwicklung oder von den Benutzern während der Nutzung vergeben worden sein. Oft kann man den letzten Wert oder aber auch den häufigsten Wert als Voreinstellung übernehmen. Auf die explizite Definition von Voreinstellungen wird oft verzichtet.

c) Hilfestellungen

Unterschiedliche Benutzer haben unterschiedlichen Bedarf an Hilfestellungen. Hilfesysteme können dem Benutzer hinsichtlich des Grades an Ausführlichkeit, der Aktivierungszeitpunkte der vermittelten Inhalte und der Präsentationsart unterschiedliche Unterstützungen anbieten.

d) Interaktionsformen

Mit Interaktionsformen bezeichnet Detlef Haaks elementare Interaktions-Konzepte von Benutzungsschnittstellen, wie z.B. Fenster, Piktogramme, Buttons, Menüs, Kommandosprachen, natürliche Sprache. Er unterscheidet dabei ausgabe- und eingabespezifische Interaktionsformen. Sie bieten dem Benutzer vielfältige Möglichkeiten der Auswahl, Anpassungen und Kombination. Die heutigen Anwendungssysteme bieten überwiegend redundante Interaktionsformen an, d.h. sie bieten verschiedene Interaktionstechniken parallel an und überlassen dem Benutzer jederzeit die freie Wahl, die eine oder andere Form der Interaktion zu wählen. Das gleichzeitige Anbieten verschiedener Interaktionsformen wird auch als Multimodalität oder Vielfältigkeit (siehe [Oppermann 1994]) bezeichnet. Es

wird als „einfachste” Form von Flexibilität gesehen.

Detlef Haaks weist darauf hin, dass die kombinierte Bereitstellung verschiedener Interaktionsformen jeweils spezifische Vor- und Nachteile in Abhängigkeit von der Benutzergruppe, der Aufgabenstellung und dem Anwendungsgebiet haben können.

e) Interaktionsinitiative

Benutzer unterscheiden sich in ihrer Abhängigkeit von Interaktionsaufforderungen. Es ist eindeutig, dass Fortgeschrittene und besonders Experten Systeme mit benutzerinitiierten Interaktionsstilen bevorzugen. Tendenziell benötigen Anfänger mehr Interaktionsführung, eventuell kann es aber zu einem Konflikt mit dem Bedürfnis kommen, ein System frei zu explorieren. Eine komplexe Form der benutzergesteuerten Interaktionsinitiative stellen so genannte Makros dar. Mit ihnen fasst der Benutzer eine Reihe von elementaren Interaktionsschritten zu einem Paket zusammen und führt sie in der Gebrauchsphase mit einem Aufruf aus.

f) Handlungspläne

Eine besondere Form von Makros stellen Handlungspläne dar. Ein Handlungsplan ist die Beschreibung eines Arbeitsablaufes, der zu einem bestimmten, vordefinierten Arbeitsziel führt.

Abbildung 8 zeigt Beispiele für hierarchische Handlungspläne auf Basis von UND-ODER-Graphen.

Handlungspläne stellen eine zielgerichtete Modellierung von Benutzeraktionen dar, die vor der eigentlichen Benutzung (durch einen Experten) definiert oder während der Nutzung von dem System abgeleitet werden.

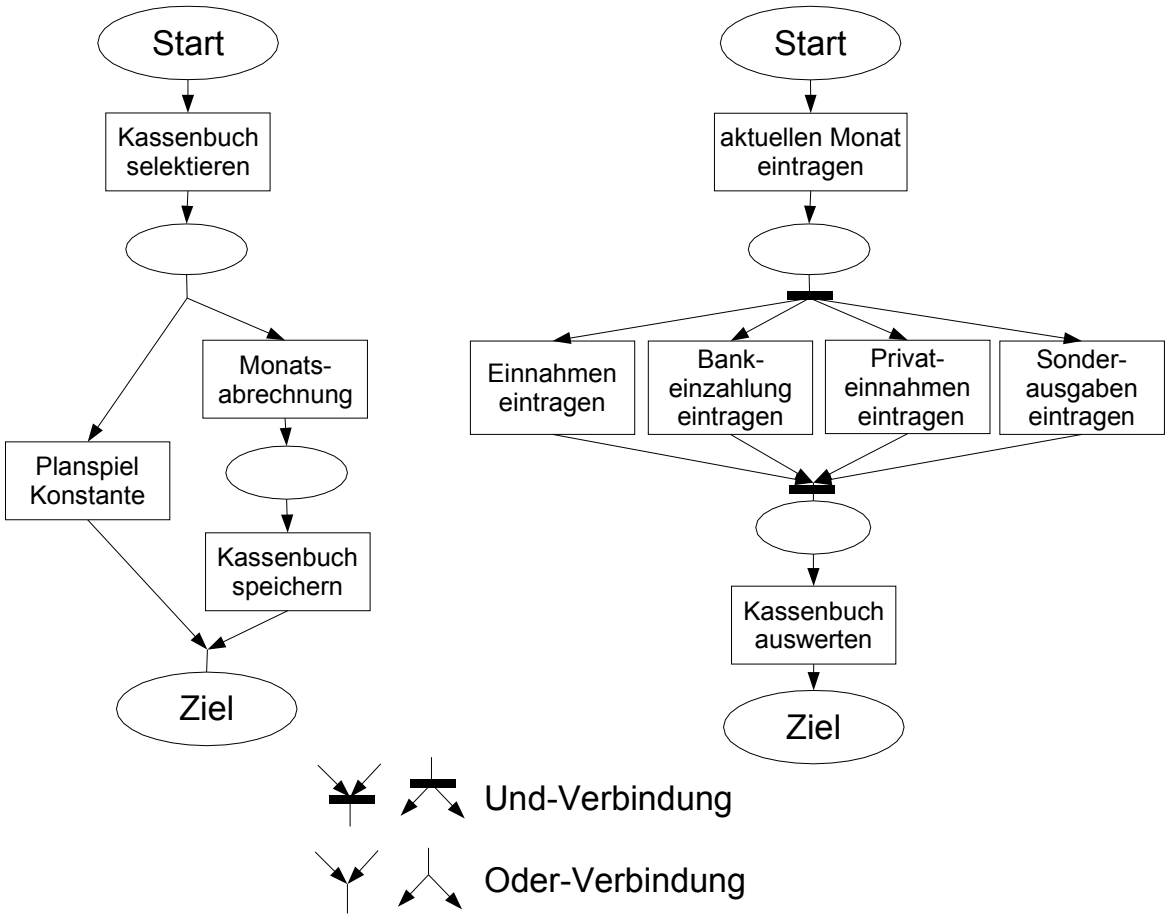

Abbildung 8: Hierarchischer Handlungsplan für eine Tabellenkalkulations-Anwendung als UND-ODER-Graph (entnommen aus [Haaks 1991])

g) Einschränkung des verfügbaren Funktionsvorrats

Es wird verschiedentlich vorgeschlagen, dem Benutzer für eine Trainingsphase einen beschränkten Funktionsvorrat zur Benutzung freizugeben. Der Vorteil soll in einer besseren Überschaubarkeit der Funktionsmenge und einem Schutz vor riskanten Operationen liegen. Nicht zugängliche Funktionen werden entweder erst gar nicht angezeigt oder passiv gehalten.

h) Modifikation und Erweiterung der Funktionalität

Die Anpassung von Funktionalität an benutzer- und aufgabenspezifische Anforderungen ist gegenwärtig erst im Ansatz möglich. Zu nennen sind stufig angebotene Funktionsumfänge, das Anlegen von Makros zur Kombination von Grundfunktionen und das Kombinieren von verschiedenen Anwendungen unter einer gemeinsamen Interaktionsoberfläche.

3.3.2 Initiator und Akteur

Einer der am häufigsten diskutierten Unterteilungen anpassbarer Systeme ist die Unterscheidung nach dem Akteur der Adaption. Dies kann das System selbst sein, wenn es sich „automatisch" und „selbstständig" an den Benutzer anpasst. Ein solches System wird *adaptiv* genannt. Geht die Initiative vom Benutzer aus,

passt also der Benutzer das System an, so spricht man von *adaptierbaren* Systemen. Die „Feinheit" dieser Unterteilung ist jedoch noch nicht allgemein akzeptiert und noch Gegenstand einer kontroversen Diskussion.

Detlef Haaks leitet daraus zwei Aspekte dieser Anpassungsdimension ab: Wer initiiert die Adaption und wer führt sie durch?

Die Rollen von Initiator und Akteur können dabei von Personen (adaptierbare Systeme) oder vom (adaptiven) System selbst übernommen werden. Die Grenze ist dabei aber nicht eindeutig. Ein System kann z.B. einen Benutzer auffordern, eine Anpassung vorzunehmen oder zu bestätigen. Oder aber ein Benutzer kann eine Adaption initiieren, die dann vom System durchgeführt wird. Bei solchen Fällen scheint es wenig sinnvoll, scharf zwischen adaptivem und adaptierbarem Verhalten trennen zu wollen.

Reinhard Oppermann unterscheidet in [Oppermann 1994] bei der Anpassung eines Systems vier unterschiedliche Rollen:

- Entwickler
- lokaler Experte
- System
- Benutzer

Entwickler

Der Entwickler implementiert vielfältige[5] Interaktionsmöglichkeiten; er implementiert adaptierbare Interaktionsvarianten und bietet für die Nutzung der Adaption entsprechende endbenutzergerechte Werkzeuge an; er implementiert schließlich auch die Mechanismen der Protokollierung und Auswertung von Benutzeraktionen zur Realisierung automatischer Adaptionen durch das System. Er übernimmt auch in adaptiven Systemen die Verantwortung für Inhalt und Zeitpunkt der Anpassung.

Lokaler Experte

Ein eventuell vorhandener lokaler Experte informiert und berät den Benutzer über Möglichkeiten vielfältiger Systemnutzung, über Adaptierungsmöglichkeiten durch den Benutzer (die er u.U. dann auch gleich in Abstimmung mit dem Benutzer für diesen ausführt) und über das Zustandekommen und die Effekte adaptiver Leistungen.

System

Das System bietet gegebenenfalls vielfältige Systemnutzungs- und Adaptierungsmöglichkeiten an und führt automatische Adaptierungen auf der Basis der vom Entwickler implementierten Mechanismen aus.

Benutzer

Der Benutzer nutzt die vielfältigen Systemhandhabungs- und

5 Hiermit ist das gleichzeitige Anbieten verschiedener Interaktionsformen gemeint, auch *Multimodalität* genannt (siehe Abschnitt 3.3, Gegenstände der Adaption, d) Interaktionsformen).

Adaptierungsmöglichkeiten sowie autoadaptive Leistungen; eine initiative Rolle kommt ihm bei der Auswahl von Varianten, bei vielfältigen Systemnutzungsmöglichkeiten und bei der Entscheidung für sowie der Realisierung von adaptierbaren Leistungen zu; bei der autoadaptiven Variante kommt es je nach Systemauslegung u.U. zu Klärungsdialogen zwischen System und Benutzer, wodurch eine autoadaptive Leistung moderiert werden kann.

3.3.3 Ziel

Diese Dimension von Anpassung sollte vielleicht besser als „Grund" bezeichnet werden. Gemeint ist die grundlegende Motivation, warum eine Anpassung durchgeführt werden soll. Die Anpassung der Software ist nur Mittel zum Zweck und nicht das eigentliche Ziel.

Folgende Ziele werden in [Haaks 1991] genannt:

- Benutzer
- Aufgaben
- Organisation
- Hardware

Eine Anpassung kann als Grund die Anpassung an den *Benutzer* oder aber die Anpassung an eine spezielle *Aufgabe* haben. Eine solche Aufteilung muss allerdings differenziert betrachtet werden. So kommt Detlef Haaks (mit Rückgriff auf [Oberquelle 1986]) zu dem Schluss, dass die Ziele „Benutzer" und „Aufgabe" voneinander abhängig sind. Persönliche Arbeitsstile können z.B. bei der gleichen Aufgabenstellung Funktionen erfordern oder überflüssig machen. Auch kann bei einer gegebenen Anpassung nicht immer sinnvoll getrennt werden, ob sie benutzer- oder aufgabenbezogen ist. Beispielsweise kann die Definition eines Makros sowohl eine Anpassung an die Aufgaben als auch an den Benutzer sein.

Ein Anwendungssystem kann auch an die Anforderungen einer *Organisation* angepasst werden. Innerhalb von Organisationen existieren Standards und Konventionen, die in der Software berücksichtigt werden sollten. Darüber hinaus sind Anwendungssysteme häufig mit hohen Investitionen verbunden. Anstatt sie bei neu auftretenden Anforderungen zu ersetzen, sollten sie angepasst werden können.

Aus dem gleichen Grund sollte das Anwendungssystem an neue *Hardware* anpassbar sein bzw. neue Hardware sollte in das System integriert werden können.

Zusammenfassend stellt Detlef Haaks allerdings fest, dass der Anpassung an den Benutzer und die Aufgabe die größte Bedeutung unter den genannten Punkten zukommt.

3.3.4 Zeitpunkt

Eine Anpassung kann *vor der ersten Benutzung* des Systems[6], *zwischen*

6 Horst Oberquelle bezeichnet in [Oberquelle 1994a] diese spezielle Form der Anpassbarkeit als „Konfigurierbarkeit".

einzelnen Benutzungsphasen und *während der Benutzung* durchgeführt werden. Mit dem Zeitpunkt der Adaption sind Einschränkungen der Anpassbarkeit verbunden. Vor der ersten Benutzung, z.B. vor Auslieferung des Systems, ist eine Anpassung an individuelle Benutzer wenig sinnvoll. Es können aber sinnvolle Anpassungen für Benutzergruppen, z.B. Großkunden, vorgenommen werden.

Der Zeitpunkt der Anpassung korreliert offensichtlich mit dem Akteur der Anpassung. Vor der ersten Systembenutzung können Anpassungen von erfahrenen Benutzern (bzw. System-Experten) durchgeführt werden. Adaptive Systeme führen Anpassungen im Allgemeinen während der Benutzung durch, genauso wie der Benutzer selbst auch.

Detlef Haaks merkt an, dass die ausschließliche Anpassung eines Systems vor der ersten Benutzung nur die interindividuellen Benutzer-Unterschiede berücksichtigt. Dies führt zu einer unzureichenden Flexibilität, die sich in statischen Arbeitsabläufen widerspiegelt und die die Arbeitsleistung eines Benutzers stark einschränken kann. Dies gilt insbesondere bei Änderungen der Aufgabenstellung. Aus den intraindividuellen Unterschieden der Benutzer ergibt sich die Notwendigkeit, ein System zu adaptieren, nachdem es bereits benutzt wurde. Die größte Flexibilität wird durch eine während der Benutzung durchführbare Adaption erreicht.

3.3.5 Geltungsbereich

Als Geltungsbereich bezeichnet Detlef Haaks den Bereich, der von einer Anpassung betroffen ist. Er nennt folgende Dimensionen bezüglich des Geltungsbereichs:

- Personenkreis
- Zeitraum
- betroffene Komponenten

Der bei einer Anpassung betroffene *Personenkreis* kann einen einzelnen Benutzer, Benutzergruppen oder alle Benutzer umfassen. Anpassungen an individuelle Benutzer sind insbesondere in heterogenen Personenkreisen von Bedeutung, während die Anpassung an Benutzergruppen oft in Zusammenhang mit einer geänderten Aufgabenstellung der Benutzergruppe zusammenhängt.

Den *Zeitraum* von Anpassungen betrachtet Reinhard Oppermann in [Oppermann 1994] etwas detaillierter als Detlef Haaks (in Klammern die Bezeichnungen aus [Haaks 1991]). Beide nennen Sitzungsperiode (Benutzungsphase) und Anpassung bis zum expliziten Widerruf (Gültigkeit bis zur nächsten Adaption). Darüber hinaus nennt Reinhard Oppermann die momentane Situation, also Anpassungen für einmalige Verwendung, z.B. in einem Dialog. Dieser Zeitraum wird häufig auch von adaptiven Systemen genutzt, z.B. wenn sie bestimmte Werte in einem Dialog vorschlagen.

Es kann eine *Komponente* gezielt oder es können mehrere Komponenten entsprechend ihrer gemeinsamen Eigenschaften adaptiert werden. Dabei sind verschiedene Granularitäten möglich, die von kleinen System-Bausteinen bis zum Gesamtsystem reichen. Beispielsweise kann ein einzelnes Interaktionsobjekt (z.B. ein bestimmtes Piktogramm), eine Gruppe von Interaktionsobjekten (z.B. alle

Menüs) oder ein ganzes Teilsystem (z.B. Hilfesystem) adaptiert werden. Dabei ist es wichtig, dass der Benutzer den Geltungsbereich der von ihm durchgeführten Adaption genau kennt.

Reinhard Oppermann zählt in [Oppermann 1994] die Gültigkeit von Anpassungen für ein einzelnes Dokument zu der Kategorie „Geltungsdauer" (bzw. Zeitraum). Mir erscheint es jedoch als sinnvoller, auch Dokumente als „Komponente" zu betrachten und an dieser Stelle einzusortieren.

Der Geltungsbereich ist nicht unbedingt unabhängig von den anderen Dimensionen der Anpassbarkeit. Oft wird der Geltungsbereich durch den Initiator einer Adaption festgelegt. Ein einzelner Benutzer ist i.d.R. nicht berechtigt, Anpassungen für andere Benutzer durchzuführen. Dies kann allerdings auch gewünscht sein. Diesen Aspekt betrachtet Horst Oberquelle unter dem Begriff „Groupware" ausführlich in [Oberquelle 1994a].

3.3.6 Einordnung der Arbeit bezüglich der Dimensionen

Nachdem ich die verschiedenen Bedeutungs-Dimensionen von Anpassbarkeit vorgestellt habe, möchte ich vor dem Hintergrund des in Abschnitt 2.3.3 definierten Begriffs *Gestaltbarkeit* für diese Arbeit einen Fokus setzen.

Im Hinblick auf die Gegenstände der Anpassung möchte ich keine Einschränkungen machen, obwohl ich in dieser Arbeit nur die Anpassung einiger weniger Gegenstände beschreiben kann. Dies soll aber keine konzeptionelle Einschränkung bedeuten. Ich möchte ganz im Gegenteil einen konzeptionellen Ansatz finden, der es erlaubt, so vieleGegenstände einer Anwendung wie möglich anpassbar zu machen.

Dem Benutzer soll Kontrolle über seine Arbeit zurückgegeben werden, insofern soll er sowohl als Initiator als auch als Akteur der Anpassungen im Fokus dieser Arbeit stehen. Damit grenze ich das *System* als Akteur und Initiator von Anpassungen aus. Den *lokalen Experten* sehe ich als wichtige Rolle, die für meine Betrachtungen allerdings eher von organisatorischer Bedeutung ist. Ich sehe ihn als einen sehr erfahrenen Benutzer. Normalen Benutzern stehen im Prinzip die gleichen Möglichkeiten zur Verfügung wie einem lokalen Experten. Die Beziehung zwischen Benutzer und Entwickler werde ich noch gesondert betrachten.

Als *Ziel* der Anpassungen möchte ich mich auf benutzer- und aufgabenspezifische Anpassungen konzentrieren. Organisations- und Hardware-spezifische Anpassungen werde ich nicht betrachten.

Mit Gestaltbarkeit ist der *Zeitpunkt* auf Anpassungen während der Benutzung gesetzt. Erst dann hat der Benutzer die Möglichkeit, die Software an eine konkrete Arbeitssituation anzupassen.

Der *Geltungsbereich* nimmt für diese Arbeit eine zentrale Rolle ein. Mit dem Konzept des *Arbeitszusammenhangs* möchte ich einen neuen, vom Anwender explizit beeinflussbaren Geltungsbereich vorstellen bzw. vergegenständlichen. Er soll Anpassungen für eine bestimmte Arbeitssituation vergegenständlichen und stellt so den „Ort" dar, an dem der Anwender seine Anpassungen vornimmt.

3.4 Praktische Hürden beim Einsatz anpassbarer Systeme

Die Vorteile von Anpassbarkeit sind allgemein akzeptiert. Anpassbare Systeme

sind aber auch immer komplexer als statische Systeme. Damit gehen spezifische Probleme einher, die ich hier kurz skizzieren möchte.

([Oberquelle 1994a]) nennt folgende Punkte:

- **erschwerte zentrale Benutzerunterstützung**

  Wegen vieler verschiedener möglicher Anpassungen wird es für eine zentrale Benutzerunterstützung schwierig, Hilfestellungen anzubieten.

- **Probleme im Vertretungsfall**

  Wird ein Mitarbeiter durch einen anderen vertreten, z.B. im Krankheitsfall, muss dieser sich erst in das angepasste Anwendungssystem einarbeiten.

- **undurchsichtig im Hinblick auf Leistungserbringung**

  Bei einem weit reichend angepassten Anwendungssystem kann es nur noch schwierig nachzuvollziehen sein, wie eine bestimmte, angepasste Leistung erbracht wird.

- **erschwerte Zusammenarbeit an gemeinsamen Dokumenten**

  Soll ein Dokument zusammen über ein Netz bearbeitet werden, so werden dafür eventuell zwei unterschiedlich angepasste Anwendungen verwendet. Dass diese immer noch kompatibel sind, ist nicht automatisch gegeben.

- **problematisch bei gemeinsamer Nutzung**

  Wird eine Anwendung von verschiedenen Benutzern geteilt, z.B. auf einer Krankenstation in einem Krankenhaus, können die Anpassungen der einzelnen Benutzer in Konflikt zueinander stehen.

Als Konsequenz fordert Horst Oberquelle in [Oberquelle 1994a], dass bei gemeinsamer Nutzung, wie z.B. beim Schichtdienst, die Anpassungen allen Beteiligten bekannt gemacht werden sollen, bzw. dass sie am besten abgestimmt werden sollen. Anpassungen sollten deshalb als sozialer Lernprozess einer Gruppe (von Benutzern) begriffen und nicht jedem Benutzer einzeln überlassen werden. Detlef Haaks ([Haaks 1991]) sieht dies genauso und vermutet darüber hinaus, dass Anpassung als Gruppenaktivität von Benutzern mit ähnlicher Aufgabe zu verstärkter Interaktion und gemeinsamem Lernen führen kann.

Horst Oberquelle weist auf ein spezielles Problem („retrofitting”) hin, das bei der Einführung einer neuer Versionen der Software auftritt. Anpassungen an der alten Version sollten mit übernommen werden, um ein möglichst reibungsfreies Weiterarbeiten zu ermöglichen. Dieses Problem ist nicht zu unterschätzen.

Fehlerrobustheit ist bei anpassbaren Systemen sehr wichtig. Im ungünstigsten Fall könnte sich der Anwender das System bis zur Unbenutzbarkeit verstellen. Horst Oberquelle schlägt hier in [Oberquelle 1994a] eine Art Versionsmanagement vor: eine *Basisversion*, zu der der Anwender immer zurückkehren kann, eine *Anfangsversion*, durch initiales Konfigurieren entstanden

(möglicherweise sogar benutzer- oder gruppenspezifisch), sowie eventuell *situationsspezifischere Versionen,* zwischen denen nach Bedarf umgeschaltet werden kann. Außerdem sollten die Anpassungen im Sinne von „Undo" rückgängig gemacht werden können. Diese beiden Punkte sind insbesondere deshalb zu beachten, da risikobehaftete Entscheidungsspielräume i.d.R. vermieden werden (siehe Abschnitt 2.4). Gerade für Einsteiger besteht jedoch die Gefahr des ungewollten Anpassens.

3.5 Zusammenfassung: nicht vorweggenommene Spielräume

In diesem Kapitel habe ich die wichtigsten Aspekte vorgestellt, die die aktuelle Forschung in Bezug auf Spielräume in Anwendungen behandelt. Zuerst habe ich eine Definition von Anpassbarkeit vorgestellt, die sich zu dem im Abschnitt 2.3.3 vorgestellten Arbeitssystem in Beziehung setzen lässt. Anpassbarkeit heißt die Möglichkeit zum Verändern der Arbeitsmittel.

Anschließend habe ich eine systematische Betrachtung der Motivation für Anpassbarkeit aufgegriffen. Die Notwendigkeit von Anpassungsmöglichkeiten ergibt sich einerseits aus Unterschieden zwischen einzelnen Anwendern (interindividuelle Unterschiede) und andererseits aus einer Dynamik in Bezug auf eine einzelne Person und die jeweilige Aufgabenstellung (intraindividuelle Unterschiede). Die Dynamik bezüglich der Aufgabenstellung ist das wesentliche Argument, das ich auch im Kapitel 2 angeführt habe.

Im zentralen Abschnitt dieses Kapitels habe ich Dimensionen vorgestellt, die bei der Charakterisierung des Begriffs *Anpassbarkeit* helfen: Gegenstände, Initiator und Akteur, Ziel, Zeitpunkt und Geltungsbereich von Anpassungen. Dabei habe ich die in dieser Arbeit betrachtete Form von Anpassungen, die vor dem Hintergrund des Gestaltbarkeits-Begriffs zu sehen ist, entsprechend dieser Dimensionen eingeordnet.

Schließlich habe ich einige praktische Hürden skizziert, die bei der Umsetzung von Anpassbarkeit in Anwendungen berücksichtigt werden sollten. Als wichtigster Punkt ist hier das Vermeiden von negativen Konsequenzen zu nennen, wie z.B. durch eine Versions-Verwaltung und die Möglichkeit zum „Undo" auch für Anpassungen.

Vorweggenommene und nicht-vorweggenommene Spielräume

Zum Abschluss dieses Kapitels komme ich nochmals auf die ursprüngliche Motivation zurück, das Erreichen von Gestaltbarkeit durch das Bereitstellen von Spielräumen. Susanne Maaß schreibt in [Maaß 1994], dass es kein objektives Maximum von Transparenz für einen Arbeitskontext gibt. Maximale Transparenz kann bedeuten, dass ein Benutzer alle Anwendungs- und Handhabungsfunktionen kennt, die von den Entwicklern für seinen Arbeitskontext vorgesehen wurden, und er alle vorgesehenen Spielräume nutzen kann. Systeme werden aber in völlig unvorhergesehener Weise genutzt. Ein bekanntes Beispiel hierfür ist die Nutzung von Tabellenkalkulationen im Rahmen der Textverarbeitung zum Aufbereiten von textuellen Tabellen, ohne dass irgendetwas berechnet wird.

Dies ist ein wichtiger Punkt, wenn man sich Gestaltbarkeit und damit einen Kontrolltransfer von den Entwicklern hin zu den Benutzern zum Ziel setzt. Entwickler können nicht alle Anwendungssituationen vorwegnehmen. Insofern

müssen Benutzer das System „zweckentfremdet" nutzen können, also in einer von den Entwicklern nicht explizit vorhergesehenen Weise.

Dabei stellt sich die Frage, wie man als Entwickler aktiv Spielräume schaffen kann, deren Nutzung man nicht explizit vorhersieht. Das von Susanne Maaß angeführte Beispiel der Tabellenkalkulationen möchte ich dabei als Negativ-Beispiel heranziehen. Der Spielraum ist nicht aktiv geschaffen worden, sondern zufällig entstanden. Es ist darüber hinaus nur schwer vorstellbar, dass es neben der genannten Zweckentfremdung noch weitere sinnvolle, nicht vorhergesehene Einsatzmöglichkeiten gibt. Der Spielraum ist in diesem Fall also nicht besonders groß und begründet sich darauf, dass lediglich ein sehr kleiner Teil der vorhandenen Funktionalität genutzt wird.

Möchte man Spielräume aktiv schaffen, sind im Prinzip zwei Ansätze vorstellbar. Zum einen können die Entwickler die verschiedenen Möglichkeiten explizit bereitstellen. Dies erfordert aber wiederum ein Vorwegnehmen der Anwendungssituation, um geeignete Möglichkeiten zu definieren. Eine andere Möglichkeit ist die Kombination von orthogonalen Konzepten. Durch das Kombinieren verschiedener Konzepte ergibt sich dann ein sehr großer möglicher Lösungsraum. Ein Beispiel sind imperative Programmiersprachen. Durch die Kombination der wenigen Konzepte *Variablen*, *prozedurale Abstraktion*, *Schleifen* und *Fallunterscheidung*, erschließt man den gesamten Spielraum der Berechenbarkeit.

Diese Art der Anpassbarkeit (kombinatorische Anpassbarkeit) möchte ich wegen ihrer Bedeutung für Gestaltbarkeit in dieser Arbeit näher untersuchen. Bevor ich jedoch hierauf näher eingehe, möchte ich auf den Stand der Forschung in Bezug auf Orientierung eingehen.

Kapitel 4: Orientierung: Aufbau mentaler Modelle

Der Aufbau mentaler Modelle über Software, oder einfacher ausgedrückt das Verstehen von Software, hat bei flexibler oder anpassbarer Software eine vielschichtige Bedeutung. Einerseits ist nach dem Kontrollbegriff von Troy (siehe Abschnitt 2.4) das Verstehen Voraussetzung für ein gezieltes Beeinflussen und damit auch für ein *Gestalten*. Der Anwender muss die Handhabung und Präsentation bzw. die Funktionalität so weit verstanden haben, dass er über eine Anpassung nachdenken kann. Dabei stellt sich die Frage, wo in der Anwendung er die Anpassung durchführt, und wie sich die Anpassung an dieser Stelle auf den restlichen Arbeitsablauf auswirkt. Schließlich muss der Anwender auch ein zweckmäßiges mentales Modell vom Anpassungssubsystem haben, um die Anpassungen auch nach seinen Wünschen durchführen zu können.

In diesem Kapitel werde ich deshalb detailliert mentale Modelle einführen und Möglichkeiten vorstellen, ihren Aufbau zu unterstützen. Abschließend möchte ich auf zwei wesentliche Perspektiven auf ein Anwendungssystem eingehen: die der Anwender und die der Entwickler.

4.1 Zum Modellbegriff

Ein zentraler Begriff dieser Arbeit ist das *Modell*. Aus diesem Grund möchte ich ihn an dieser Stelle ausführlich einführen. Zunächst gehe ich auf grundlegende Merkmale des Modellbegriffs ein und anschließend auf den Modellbegriff selbst.

4.1.1 Grundlegende Merkmale von Modellen

Herbert Stachowiak beschreibt in [Stachowiak 1973] drei wesentliche Merkmale von Modellen:

- **Das Abbildungsmerkmal**

  Modelle sind stets Modelle von etwas, nämlich Abbildungen, Repräsentationen natürlicher oder künstlicher Originale, die selbst wieder Modelle sein können.

- **Das Verkürzungsmerkmal**

  Modelle erfassen im Allgemeinen nicht alle Attribute des durch sie repräsentierten Originals, sondern nur solche, die den jeweiligen Modelleigenschaften und/oder Modellbenutzern relevant erscheinen. Neben diesen übergangenen Originalattributen (auch *präterierte Attribute* genannt), kann es auch noch Attribute in einem Modell geben, die keine Entsprechungen im Original haben (*abundante Attribute* oder *informelle Zutaten*). Es kann unterschiedliche Modelle zum gleichen Original geben.

- **Das pragmatische Merkmal**

  Modelle sind ihren Originalen nicht per se eindeutig zugeordnet. Sie erfüllen ihre Ersetzungsfunktion

  a) für bestimmte modellbenutzende (erkennende und/oder handelnde) Subjekte

  b) innerhalb bestimmter Zeitintervalle

c) unter Einschränkung auf bestimmte gedankliche oder tatsächliche Operationen

Herbert Stachowiak weist ausdrücklich darauf hin, dass Modelle immer Modelle von *etwas* für *jemanden* und für einen bestimmten *Zweck* sind. Sie erfüllen ihre Funktion i*n der Zeit*, innerhalb eines Zeitintervalls. Eine pragmatische Bestimmung des Modellbegriffs hat nicht nur die Frage zu berücksichtigen, *wovon* etwas Modell ist, sondern auch, *für wen*, *wann* und *wozu* bezüglich seiner spezifischen Funktionen es Modell ist.

Stephan Dutke bemerkt in [Dutke 1994], dass ein Modell unterschiedlichen Zwecken dienen kann: Auf allgemeinster Ebene kann es als erkenntnisgewinnend und kommunikativ bezeichnet werden. Simuliert man z.B. unterschiedliche Herstellungsprozesse eines bestimmten Produkts in einem Rechnermodell, kann dies helfen, langfristige Kosten-Nutzen-Verhältnisse unterschiedlicher Produktionsmethoden zu vergleichen. Die mit der Planung eines solchen Prozesses befasste Person gewinnt so Erkenntnisse, ohne die Produktionsmethoden tatsächlich zu erproben. Gleichzeitig dient das Rechnermodell aber auch kommunikativen Zwecken: Durch die Vorführung der Prozesssimulation kann anderen Kollegen Wissen über die Eigenschaften des Prozesses vermittelt werden. Anders formuliert: Das (Rechner-)Modell ist dazu gestaltet worden, mentale Modelle zu erzeugen.

Die Wissensgewinnung oder -vermittlung besteht in beiden Fällen in der Übertragung von Attributen des Modells auf das Original (im Gegensatz zur Modellkonstruktion, bei der Attribute des Originals im Modell abgebildet werden). Der Transfer kann umso fehlerfreier gelingen, je stärker Modell und Original einander angeglichen sind. Herbert Stachowiak unterscheidet dabei in [Stachowiak 1973] zwei Typen von Modell-Original-Angleichungen: Strukturell-formale und Material-inhaltliche.

4.1.2 Der Modellbegriff

Herbert Stachowiak führt in [Stachowiak 1973] Modelle auf „attributive Systeme" zurück. Die „Erstellungsmittel" dieser Systeme unterteilt er in Individuen und Attribute (dieser Individuen). Diese Zweiteilung ist nicht grundlegender Art, sondern nur eine Konzession an die Sprachlogik, die sich auch zweiteilt in Nomen und Verben bzw. Subjekte und Prädikate.

- **Individuen**

  Individuen existieren nicht per se, sondern können in einem anderen Zusammenhang Attribut-Funktion erfüllen. Sie werden also willkürlich festgelegt. Individuen sind wohl unterscheidbare, nicht weiter zerlegbare Teilobjekte. Sie sind die Träger von Eigenschaften, ihnen kann ggf. eine Relationenstruktur aufgeprägt werden. Herbert Stachowiak bezeichnet sie auch als „Attribute nullter Stufe" bzw. „uneigentliche Attribute".

- **Attribute**

  Attribute sind entweder Eigenschaften der Individuen oder Relationen zwischen Individuen (Attribute erster Stufe). Darüber

hinaus sind auch Eigenschaften von Eigenschaften oder Eigenschaften von Relationen denkbar (Attribute zweiter Stufe), usw.

Über diese Definition hinaus benötigen Modelle als Konstruktionsgebilde oder technisch herzustellende Objekte zusätzlich zu der zustandsmäßigen Beschreibung *Konstruktions-* und *Herstellungsanweisungen*.

Diese Definition stimmt weitestgehend mit der überein, die Heinz Züllighoven als Begriffsgerüst zur Modellierung in [Züllighoven et al. 1998] beschreibt. Lediglich die Benennungen unterscheiden sich. Individuen bezeichnet er als Elemente und Relationen als Beziehungen. Die Herstellungsanweisungen lassen sich in den Bildungsgesetzen wiedererkennen. Ich werde zur Erläuterung der theoretischen Aspekte die Bezeichnungen von Herbert Stachowiak verwenden, um näher am Original zu bleiben. Im späteren Verlauf der Arbeit werde ich dann aufgrund der besseren „Griffigkeit" auf die Bezeichnungen von Heinz Züllighoven zurückgreifen.

4.1.3 Analogie als Beziehung zwischen Modellen

Nach dieser Definition des Modellbegriffs wird das Original mittels einer Abbildungsfunktion auf das Modell abgebildet. Es ist nahe liegend, nach wichtigen Abbildungen und Klassen von Abbildungen zu suchen, um spezifischere Aussagen treffen zu können.

Für den nachfolgenden Abschnitt über mentale Modelle ist die *Analogie* als Beziehung zwischen Original und Modell von besonderer Bedeutung. Bei der Analogie gleichen die Relationen zwischen den Elementen des Basisbereichs zumindest teilweise den Relationen zwischen den Elementen des Zielbereichs. Zwischen den Elementen an sich muss keine besondere Beziehung stehen. Damit unterscheidet sich die Analogie von der *Ähnlichkeits*beziehung. Dort wird eine zumindest partielle Identität der Eigenschaften der Elemente selbst vorausgesetzt. Als Beispiel einer Analogie wird in [Dutke 1994] Sonnensystem und Atom genannt.

Im Sonnensystem kreisen die Planeten um die Sonne. In einem Atom kreisen Elektronen um einen Atomkern. Sonne und Atomkern, sowie Planeten und Elektronen sind zwar in ihren Eigenschaften sehr verschieden, die Relation „kreist umeinander" und damit die Struktur sind aber gleich. Beide Modelle sind *analog*. Für mentale Modelle merkt Stephan Dutke jedoch an, dass eine Ähnlichkeit der Elemente das Erkennen einer Analogiebeziehung erleichtert.

4.2 Mentale Modelle

Unter dem Etikett „mentale Modelle" lassen sich eine Reihe unterschiedlicher Ansätze zu einer gemeinsamen Perspektive auf problemlösendes Denken zusammenfassen. Mit „Theorien", „naiven Theorien" und „intuitiven Theorien" seien nur einige wenige Ansätze benannt, die von Stephan Dutke in [Dutke 1994] aufgegriffen werden. Obwohl viele Ansätze gegenseitige Bezüge aufweisen, sind nur isolierte Aspekte belegt. Im Zentrum steht dabei eine ganzheitliche Betrachtung der menschlichen Informationsverarbeitung im Gegensatz zu der bisher häufig anzutreffenden Konzentration auf isolierte Phänomene. Hier einfließende und bislang isoliert betrachtete Prozesse sind: analoges Denken,

Gedächtnisorganisation, Vorstellung, deduktives Schließen, Urteilen unter Unsicherheit, Handlungsregulation, usw.

Der ganzheitliche Ansatz und das Zusammenfließen zahlreicher Ansätze lassen die Theorie der mentalen Modelle geeignet erscheinen, als Grundlage für die Untersuchung von Orientierung in Anwendungssystemen zu dienen.

4.2.1 Der Begriff „mentales Modell"

Stephan Dutke umschreibt mentale Modelle folgendermaßen:

> ***Mentale Modelle (aus [Dutke 1994])***
>
> *Mentale Modelle sind Ausdruck des Verstehens eines Ausschnitts der realen Welt. Damit sind sie aber gleichzeitig auch Grundlage zur Planung und Steuerung von Handlungen. Individuelle mentale Modelle können ihre eigenen Schwerpunkte aufweisen: manche sind stärker verstehensorientiert, andere eher handlungsorientiert.*

Darüber hinaus nennt er zwei mögliche „Quellen" für mentale Modelle. Sie können von außen angeregt werden, wie z.B. in einem Lehrbuch, das die Entwicklung mentaler Modelle beim Leser zum Ziel hat. Oder aber sie entstehen aus eigenem Antrieb, unabhängig von äußeren Vorgaben. Menschen organisieren Alltagswissen in Form gedanklicher Modelle, um sich das Verstehen oder Behalten bestimmter Sachverhalte zu erleichtern.

Stephan Dutke weist ausdrücklich darauf hin, dass es sich bei mentalen Modellen um hypothetische Konstrukte handelt, die sich der unmittelbaren Beobachtung ent-ziehen. Sie dienen als Erklärung menschlicher Informationsverarbeitungsleistungen und werden als kognitive Konstrukte aufgefasst, die auf einer Interaktion von Wahrnehmung und Gedächtnis beruhen. Wichtig ist dabei, dass sie von den Zielen und Absichten des Informationsverarbeiters und damit auch indirekt von der Aufgabe abhängen. Aufgrund ihres hypothetischen Charakters bedürfen sie einer Validierung durch beobachtbares Verhalten. Im folgenden Abschnitt möchte ich deshalb die zentralen Merkmale mentaler Modelle vorstellen, die in [Dutke 1994] genannt werden.

4.2.2 Merkmale mentaler Modelle

Nachdem ich den Begriff *mentale Modelle* allgemein vorgestellt habe, möchte nun auf die Merkmale und Funktionen im Detail eingehen. Stephan Dutke zählt in [Dutke 1994] folgende Merkmale und Funktionen auf:

- **Mentale Modelle sind hypothetische Konstrukte.**
- **Mentale Modelle sind reduzierende und elaborierende Abbildungen.**
- **Mentale Modelle dienen unterschiedlichen Funktionen.**
- **Nützliche mentale Modelle sind schwer zu verändern.**
- **Mentale Modelle basieren auf Analogien.**

- **Analogien erfordern schematisches Wissen.**
- **Mentale Modelle sind Aktivierungen von Schemata.**
- **Mentale Modelle dienen dem dynamischen Simulieren.**
- **Mentale Modelle sind anschaulich.**
- **Mentale Modelle sind keine Repräsentationstheorien.**
- **Mentale Modelle sind transitorische Produkte der Vorstellung.**

Mentale Modelle sind hypothetische Konstrukte.

Mentale Modelle sind hypothetische Konstrukte, mit denen Leistungen menschlicher Informationsverarbeitung beschrieben und erklärt werden sollen. Eigenschaften und Funktionen mentaler Modelle können daher nur indirekt aus der experimentellen oder simulationsbasierten Analyse menschlicher Informationsverarbeitung erschlossen werden.

Mentale Modelle sind reduzierende und elaborierende Abbildungen.

Es werden nicht alle Merkmale eines Originals in einem inneren Modell abgebildet. Die Menge der Merkmale und Relationen wird verkürzt. Welche Merkmale abgebildet werden, hängt vom Vorwissen der Person ab und von der Funktion des mentalen Modells bzw. von den Intentionen des Modellierers. Das mentale Modell kann jedoch auch gegenüber dem Original, gegenüber der subjektiven Wahrnehmung des Originals oder gegenüber Mitteilungen über das Original zusätzliche Merkmale enthalten.

Mentale Modelle dienen unterschiedlichen Funktionen.

Die beiden wichtigsten Funktionen sind, das Verstehen von Sachverhalten der Umwelt zu ermöglichen und eine Grundlage zur Planung und Steuerung von Handlungen bereitzustellen. Je nach Funktionsschwerpunkt werden Modelle über unterschiedliche Aspekte des gleichen Gegenstands gebildet. Ein mentales Modell zur Steuerung von Routinehandlungen ist besonders wirksam, wenn es schematische Beziehungen zwischen Umweltzuständen, Handlungen und Handlungsfolgen in der Umwelt enthält, wenn es also von Einzelereignissen abstrahiert. Ein mentales Modell zum Verstehen eines unbekannten Sachverhalts ist besonders wirksam, wenn es die Probleminhalte der spezifischen Situation abbildet und nicht bei der Verarbeitung schematischen Wissens stehen bleibt.

Nützliche mentale Modelle sind schwer zu verändern.

Ist ein mentales Modell gebildet worden, das einer bestimmten Funktion gerecht wird, ist dieses Modell nur schwer zu verändern, selbst wenn es eine Gegebenheit der Umwelt objektiv falsch oder unzureichend abbildet. Auch unzutreffende mentale Modelle können unter eingeschränkten Bedingungen nützlich sein. Ihre Korrektur ist mit kognitivem Aufwand verbunden, dem. u.U. kein unmittelbarer Nutzen gegenübersteht. Die Qualität eines mentalen Modells bemisst sich subjektiv nur zu einem geringeren Anteil an seiner objektiven Korrektheit und zu einem größeren Teil an seiner Nützlichkeit zur Erreichung von individuellen Zielen.

Mentale Modelle basieren auf Analogien.

Mentale Modelle zum Verstehen neuer Sachverhalte basieren häufig auf Analogien. Eine Analogie ist ein Spezialfall eines Modells, bei der nur Relationen zwischen Elementen eines Basisbereichs (bereits gespeichertes Wissen) auf die Elemente eines neuen Zielbereichs übertragen werden. Der Erkenntnisgewinn beruht auf der Übertragung der Relationen, nicht auf Ähnlichkeiten der Elemente selbst.

Analogien erfordern schematisches Wissen.

Das Erkennen oder Konstruieren einer Analogie erfordert schematisches Wissen. Die Vergleichbarkeit der Relationen im Basis- und im Zielbereich können nur dann konstruiert bzw. erkannt werden, wenn es ein oder mehrere langzeitlich gespeicherte *Gedächtnisschemata* gibt, aus denen sowohl die Relationen im Basisbereich als auch die im Zielbereich abgeleitet werden können. Dazu muss dieses Schema abstrakter sein als die Relationen im Basis- und im Zielbereich. Eine hohe Ähnlichkeit der Elemente im Basis- und im Zielbereich begünstigt die Aktivierung eines geeigneten Schemas.

Den Begriff (*Gedächtnis-)Schema* werde ich im nächsten Abschnitt detailliert einführen.

Mentale Modelle sind Aktivierungen von Schemata.

Ein mentales Modell zum Verstehen eines neuen Sachverhalts ist eine Aktivierung[7] eines oder mehrerer Schemata. Bei der Konstruktion eines mentalen Modells zum Verstehen eines neuen Sachverhalts werden die Leerstellen eines Schemas durch die Gegebenheiten einer spezifischen Situation ausgefüllt. Damit werden die abstrakten Relationen eines Schemas auf dieses spezifische Modell übertragen. Die Folge ist eine Anreicherung des Modells durch Kontextwissen, das Bestandteil des Schemas ist. Das Ableiten neuer Folgerungen wird dadurch erleichtert oder ermöglicht, dass die abstrakten Relationen durch Beispiele vertrauter, vorstellbarer Sachverhalte repräsentiert sind. Das mentale Modell ist also konkreter als die zugrunde liegende schematische Wissensbasis.

Mentale Modelle dienen dem dynamischen Simulieren.

In einem mentalen Modell können Sachverhalte der Umwelt dynamisch simuliert werden. Gedankliches Probehandeln oder Durchspielen von Ereignisfolgen kann das mentale Modell so verändern, dass neue Modellzustände vorher nicht bekannte Zusammenhänge und Folgerungen repräsentieren. Das Ergebnis der Simulation kann aus dem Endzustand des Modells abgelesen werden. Dieser kann auf die Wissensbasis des Modells zurückwirken und Gedächtnisschemata verändern. Da die kognitive Simulation aus der Manipulation konkreter Gedächtnisinhalte in der Vorstellung besteht, bedarf sie nicht unbedingt der

7 Stephan Dutke verwendet im Original das Wort „Instantiierung". Dies ist jedoch kein deutsches Wort im engeren Sinne. Der Duden ([Duden 2000]) verzeichnet lediglich das Wort „Instanz", jedoch mit der hier nicht passenden Bedeutung „zuständige Stelle". Stephan Dukte bezieht sich auf das englische Wort „to instance", was im Bereich der objektorientierten Programmierung die Erzeugung eines Objekts einer Klasse bezeichnet. Ich werde deshalb im weiteren den von ihm alternativ eingeführten Begriff „Aktivierung" bzw. „aktivieren" verwenden. Ein mentales Modell ist eine Aktivierung eines oder mehrerer Schemata.

Anwendung formaler Schlussfolgerungsregeln.

Mentale Modelle sind anschaulich.

Anschaulichkeit bedeutet nicht zwangsläufig „Bildhaftigkeit", obwohl dies in vielen Fällen zutreffen mag. Denn Vorstellungsbilder sind Sichtweisen auf ein Modell in einem bestimmten Zustand. An einem mentalen Modell können jedoch auch Repräsentationen anderer Sinnesmodalitäten beteiligt sein, sodass „anschaulich" im Sinne von „vorstellbar" zu verstehen ist.

Mentale Modelle sind keine Repräsentationstheorien.

Theorien mentaler Modelle sind keine Repräsentationstheorien im engeren Sinne. Sie erheben keinen Anspruch darauf, dass mentale Modelle unmittelbar in der neurologischen Struktur des Gehirns repräsentiert werden. Mentale Modelle sind zumindest teilweise analoge Repräsentationen, ihre Wissensbasis kann propositional oder anderweitig symbolisch repräsentiert sein. Es muss nicht zwangsläufig angenommen werden, dass mentale Modelle eine eigenständige und gegenüber analogen Repräsentationen abgrenzbare Repräsentationsform darstellen.

Mentale Modelle sind transitorische Produkte der Vorstellung.

Mentale Modelle werden auf der Grundlage abstrakten, schematischen Wissens zu einem bestimmten Zweck gebildet. Es ist deshalb auch aus Gründen der Speicherökonomie nicht anzunehmen, dass mentale Modelle Einheiten der langzeitlichen Speicherung sind. Dafür spricht auch die begründete Hypothese, dass die Art der Repräsentation eines mentalen Modells von seinem Verwendungszweck, also von den situativen Anforderungen sowie den Kompetenzen und Intentionen der Person abhängt. Die Bildung mentaler Modelle könnte man auch als Heuristik bezeichnen, die immer dann von Nutzen ist, wenn Anforderungen an die Informationsverarbeitung gestellt werden, für die entweder nur unzureichende informationelle Grundlagen vorliegen (z.B. bei Verständnisproblemen) oder bei denen die Nutzung derselben unökonomisch wäre (z.B. bei Routinehandlungen).

4.2.3 Analogie, Schema und mentales Modell

Orientierung ist Voraussetzung für Gestaltbarkeit. Aus diesem Grund ist es eine zentrale Frage dieser Arbeit, wie Orientierung erreicht wird, d.h. wie werden (geeignete) mentale Modelle aufgebaut? Stephan Dutke nennt in [Dutke 1994] das Erkennen einer Analogiebeziehung als Voraussetzung für den Aufbau eines mentalen Modells. Dabei erfordert die Entdeckung einer Analogiebeziehung i.d.R. eine Abstraktion von den konkreten Beispielen. Analogien werden mithilfe schematischen Wissens entdeckt und konstruiert. I.d.R. findet nur eine teilweise Überlappung der Relationen statt, d.h. es gibt Relationen, die nicht übertragbar sind. Trotz dieser Unterschiedlichkeit leidet die Analogie nicht, sie führt trotzdem zu einem Erkenntnisgewinn. Dies ist erklärungsbedürftig.

Um schematisches Wissen bzw. die Organisation von Wissen im Gedächtnis zu beschreiben, gibt es den Begriff des (Gedächtnis-)Schemas. Genau wie bei mentalen Modellen handelt es sich bei ihnen um hypothetische Konstrukte, die

Phänomene im Zusammenhang mit der Gedächtnisorganisation von Menschen beschreiben. Insofern definieren sie sich - analog zu mentalen Modellen - über ihre Merkmale.

Stephan Dutke beschreibt einen Versuch von F.C. Bartlett ([Bartlett 1932]), der wichtige Eigenschaften von Schemata verdeutlicht. Versuchspersonen bekamen eine Geschichte zu lesen. Diese Geschichte wurde so verfasst, dass der Text die Zusammenhänge möglichst undeutlich beschrieb, dass der Text wenig kohärent war und dass die Geschichte in einer wenig vertrauten kulturellen und sozialen Umgebung spielte. Die Versuchspersonen wurden aufgefordert, diese Geschichte nach einiger Zeit aus dem Gedächtnis wiederzugeben. Dabei fielen zwei Phänomene auf: *Rationalisierung* und *Elaborierung*.

Rationalisierung insofern, als dass der Text auf eine Art und Weise verändert wiedergegeben wurde, die eine für den Leser nachvollziehbare Interpretation ermöglicht. Das Gedächtnismaterial wird also anscheinend reduziert und dabei in das bereits bestehende Wissen der Person eingefügt.

Elaborierung insofern, als dass beim Wiedergeben der Geschichte Sachverhalte ungewollt um Dinge ergänzt wurden, die im Original nicht enthalten waren. Daraus folgt, dass Erinnerungen in einen Zusammenhang eingebunden werden, der ihnen Sinn verleiht.

Auf Basis dieser beiden Phänomene wird gefolgert, dass Erinnern und Wiedergeben *konstruktive* Prozesse sind. Das erfordert aber, dass das Gedächtnismaterial in strukturierter Form vorliegt, als Einheiten, die typische Zusammenhänge enthalten, anhand derer dann die Reproduktion aufgebaut werden kann. Diese Einheiten werden (Gedächtnis-)Schemata genannt.

Schema ist einer der zentralen Begriffe dieser Arbeit, von dem ich im nächsten Kapitel den Begriff *Konzept* ableiten werde. Aus diesem Grund möchte die wesentlichen Merkmale von Schemata vorstellen (siehe [Dutke 1994]):

- **Schemata sind Strukturen allgemeinen Wissens**

  Schemata sind Strukturen allgemeinen Wissens, die typische Zusammenhänge eines Realitätsbereichs enthalten.

- **Schemata sind Abstraktionen konkreter Erfahrung**

  Schemata bilden sich durch die Abstraktion konkreter Erfahrungen.

- **Schemata enthalten Leerstellen**

  Schemata enthalten *Leerstellen*, die die Relationen zu bestimmten Kategorien angeben („Daten”, „Suchkriterien”). Diese werden je nach konkreter Situation ergänzt (Aktivierung). Die Ausfüllung einer Leerstelle bestimmt, welche konkreten Werte in einer anderen Leerstelle als sinnvoll akzeptiert werden. Genau diese Einschränkungen von Aktivierungen stellen den Informationsgehalt eines Schemas dar.

- **Schemata enthalten Voreinstellungen für Leerstellen**

  Nicht immer stehen Informationen über die konkreten Ausprägungen aller Leerstellen zur Verfügung. Deshalb enthalten Schemata Informationen über *typische Merkmalsausprägungen*

(Voreinstellungen).

- **Schemata sind hierarchisch verschachtelt**

  Schemata werden auf unterschiedlichen Abstraktionsebenen gebildet und sind hierarchisch verschachtelt. Eine zunehmende Differenzierung und Unterscheidbarkeit von Schemata ist ein Kennzeichen wachsender Expertise in einer Wissensdomäne. Wichtig dabei ist, dass allgemeine Schemata schwieriger zu ändern sind als spezifische.

- **Schemata unterliegen Veränderungen**

  Schemata sind nicht statisch, sondern unterliegen selbst der ständigen Veränderung. Einerseits leiten Gedächtnisschemata Wahrnehmung und Erinnerung (Assimilation), andererseits können neue Erfahrungen bewährte Schemata modifizieren (Akkomodation).

- **Schemata beeinflussen kognitive Leistungen**

  Gedächtnisschemata beeinflussen darüber hinaus die unterschiedlichsten kognitiven Leistungen, z.B. lenken sie elementare Wahrnehmungsprozesse und das Verstehen von Ereignisabfolgen in sozialen Situationen.

Laut [Dutke 1994] sind Schemata die Grundlage für die Konstruktion mentaler Modelle. Abbildung 9 zeigt eine Übersicht der dort beschriebenen Zusammenhänge. Ein mentales Modell für einen neuen Sachverhalt wird auf Basis einer erkannten Analogie zwischen einem bekannten Sachverhalten A (Basisbereich) und einem neuen Sachverhalt B (Zielbereich) aufgebaut. Das Erkennen einer Analogie setzt voraus, dass ein Gedächtnisschema aktiviert werden kann, aus dem sowohl der Sachverhalt des Zielbereichs als auch der Sachverhalt des Basisbereichs abgeleitet werden kann (mentale Modelle). Beide Sachverhalte müssen also Aktivierungen des gleichen Schemas sein, ansonsten wird die Bildung des mentalen Modells für den neuen Sachverhalt B verhindert.

Für eine Nichtaktivierung kann die Ursache darin liegen, dass entweder kein relevantes Gedächtnisschema existiert oder aber dass die Hinweisreize in der entsprechenden Situation nicht für eine Aktivierung geeignet sind.

Stephan Dutke fasst den Unterschied zwischen mentalen Modellen und Gedächtnisschemata folgendermaßen zusammen: „Mentale Modelle werden auf Basis schematischen Wissens konstruiert und sind selbst konkrete Instantiierungen (Aktivierungen) eines oder mehrerer Schemata.”

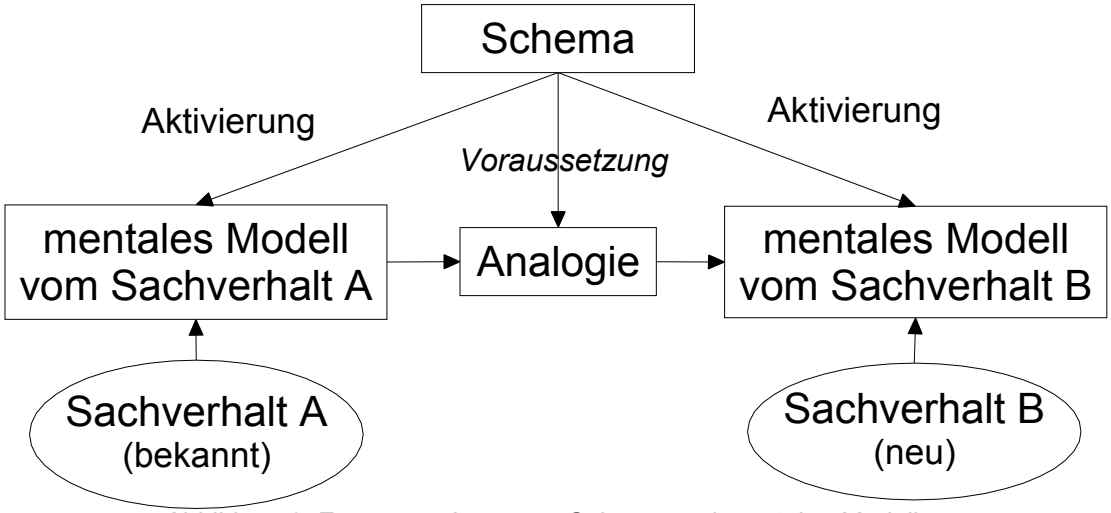

Abbildung 9: Zusammenhang von Schema und mentalen Modellen

Im Zusammenhang mit dem Verstehen von Software weist Susanne Maaß in [Maaß 1994] darauf hin, dass eine transparente Anwendungssituation nur dann zustande kommt, wenn sich der Anwender ein *zweckmäßiges mentales Modell* gebildet hat. Ob ein mentales Modell zweckmäßig ist, hängt wesentlich von dem individuellen Benutzer und der jeweiligen Aufgabe ab.

4.3 Den Aufbau mentaler Modelle unterstützen

Nachdem ich geschildert habe, wie mentale Modelle grundsätzlich aufgebaut werden, möchte ich mich nun Möglichkeiten zuwenden, den Aufbau von mentalen Modellen zu unterstützen. Stephan Dutke nennt als mögliche Instrumente *konzeptionelle Modelle* und *Metaphern*. Susanne Maaß nennt in [Maaß 1994] *Metaphern* und *Konsistenz*.

Darüber hinaus möchte ich Ergebnisse von John M. Carroll vorstellen, die besonders die aktive Rolle des Benutzers hervorheben und die vorgestellten Konzepte aus einer anderen Perspektive beleuchten. Daraus möchte ich ein viertes Instrument ableiten: *aktives Lernen*.

Im Folgenden möchte ich die genannten Instrumente erläutern.

4.3.1 Konzeptionelle Modelle

Als konzeptionelles Modell beschreibt Stephan Dutke in [Dutke 1994] ein vorgegebenes externes Modell, welches die Analogiebeziehung verdeutlichen soll und so den Aufbau eines mentalen Modells unterstützt. Dazu greifen konzeptuelle Modelle tendenziell Begriffe des Zielbereichs auf. Konzeptuelle Modelle werden explizit zu einem bestimmten Zweck konstruiert.

Ein Beispiel für ein konzeptuelles Modell ist die Darstellung eines Stromkreises als ein Wasserkreislauf mit einem Staudamm als Batterie, dem Gefälle als Spannungsunterschied und einem Mühlrad als Verbraucher. Konzeptionelle Modelle werden häufig grafisch dargestellt, sie sind aber nicht auf eine visuelle Darstellung beschränkt.

4.3.2 Metaphern

Viele Autoren werten Metaphern als sehr wichtiges Mittel, um Software verständlicher zu machen ([Preece et al. 1994], [Dix et al. 1998], [Beck und Andres 2004]). Susanne Maaß sieht in [Maaß 1994] als wesentliche Eigenschaft von Metaphern, dass sie eine Analogie zwischen einem vertrauten und einen unbekannten Bereich ziehen (Quelle und Ziel). Sie ermöglichen eine Übertragung vorhandenen Wissens in neue Kontexte. Metaphern erleichtern die Orientierung im neu erschlossenen Bereich, indem sie die Komplexität des neuen Bereichs auf bekannte Konzepte reduzieren, von denen ausgehend dann neues Wissen erworben werden kann.

Susanne Maaß sieht Metaphern besonders in frühen Lernstadien als lernförderlich, da sie dem Anwender durch ihre vereinfachende Darstellung eine grobe Orientierung ermöglichen. Wenn aber später das übertragene Wissen fest mit dem vorhandenen Wissen assoziiert wird, stellen sie nur noch Erinnerungshilfen dar.

Stephan Dutke weist in [Dutke 1994] auf einen Unterschied zwischen der Metapher und der Analogiebeziehung hin. Die Analogie ist eine Relation zwischen zwei Modellen, während die Metapher ein externes, didaktisches Mittel ist, um auf diese Relation hinzuweisen und sie damit zu veranschaulichen. Den wesentlichen Unterschied zwischen Metapher und konzeptionellem Modell sieht er darin, dass Metaphern auf das Wissen bekannter Gegenstandsbereiche zurückgreifen, während konzeptionelle Modelle Zusammenhänge eher in Begriffen des Zielbereichs beschreiben.

Wichtig für ihn ist die Erkennbarkeit von Metaphern. Sind sich der Basis- und der Zielbereich zu ähnlich, besteht die Gefahr einer wörtlichen Interpretation. Sind sie sich zu unähnlich, so sind sie nur schwer zu interpretieren. Besonders wirkungsvoll sind Metaphern, wenn Basis- und Zielbereich eine mittlere Ähnlichkeit haben.

Susanne Maaß ergänzt, dass eine perfekte Nachbildung der gegenständlichen Welt nicht im Sinne einer Computerunterstützung sein kann. Der Computer ermöglicht auch neue Arten der Verarbeitung, insofern kann es schwer sein, geeignete Metaphern zu finden.

4.3.3 Konsistenz

Eine allgemein anerkannte und zentrale Forderung im Sinne benutzergerechter Systemgestaltung ist *Konsistenz* ([Dix et al. 1998], [Shneiderman 1998]). Susanne Maaß definiert in [Maaß 1994] konsistente Systeme als Systeme, die sich regelhaft, gleichmäßig verhalten. Der Benutzer kann Erwartungen ausbilden und das System reagiert erwartungskonform.

Sie unterscheidet dabei *interne* Konsistenz und *externe* Konsistenz. Interne Konsistenz meint dabei Konsistenz innerhalb eines Systems. Verschiedene Systemteile verhalten sich konsistent. Externe Konsistenz meint die Konsistenz gegenüber anderen Systemen.

Wendy Kellogg ([Kellog 1987]) unterscheidet *prozedurale* und *konzeptionelle* Konsistenz. Prozedurale Konsistenz bezieht sich auf die visuelle Darstellung, wie die Struktur der Ausgabe oder die Syntax der Eingabe. Sie wird mit den heute

üblichen Styleguides adressiert (*[Dix et al. 1998]*). Konzeptionelle Konsistenz bezieht sich hingegen auf semantische Konzepte, wie z.B. der Abbildung von Arbeitszielen auf Systemobjekte und -funktionen. Wendy Kellogg bezeichnet sie auch als „interne Kohärenz der Systemstruktur". Unter anderem stellt sie fest, dass Systeme, die sowohl prozedural als auch konzeptionell konsistent sind, von Anwendern schneller gemeistert werden, als Systeme, die nur prozedural konsistent sind. Dabei muss die Konsistenz vom Benutzer auch wahrgenommen werden.

Susanne Maaß weist weiter darauf hin, dass Konsistenz auch im Sinne der Systementwickler liegt, da (formal) konsistente Systeme auch vollständig formal beschrieben werden können. Die Aufgabenangemessenheit darf von diesem Aspekt aber nicht beeinträchtigt werden.

4.3.4 Minimalist Instruction: aktives Lernen

In diesem Abschnitt möchte ich das Konzept „Minimalist Instruction" von John M. Carroll vorstellen (siehe [Carroll 1989]). Dieses Konzept unterscheidet sich von den anderen Instrumenten zum Unterstützen von Orientierung, die ich bisher in diesem Abschnitt vorgestellt habe. Im Wesentlichen handelt es sich hierbei um fünf charakteristische Typen von „Fehlverhalten" bei Anwendern. John M. Carroll hat diese allerdings als spezifische menschliche Eigenschaften beim Lernen und Schlussfolgern identifiziert. Sie bilden in dem Sinne kein einzelnes Instrument, sondern beleuchten die bisher vorgestellten Instrumente aus einer anderen Perspektive und lenken den Blick auf einen neuen Aspekt: *aktives Lernen*. Diese Eigenschaften sind das Ergebnis einer Untersuchung, in der Anwender beim Benutzen von Online-Hilfe beobachtet wurden. Ich werde die einzelnen Eigenschaften entsprechend der Förderung von Orientierung jeweils explizit einordnen.

1. People learn by doing; they try to act in order to learn.

Menschen lernen beim Ausführen; sie möchten aktiv handeln, um zu lernen. Eine Person wird nur dann lernen, wenn sie Wissen beim Anwenden selbst erfahren kann.

Einordnung: Diesen neuen Aspekt möchte ich mit *aktivem Lernen* bezeichnen. Möchte ein Anwender einen bestimmten Aspekt einer Anwendung verstehen, so muss er ihn *be-greifen* können. Er muss die Möglichkeit haben, diesen Aspekt der Anwendung zu manipulieren. Dafür müssen ihm geeignete Werkzeuge zur Verfügung gestellt werden.

2. People learn by thinking and reasoning; they generate and test hypothesis in order to learn.

Menschen lernen durch Denken und Schlussfolgern; um zu lernen, entwickeln und testen sie Hypothesen. Effizientes Lernen erfordert selbstbestimmtes Denken und Schlussfolgern. Vorgefertigte Anweisungsschritte sind dazu wenig geeignet, da sie weder aktiv noch herausfordernd genug sind.

Einordnung: Hier wird der Aspekt der Simulationsfähigkeit von

mentalen Modellen deutlich. Das Aufstellen von Hypothesen kann als eine gezielte Veränderung des mentalen Modells gesehen werden. Das Ergebnis des Testens kommt einer Bewertung dieser Veränderung gleich.

3. People seek to work in a meaningful context and towards meaningful goals.

Menschen wollen in einem relevanten Kontext auf ein sinnvolles Ziel hinarbeiten. In erster Linie veranlasst der Wunsch, eine bestimmte Aufgabe zu erledigen, den Menschen dazu, Computerwerkzeuge überhaupt erlernen zu wollen. Somit orientiert sich der Lernerfolg an einem praktisch relevanten Fortschritt. Ein großer Fehler von Handlungsanweisungen ist es, mit zu vielen Einzelschritten und zu umfangreichen Lerneinheiten das praktisch relevante Vorankommen zu behindern. Jemand, der gerade eine Online-Lernhilfe benutzte, beschwerte sich folgendermaßen: „I want to do something, not to learn how to do everything." („Ich möchte etwas Bestimmtes tun und nicht lernen, wie ich alles Mögliche tun kann.")

Einordnung: Diese Eigenschaft unterstreicht die Einordnung von Punkt 1. Das Lernen sollte dabei nicht nur anhand und mit der Anwendung selbst möglich sein. Das aktive Lernen sollte auch im aktuellen Problemkontext möglich sein, also z.B. nicht ausschließlich in extra dafür ausgewiesenen Bereichen der Anwendung, wie z.B. ausschließlich an Lernbeispielen.

4. People rely on their prior knowledge when they try to manage and assimilate new experience.

Menschen stützen sich auf ihr bereits vorhandenes Wissen, wenn sie neue Erfahrungen machen und verarbeiten. Neue Dinge in Beziehung zu bereits Bekanntem zu setzen, erleichtert es ihnen, sich zu erinnern und das neue Wissen in passenden Situationen einzusetzen.

Einordnung: Vorhandenes Wissen kann hier unterschiedlich interpretiert werden. Einerseits stellen Metaphern eine Art intuitives Wissen dar. Wissen kann aber auch als Gedächtnisschemata repräsentiert sein. Schließlich kann man Konsistenz so sehen, dass vorhandenes Wissen erfolgreich auf unbekannte Teile des Systems übertragen werden kann.

5. People use error diagnosis and recovery episodes as a means of exploring the boundaries of what they know.

Menschen benutzen Fehlererkennung und -beseitigung, um die Grenzen ihres Wissens auszuloten. Fehler spielen beim Lernen eine sehr viel größere Rolle als Probleme. Ein Fehler kann der Prüfstein für ein intellektuelles Auskundschaften sein, ein Vehikel, um zu entdecken, was bereits bekannt ist und was noch nicht. In einem gewissen Sinn sind Fehler die Vorbedingung für jede Art von Lernen. Für den Erfolg solch eines fehlerbasierten Lernens müssen Menschen erkennen können, dass sie einen Fehler gemacht haben, sie müssen folgern können, was den Fehler ausgelöst hat und wie mit ihm umgegangen

werden kann.

Einordnung: Diese Eigenschaft steht in Verbindung mit Punkt 2. Beim Ausprobieren von Hypothesen sind Fehler negative Reaktionen des Systems oder aber sie weisen auf nicht korrekte Teile von Hypothesen hin. Fehlermeldungen, aber auch Reaktionen des Systems allgemein, sollten so gestaltet werden, dass der Anwender Möglichkeiten hat, das *Warum* zu erkunden.

Ein Negativbeispiel sind ausgegraute Menüpunkte. Dem Anwender wird deutlich gemacht, dass diese Funktion nicht zur Verfügung steht. Warum diese Funktion ausgegraut ist und welche Möglichkeiten er hat, diese Funktion aktivierbar zu machen, ist für ihn nicht erkennbar.

Auf die in diesem Abschnitt vorgestellten neuen Aspekte möchte ich im Weiteren unter dem Begriff *aktives Lernen* verweisen. Man muss allerdings auch die Umstände der Entstehung berücksichtigen und dass John M. Carrolls Konzept des *Minimalist Instruction* als Gegenpol zu langen textuellen Beschreibungen von Anwendungssystemen zu sehen sind. Der Idee des „Lernens durch Manipulieren" kommt in dieser Arbeit eine fundamentale Bedeutung zu. Insbesondere muss eine Anwendung, die diese Idee umsetzen will, für jede (durch den Anwender) erdenkliche Hypothese eine korrekte Antwort geben. Korrekt heißt in diesem Zusammenhang, dass sie korrekte Hypothesen über das System bestätigen und bei falschen Hypothesen nachvollziehbare Fehler produzieren muss.

4.3.5 Zusammenfassung und Bewertung

Zum Abschluss dieses Abschnitts möchte ich die vorgestellten Instrumente bewerten und in Beziehung zueinander setzen. Als wichtigstes Instrument für diese Arbeit sehe ich konzeptionelle Modelle. Insbesondere gilt dies für ihren Modellcharakter. Konzeptionelle Modelle bestehen für mich aus Elementen mit Eigenschaften, die miteinander in Beziehung stehen. Dabei ist es wesentlich für den Modellnutzer, dass er die Elemente und ihre Beziehungen untereinander erkennen kann. Wichtig ist ebenfalls, dass konzeptionelle Modelle in Begriffen des Zielbereichs formuliert sind. Zusammen mit dem Modellcharakter ergibt sich als Vorteil gegenüber Metaphern, dass konzeptionelle Modelle sind bedeutungsschärfer sind.

Trotzdem ist es möglich, Metaphern in konzeptionellen Modellen aufzugreifen. Das zu vermittelnde Konzept, die Elemente und die Beziehungen können mit Metaphern bezeichnet werden. So lässt sich die Bedeutungsschärfe von konzeptionellen Modellen mit dem intuitiven, aber unscharfen Verstehen von Metaphern kombinieren.

Konsistenz ist ein weiteres wichtiges Instrument, um den Aufbau mentaler Modelle zu unterstützen. Man könnte sie als Regelhaftigkeiten des Systems beschreiben, die auf das Herausbilden von Gedächtnisschemata abzielen. Zwei verschiedene Teile des Systems können dann als konsistent betrachtet werden, wenn sie sich analog bedienen lassen bzw. analog verhalten. Der Anwender muss jedoch diese Konsistenz wahrnehmen, damit sie Orientierung unterstützen kann.

Aktives Lernen greift schließlich den Modellcharakter von konzeptionellen

Modellen wieder auf. Die Elemente, Eigenschaften und Beziehungen eines konzeptionellen Modells sollten nicht nur erkennbar, sondern auch erkundbar und manipulierbar sein. Entsprechende Werkzeuge zum *be-greifen* des konzeptionellen Modells sollten angeboten werden. Dabei werden die vorher genannten Instrumente auch in die Nähe der Anwendung selbst gerückt. Das Erkunden konzeptioneller Modell ist nicht etwas, das außerhalb der eigentlichen Anwendung passiert. Vielmehr sollten das konzeptionelle Modell und die Werkzeuge in die Anwendung selbst integriert sein.

4.4 Konzeptionelle Modelle von Software

Nachdem ich im vorhergehenden Abschnitt konzeptionelle Modelle als eines der wichtigsten Instrumente für den Aufbau mentaler Modelle identifiziert habe, möchte ich in diesem Abschnitt einen Blick auf existierende konzeptionelle Modelle über Software werfen. Dabei ist mir insbesondere an zwei verschiedenen Perspektiven auf ein Anwendungssystem gelegen: die der Anwender und die der Entwickler.

Diese beiden Perspektiven möchte ich anschließend genauer betrachten. Für die Perspektive der Entwickler möchte ich den Begriff *Architektur* am Beispiel der WAM-Modellarchitektur betrachten. Stellvertretend für konzeptionelle Modelle für den Anwender möchte ich das *Object-Action Interface Model* von Ben Shneiderman vorstellen.

4.4.1 Perspektiven auf Software

Donald A. Norman unterscheidet in [Norman 1986] zwei mentale und ein physisches Modell in Bezug auf Anwendungssysteme. Abbildung 10 zeigt eine Übersicht.

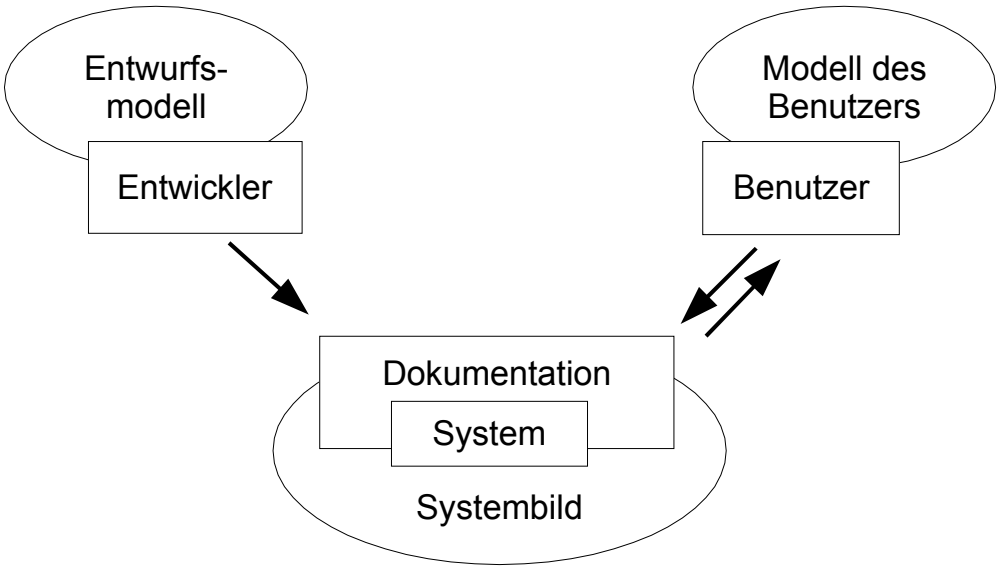

Abbildung 10: mentale und physische Modelle eines Anwendungssystems (nach [Norman 1986])

Das *Entwurfsmodell* (im Original: *Design Model*) ist ein Modell des Entwicklers über das zu entwickelnde Anwendungssystem. Im Idealfall basiert es auf den vom

Anwender zu erledigenden Aufgaben, seinen Anforderungen und Fähigkeiten. Der Entwickler muss dabei den Hintergrund, die Erfahrung und die Leistungsfähigkeit des Anwenders berücksichtigen. Leistungsfähigkeit meint in diesem Zusammenhang die Grenzen der menschlichen Informationsverarbeitung und die Grenzen des Kurzzeitgedächtnisses. Auf Grundlage dieses Modells erzeugt der Entwickler das Anwendungssystem.

Das *Modell des Benutzers* über das Anwendungssystem (im Original: *User's Model*) bezeichnet das mentale Modell, das sich der Anwender über das Anwendungssystem bildet. Donald A. Norman weist darauf hin, dass als Grundlage nicht das Entwurfsmodell dient, sondern dass es auf Grundlage des *Systembilds* vom Anwender konstruiert wird.

Das Systembild umfasst dabei sowohl die erzeugte physische Struktur (das Anwendungssystem) als auch bekannte Dokumentationen und Anleitungen.

Die Bedeutung der Pfeile wird in [Norman 1986] nicht explizit erläutert. Sie lassen sich allerdings folgendermaßen interpretieren: Das Entwurfsmodell ist die Grundlage, auf der das System entwickelt bzw. das Systembild erzeugt wird. Das Systembild ist wiederum Grundlage für das Modell des Benutzers. Der Pfeil zurück unterstreicht dabei, dass das Modell Grundlage für Erwartungen an das System und die Interaktion mit dem System ist. Die Interaktion mit dem System beeinflusst wiederum das Modell des Benutzers.

Nach Donald A. Norman sollte es die dringendste Aufgabe des Entwicklers sein, ein geeignetes Systembild zu erzeugen. Dabei trägt jede Form der Interaktion dazu bei, dieses Bild zu formen: physische Knöpfe, Wählscheiben, Tastaturen und Anzeigen. Hinzu kommen Dokumentation, Anleitungen, Hilfe-Möglichkeiten, Ein- und Ausgabe von Texten, sowie Fehlermeldungen. Das Ziel des Entwicklers sollte dabei sein, das Modell des Benutzers mit dem zugrunde liegenden konzeptionellen Modell, dem Entwurfsmodell, kompatibel zu machen.

Explizit weist Donald A. Norman noch einmal darauf hin, dass diese Kompatibilität nur durch Interaktion des Anwenders mit dem Systembild entstehen kann. Dazu muss das Systembild explizit, klar und konsistent sein. Dies ist die Voraussetzung dafür, dass der Anwender das System verstehen, geeignet verwenden und das Verwenden „genießen" kann (im Original: *to enjoy using it*)[8]. Dabei ist zu berücksichtigen, dass Anwender nicht immer die Dokumentation lesen, sodass die Hauptlast auf dem Bild liegt, das das System selbst von sich vermittelt.

Das *Modell des Benutzers* über das Anwendungssystem (im Original: *User's Model*) ist laut Donald A. Norman eine Vereinfachung. Dieses Modell adressiert im Grunde zwei verschiedene Dinge: das persönliche Modell des individuellen Nutzers und das verallgemeinerte Modell eines „typischen Benutzers".

Das *Entwurfsmodell* und das *Modell des Benutzers* sind nach Donald A. Norman mentale Modelle. Um sich von den vielen verschiedenen Bedeutungen abzugrenzen, bezeichnet er sie als konzeptionelle Modelle. Er führt diesen Begriff und seine Bedeutung nicht explizit ein. Im Abschnitt 4.3.1 habe ich diese Begriffe bereits genau definiert. Wenden wir diese Definitionen auf das Modell von Donald A. Norman an, so kommen wir zu einer leicht unterschiedlichen, aber trotzdem sinnvollen Sichtweise (siehe Abbildung 11). Den mentalen Modellen wird jeweils

8 Dies ist eine der Kernforderungen aus [Norman 1986]: „I want a system that is enjoyable to use." (Ich möchte ein System, dessen Verwendung man genießen kann.)

ein externes konzeptionelles Modell gegenübergestellt. Diese lassen sich wie folgt interpretieren.

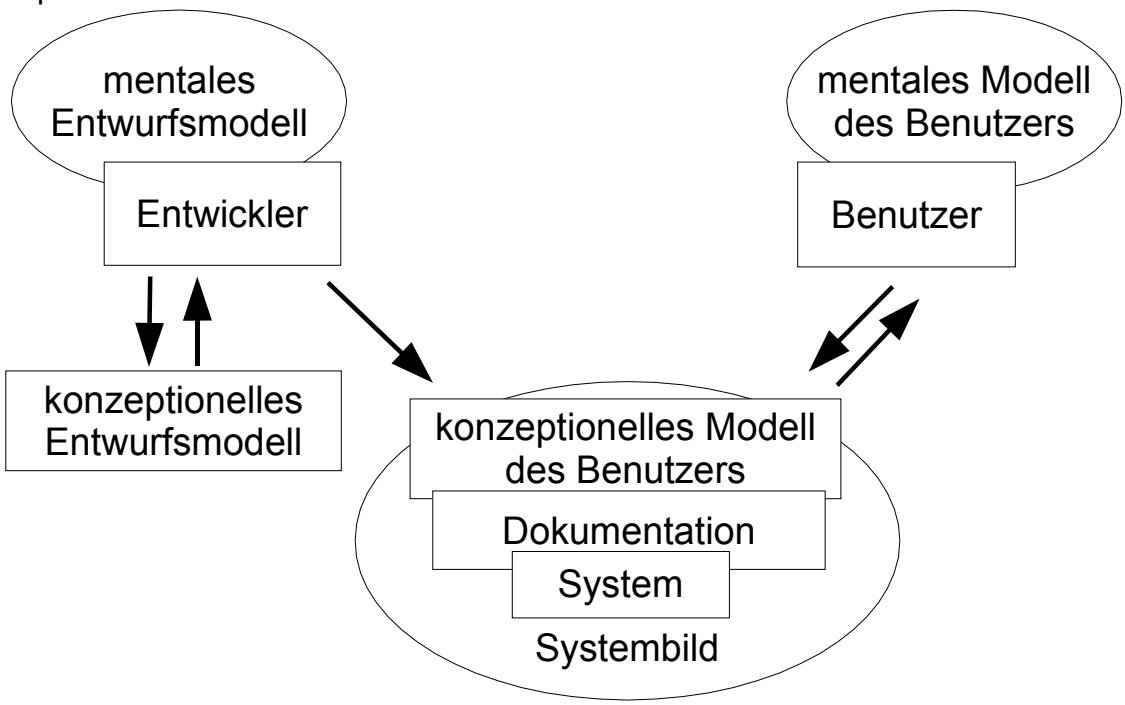

Abbildung 11: mentale und konzeptionelle Modelle eines Anwendungssystems (in Anlehnung an [Norman 1986])

Das *konzeptionelle Entwurfsmodell* ist ein externes Modell mit dem Ziel, das Ausbilden eines geeigneten *mentalen Entwurfsmodells* (von dem zu entwickelnden Anwendungssystem) bei dem Entwickler zu unterstützen. Dies klingt zuerst abwegig, da der Entwickler das System ja selbst entwickelt. Sein mentales Modell des Anwendungssystems sollte eigentlich Ausgangspunkt für ein konzeptionelles Modell sein. Dass dies nicht uneingeschränkt gilt, werde ich im nächsten Abschnitt erläutern.

Das *konzeptionelle Modell des Benutzers* ist dann ein externes Modell mit dem Ziel, das Ausbilden eines geeigneten mentalen Modells vom Anwendungssystem bei dem Benutzer zu unterstützen. Dies erscheint sinnvoll. Ein konzeptionelles Modell vom Anwendungssystem gehört damit zum Systembild. Da der Begriff sehr länglich und relativ unhandlich ist, möchte ich im Weiteren das *konzeptionelle Modell des Benutzers* kurz als *Benutzungsmodell* (in Anlehnung an [Züllighoven et al. 1998]) bezeichnen.

In den nächsten beiden Abschnitten möchte ich auf diese beiden konzeptionellen Modelle eingehen und Beispiele vorstellen, die sich als *konzeptionelles Entwurfsmodell* und als *Benutzungsmodell* identifizieren lassen.

4.4.2 Softwarearchitektur und Modellarchitektur

In Bezug auf das mentale Modell des Benutzers hat Donald A. Norman bereits darauf hingewiesen, dass die beschriebene Sicht vereinfachend ist. Es handelt sich eigentlich um ein Modell des *typischen* Benutzers und stellt somit eine Verallgemeinerung verschiedener individueller Modelle dar.

Betrachtet man das mentale Entwurfsmodell genauer, ist die Situation ähnlich.

Projekte zur Erstellung von Individualsoftware werden i.d.R. mit Teams aus mehreren Entwicklern durchgeführt. Jeder Entwickler realisiert dabei verschiedene Teile des Gesamtsystems. Für ein Systemteil beschreibt die Abbildung 10 also annähernd die Realität, nicht jedoch für das Gesamtsystem. Aber auch für ein einzelnes Systemteil ist der jeweilige Entwickler nicht völlig frei in der Konstruktion.

Damit ein großes Anwendungssystem über längere Zeit erweiterbar, verständlich und damit beherrschbar bleibt, muss eine bestimmte innere Qualität sichergestellt werden. Die einzelnen softwaretechnischen Artefakte dürfen deshalb nicht beliebig konstruiert und miteinander verknüpft werden, sondern müssen gewissen Rahmenbedingungen genügen, z.B. dem Geheimnisprinzip (siehe auch [Floyd und Züllighoven 1999]). Diese Randbedingungen bei der Konstruktion werden im Allgemeinen als *Softwarearchitektur* bezeichnet.

Der Begriff Softwarearchitektur wird häufig mit unterschiedlichen Bedeutungen verwendet, deshalb möchte ich an dieser Stelle die Definition aus [Züllighoven et al. 1998] vorstellen.

> ***Softwarearchitektur***
>
> *Eine Softwarearchitektur bezeichnet die Modelle und die konkreten Komponenten eines Softwaresystems in ihrem statischen und dynamischen Zusammenspiel. Sie kann selbst als explizites Modell dargestellt werden.*
>
> *Eine Softwarearchitektur beschreibt ein konkretes System in seinem Anwendungskontext.*

Bezeichnet man den programmiersprachlichen Code eines Anwendungssystems als *Implementationsmodell*, so stellt eine Softwarearchitektur eine Abstraktion über das Implementationsmodell, d.h. ein Model des Implementationsmodells dar. Das Ziel dieses Modells ist eine übersichtliche und verständliche Beschreibung des kompletten Anwendungssystems oder zumindest größerer Teile davon. Mit dem Begriff der Softwarearchitektur hängt der Begriff der *Modellarchitektur* zusammen. Auch hier möchte ich auf die Definition aus [Züllighoven et al. 1998] zurückgreifen.

> ***Modellarchitektur***
>
> *Eine Modellarchitektur beschreibt die allgemeinen Prinzipien hinter einer Softwarearchitektur. Sie umfasst die grundlegenden Elemente, deren Verknüpfungen und die Regeln, die für die Softwarearchitektur gelten.*
>
> *Eine Modellarchitektur gibt Anleitungen bei der softwaretechnischen Realisierung eines Softwaresystems.*

Eine Modellarchitektur dient also als Vorlage für eine konkrete Softwarearchitektur. Beide zusammen bilden das konzeptionelle Entwurfsmodell. Abbildung 12 zeigt, wie sich Modellarchitektur und Softwarearchitektur in das

Diagramm von Donald A. Norman einordnen lassen. (Die Benutzerseite ist wegen der besseren Übersichtlichkeit nicht dargestellt.)

Die Modellarchitektur dient als Vorlage bei der Realisierung von Systemteilen. Der Entwickler entscheidet auf dieser Grundlage, wie er die Softwarearchitektur zu verändern hat, d.h. welche Art von neuen Elementen er hinzufügen muss, wie diese zu den bestehenden Elementen in Beziehung stehen sollen oder wie bestehende Elemente verändert werden müssen. Daraufhin verändert er das Implementationsmodell (das System). Die Tätigkeit des Programmierens beschränkt sich also nicht nur auf reine programmiersprachliche Realisierung, sondern erfordert eine Realisierung, die konsistent zur Modellarchitektur ist. Dies ist keine triviale Forderung und erfordert aufseiten des Entwicklers genaueste Kenntnisse über die Modellarchitektur, aber auch darüber, wie sich eine konkrete Anforderung konsistent zur Modellarchitektur realisieren lässt.

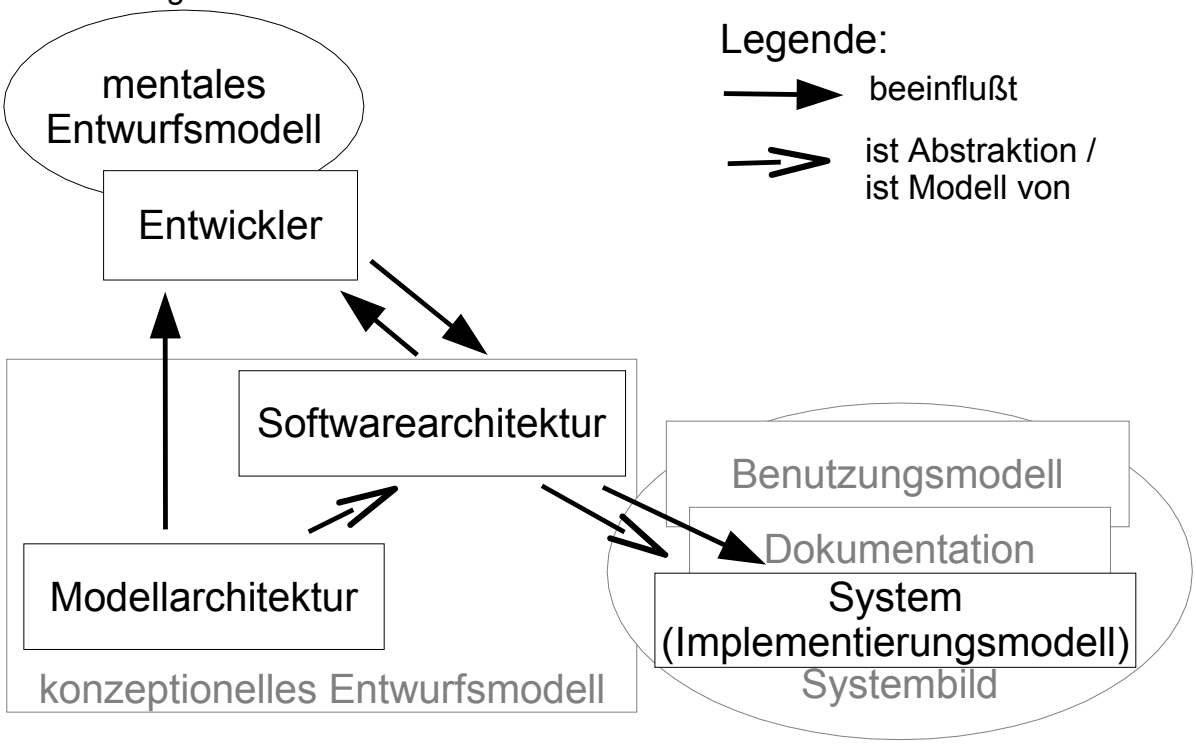

Abbildung 12: Softwarearchitektur und Modellarchitektur als konzeptionelle Entwurfsmodelle (in Anlehnung an [Norman 1986])

Erfahrungen der C1 WPS GmbH ([C1 WPS 2005]) mit dem Software-Tomographen ([Sotograph 2005]), einem Werkzeug zur Analyse des Implementationsmodells, zeigen, dass Abweichungen von der gewünschten Softwarearchitektur und auch der Modellarchitektur üblich sind.

Das Einhalten der Modellarchitektur kann durch ein *Rahmenwerk* unterstützt werden, das ihre wesentlichen Konzepte vergegenständlicht. [Weiß 1997] definiert diesen Begriff folgendermaßen:

Durch das Rahmenwerk wird das Implementationsmodell unmittelbar mit der Modellarchitektur gekoppelt. Die Modellarchitektur wird für den Entwickler unmittelbar *be-greifbar* und öffnet sich dadurch für ein *aktives Lernen* (siehe Abschnitt 4.3.4). Verletzungen der Modellarchitektur sind immer noch möglich, da nicht jeder Aspekt der Modellarchitektur im Rahmenwerk vergegenständlicht sein

> ***Rahmenwerk ([Weiß 1997])***
>
> *Ein Rahmenwerk (Framework) ist eine Architektur aus Klassenhierarchien, die eine allgemeine generische Lösung für ähnliche Probleme in einem bestimmten Kontext vorgibt.*
>
> *Wiederverwendet werden dabei nicht einzelne Klassen, sondern die gesamte Konstruktion aus zusammenspielenden Komponenten. Ein Rahmenwerk gibt so den Kontrollfluss für die Anwendung vor.*
>
> *Ein Rahmenwerk wird zu einer konkreten Anwendung, indem dafür vorgesehene Klassen spezialisiert oder vorgegebene Parameterobjekte erzeugt werden.*

kann, oder das Rahmenwerk bei der Programmierung einfach umgangen wird. Der Vorteil einer solchen softwaretechnischen Vergegenständlichung ist aber deutlich erkennbar.

Wie bereits im Abschnitt 2.3.1 erwähnt, möchte ich mich in dieser Arbeit auf WAM-Anwendungssysteme konzentrieren, d.h. Anwendungssysteme, die nach der WAM-Modellarchitektur entwickelt wurden. Zur Unterstützung der Konstruktion wurde das Rahmenwerk JWAM ([JWAM 2005]) entwickelt, das weite Teile der WAM-Modellarchitektur vergegenständlicht. JWAM wird sogar als didaktisches Mittel zum Vermitteln und zum Diskutieren der WAM-Modellarchitektur in der Lehre am Arbeitsbereich SWT eingesetzt.

Die WAM-Modellarchitektur selbst werde ich in dieser Arbeit nicht explizit vorstellen. Hierfür möchte ich auf [Züllighoven et al. 1998] verweisen. Die für diese Arbeit wesentlichen Konzepte werde ich, angepasst an die Zielsetzung dieser Arbeit, im Kapitel 6 einführen. Ein Aspekt der WAM-Modellarchitektur ist dabei für diese Arbeit von grundlegender Relevanz: *Strukturähnlichkeit*.

Strukturähnlichkeit

In [Züllighoven et al. 1998] wird sie folgendermaßen definiert:

> ***Strukturähnlichkeit***
>
> *Strukturähnlichkeit bezieht sich im WAM-Ansatz auf das Verhältnis von Software und Anwendungsbereich.*
>
> *Die softwaretechnischen Komponenten eines Anwendungssystems modellieren die relevanten Konzepte und Gegenstände des Anwendungsbereichs.*
>
> *Die Architektur des Anwendungssystems spiegelt die wesentlichen Bezüge zwischen den Konzepten und Gegenständen des Anwendungsbereichs wider.*

Heinz Züllighoven nennt zwei entscheidende Vorteile von *Strukturähnlichkeit:* Die Anwender finden die Gegenstände ihrer Arbeit und die Begriffe ihrer Fachsprache im Anwendungssystem repräsentiert. Sie können ihre Arbeit in gewohnter Weise organisieren. Die Entwickler können Softwarekomponenten und

Anwendungskonzepte bei fachlichen Änderungen zueinander in Beziehung setzen und somit wechselseitige Abhängigkeiten erkennen.

Strukturähnlichkeit ist jedoch auf bestimmte Aspekte beschränkt. Im Wesentlichen umfasst sie das Begriffsmodell (siehe [Züllighoven et al. 1998]), d.h. die relevanten Arbeitsgegenstände des Anwendungsbereichs. Trotzdem hilft Strukturähnlichkeit dabei, Anwendungen zu konstruieren, die von Anwendern mit wenig Aufwand zu erlernen sind, falls sie mit dem Anwendungsbereich vertraut sind.

4.4.3 Zum Begriff Benutzungsmodell

Ein Benutzungsmodell habe ich bisher nicht explizit gefordert. Vielmehr erschien es automatisch, nachdem ich die Definitionen von mentalem und konzeptionellem Modell auf die Grafik von Donald A. Norman angewandt habe. Dieses „natürliche" Auftauchen legt die Verwendung eines explizit erstellten Benutzungsmodells für die Unterstützung von Orientierung in einer Anwendung nahe.

Der Begriff *Benutzungsmodell* wird sowohl von Horst Oberquelle ([Oberquelle 1998]) als auch Heinz Züllighoven (in [Züllighoven et al. 1998]) aufgegriffen. Obwohl ich diese Definitionen in dieser Arbeit nicht explizit aufgreifen möchte, stelle ich sie der Vollständigkeit halber vor.

Benutzungsmodell (aus [Züllighoven et al. 1998])

Ein Benutzungsmodell ist ein fachlich orientiertes Modell darüber, wie Anwendungssoftware bei der Erledigung der anstehenden Aufgaben im jeweiligen Einsatzkontext benutzt werden kann.

Das Benutzungsmodell umfasst eine Vorstellung von der Handhabung und Präsentation der Software aber auch von den fachlichen Gegenständen, Konzepten und Abläufen, die von der Software unterstützt werden.

Es ist sinnvoll, ein Benutzungsmodell auf der Grundlage eines Leitbilds mit Entwurfsmetaphern zu realisieren.

Benutzungsmodell (aus [Oberquelle 1998])

„Benutzungsmodelle = konzeptuelle Modelle für Benutzer

Ebene 1:

Konzepte (anwendungsbezogen, Fachsprache berücksichtigt), Orte, Materialien, Operationen, Werkzeuge, Steuerungen

Ebene 2:

Realisierung, Darstellung auf dem Bildschirm, Hilfsvorstellungen (z.B. eine fachliche „Lupe" ist technisch als Fenster realisiert) Realisierung von Werkzeugen"

Beide Definitionen sind durchaus vergleichbar. Heinz Züllighoven verzichtet darauf, explizit zwei Ebenen zu trennen. Die von ihm genannten Punkte lassen sich aber problemlos auf die Ebenen von Horst Oberquelle abbilden: Handhabung und Präsentation, sowie Entwurfsmetaphern lassen sich der Ebene 2 zuordnen. Fachliche Gegenstände, Konzepte und Abläufe finden sich auf der Ebene 1 wieder. Horst Oberquelle sieht darüber hinaus Benutzungsmodelle explizit als konzeptionelle Modelle. Die in diesem Kapitel ad hoc eingeführte Interpretation des Begriffs Benutzungsmodell geht somit mit den genannten Definitionen konform.

Ein Benutzungsmodell macht somit im Sinne eines *konzeptionellen Modells des Benutzers* explizit, was der Anwender vom Anwendungssystem wissen muss, um es sinnvoll nutzen zu können. Dazu muss das Benutzungsmodell vielschichtig sein. Es muss über Details der Anwendung ebenso Auskunft geben können, wie über allgemeine Zusammenhänge. An dieser Stelle möchte ich ein Beispiel für ein Benutzungsmodell vorstellen, das Object-Action Interface Model.

4.4.4 Das Object-Action Interface-Model

Mit dem Object-Action Interaction (OAI) Model beschreibt Ben Shneiderman in [Shneiderman 1998] ein konzeptionelles Modell, mit dem in Online-Hilfen Anwendungssysteme im Ganzen beschrieben werden können. Im engeren Sinne ist es kein *Benutzungsmodell*, wie ich es eben beschrieben habe. Vielmehr handelt es sich um eine Vorlage für konkrete Benutzungsmodelle, ich möchte es als *Benutzungsmodell-Rahmen* bezeichnen. Trotzdem, oder gerade deswegen möchte ich es deshalb an dieser Stelle als Beispiel vorstellen und diskutieren.

Das OAI-Modell beschreibt eine Anwendung auf Basis von vier Hierarchien. Zwei Bereiche, der fachliche Aufgabenbereich und der Oberflächenbereich, bestehen aus jeweils zwei Hierarchien: eine objekt- und eine aktionsbezogene. Diese Hierarchien beschreiben die Objekte bzw. Aktionen für den jeweiligen Bereich auf verschiedenen Detaillierungsstufen. Abbildung 13 zeigt eine Übersicht.

Shneiderman merkt dabei an, dass Hierarchien nicht perfekt, aber trotzdem verständlich und nützlich sind. Außerdem sei es eine natürliche Herangehensweise, Dinge oder Probleme schrittweise in kleinere zu zerlegen.

Sowohl für den Designprozess, als auch für ein Verständnis des Anwendungssystems schlägt Ben Shneiderman folgende Reihenfolge vor:

1. Objekte des Aufgabenbereichs
2. Aktionen des Aufgabenbereichs
3. Objekte der Oberfläche
4. Aktionen der Oberfläche

Auf diese Weise ermöglicht das OAI-Modell, eine Anwendung strukturiert und im Ganzen zu beschreiben. Der Anwender kann diese Beschreibung so nachvollziehen, dass es seinen Lernprozess unterstützt. Eine besondere Rolle spielt dabei die Verwendung von Hierarchien. Sie ermöglichen es dem Anwender, die Abstraktionsstufe zu wählen, die ihm geeignet erscheint, und er kann bei Bedarf detailliertere Informationen abfragen.

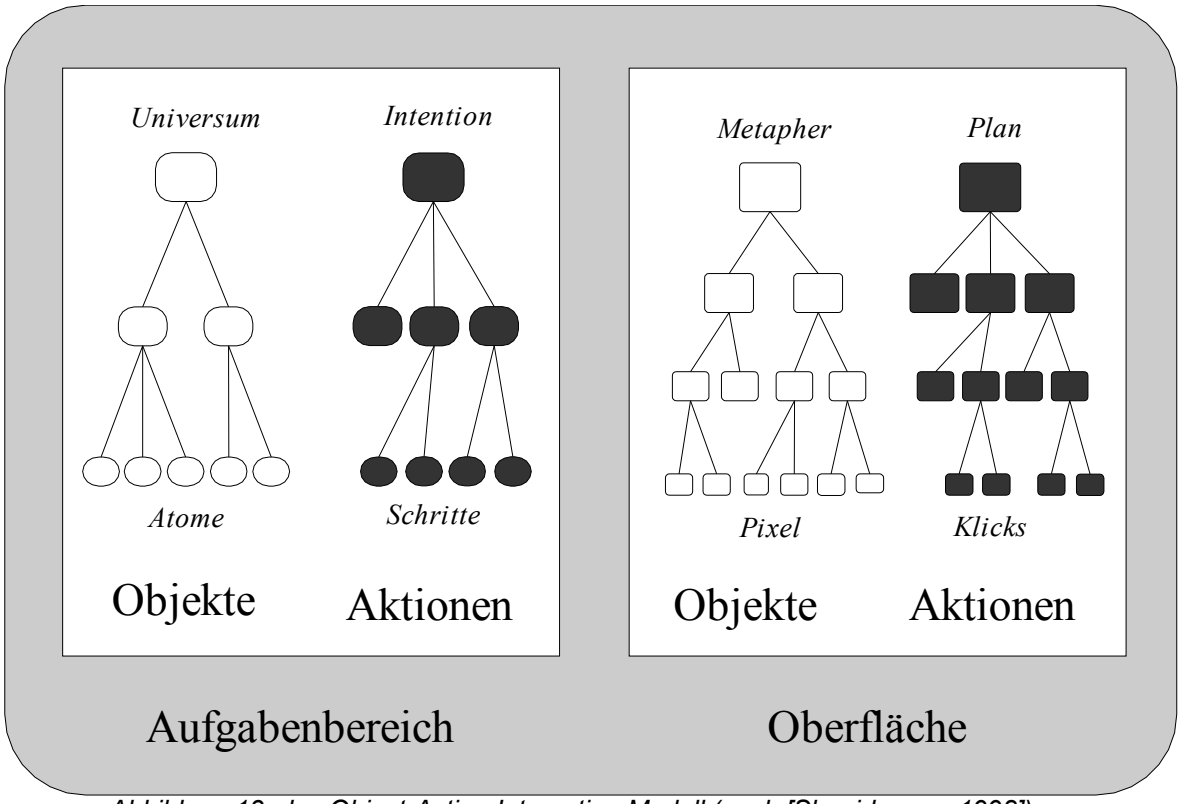

Abbildung 13: das Object-Action Interaction Modell (nach [Shneiderman 1998])

Bewertung

Positiv hervorzuheben ist, dass dieses konzeptionelle Modell ein Anwendungssystem im Ganzen und in allen wesentlichen Aspekten beschreiben kann. Hervorzuheben ist auch die Verwendung von Hierarchien.

Es gibt allerdings auch einige Punkte zu kritisieren. Zuerst sind die fehlenden Querbezüge zu nennen. Die vier Hierarchien stehen im Wesentlichen unabhängig nebeneinander. Inwiefern ein Objekt der Oberfläche mit einem Objekt des Aufgabenbereichs zusammenhängt, ist aus dem konzeptionellen Modell nicht unmittelbar ersichtlich. Durch vier voneinander unabhängige Hierarchien wird das konzeptionelle Modell auch relativ kompliziert.

Außerdem schein es nur eine Art von Beziehung zu geben, die als Grundlage für Hierarchien verwendet wird: die Komposition. Insbesondere in objektorientierten Anwendungen kommt jedoch noch mindestens eine hierarchische Beziehung hinzu: die Vererbungs- bzw. die Subtyp-Beziehung. Im Sinne einer Taxonomie kann auch das Wissen über diese Hierarchien für den Anwender sinnvoll sein. Denkbar wären auch noch weitere Beziehungen zwischen den Elementen des Anwendungssystems, die für ein Verständnis wichtig wären.

4.4.5 Probleme beim Erstellen des Benutzungsmodells

Nachdem ich definiert habe, was ich unter einem Benutzungsmodell verstehe, und ein Beispiel für einen Benutzungsmodell-Rahmen vorgestellt habe, möchte ich nun auf einige Probleme bei der Erstellung eingehen.

Zuerst ist die *Form* völlig offen. Ben Shneidermans OAI-Modell zielt auf ein

Benutzungsmodell in Form einer artikelbasierten Online-Hilfe, dies muss aber nicht die einzige oder die beste Möglichkeit sein. Offen ist auch, wie der Inhalt konkret aussehen soll. Was genau ist in einem konkreten Benutzungsmodell enthalten? Auch im WAM-Ansatz gibt es keinen Dokumenttyp „Benutzungsmodell".

Darüber hinaus ist unklar, welche Person das Erstellen des Benutzungsmodells übernimmt. Online-Hilfen werden i.d.R. von speziell ausgebildeten Personen erstellt. In Individualsoftware-Projekten liegt diese Tätigkeit oft ganz außerhalb der finanziellen Möglichkeiten. Häufig wird deshalb auf die Erstellung einer Online-Hilfe ganz verzichtet. Insofern verbleibt lediglich die Rolle des Entwicklers als Ersteller des Benutzungsmodells. Dies ist insofern von Vorteil, als dass dies die Konsistenz von Benutzungsmodell und Anwendung unterstützt. Problematisch ist jedoch, dass, wie im vorhergehenden Abschnitt angesprochen, bereits die technische Realisierung des Systems eine nicht zu unterschätzende Herausforderung für Entwickler darstellt.

Darüber hinaus sollten Systemteile so programmiert werden, dass ihr Benutzungsmodell konsistent zum Benutzungsmodell der anderen Systemteile ist. Dies ist keine triviale Forderung, insbesondere nicht für größere Softwareprojekte mit vielen Entwicklern.

Möchte man auf das Instrument des *aktiven Lernens* zurückgreifen (siehe Abschnitt 4.3.4), so muss das Benutzungsmodell in maschinenlesbarer Form vorliegen. Erst dann können Werkzeuge darauf zugreifen, um es für den Anwender *be-greifbar* zu machen. Dies präsentiert sich als ein schwieriges Problem, da sich der Anwender beliebige Hypothesen ausdenken kann, auf die das Benutzungsmodell mithilfe der Werkzeuge die richtige und zum Verhalten des Systems konsistente Antwort erzeugen muss.

Diese Probleme verstärken sich weiter, wenn man die zentrale Forderung dieser Arbeit nach *Gestaltbarkeit* berücksichtigt. Das Benutzungsmodell muss darüber Auskunft geben können, welche Gegenstände der Anwendung anpassbar sind und wie und mit welchen Werkzeugen sie angepasst werden können. Außerdem sollte es auch für diese Werkzeuge selbst wieder Benutzungsmodelle geben.

Problematisch ist auch die Forderung nach *nicht-vorweggenommenen Spielräumen* (siehe Abschnitt 3.5). Bei der Erstellung eines Benutzungsmodells muss der Zweck, d.h. die Arbeitssituation und die Aufgabenstellung, bekannt sein. Für diese Art von Spielräumen können somit keine Benutzungsmodelle explizit erstellt werden, da dies wiederum ein Vorwegnehmen bedeuten würde.

Ein Anliegen dieser Arbeit ist es, einen Vorschlag zur Lösung dieser Probleme zu erarbeiten. Diesen Vorschlag werde ich im nächsten Kapitel vorstellen. Die WAM-Modellarchitektur mit dem Konzept der Strukturähnlichkeit ist dabei ein wesentlicher Aspekt.

4.5 Zusammenfassung

In diesem Kapitel habe ich vorgestellt, was aktuell in der Literatur unter dem Begriff mentale Modelle verstanden wird und wie ihr Aufbau unterstützt werden kann.

Zuerst bin ich auf den allgemeinen Modellbegriff eingegangen. Die drei

wesentlichen Merkmale von Modellen sind das *Abbildungsmerkmal*, das *Verkürzungsmerkmal* und das *pragmatische Merkmal*. Modelle bilden Originale auf Modelle ab. Dabei werden nicht alle Attribute übertragen, die Abbildung ist verkürzend. Das pragmatische Merkmal besagt, dass Modelle immer Modelle für handelnde Subjekte, für einen bestimmten Zeitraum und nur für einen bestimmten Zweck sind. Modelle sind immer Modelle von *etwas* für *jemanden* und für einen bestimmten *Zweck*. Sie bestehen dabei aus *Individuen* und *Attributen*. Attribute sind dabei entweder Relationen zwischen Individuen oder Eigenschaften von Individuen. Eine wichtige Abbildung zwischen Modellen ist dabei die *Analogie*. Zwei Modelle sind dann analog, wenn ihre Relationen im Wesentlichen gleich sind. Die Eigenschaften der Individuen sind dabei zu vernachlässigen.

Nach dem Modellbegriff habe ich mich dem Begriff *mentale Modelle* zugewendet. Unter diesem Begriff lassen sich eine Reihe an Ansätzen zusammenfassen, denen aber eines gemein ist: eine ganzheitliche Sichtweise, die die menschliche Informationsverarbeitung nicht auf einzelne begrenzte Phänomene reduziert. Mentale Modelle sind hypothetisch und lassen sich deshalb sinnvollerweise durch ihre Merkmale beschreiben, von denen ich an dieser Stelle nur die wichtigsten wiederhole. Mentale Modelle bilden Sachverhalte reduzierend und elaborierend ab. Sie basieren auf Analogien, deren Erkennen wiederum schematisches Wissen voraussetzt. Mentale Modelle sind deshalb Aktivierungen eines oder mehrerer Schemata. Sie dienen dem dynamischen Simulieren und sind dabei transitorische Produkte der Vorstellung.

Gedächtnischemata sind somit die „Vorlagen" für die Erzeugung von mentalen Modellen und repräsentieren einen Teil des Langzeitgedächtnisses. Insofern kommt ihnen eine Schlüsselrolle zu, die ich im nächsten Kapitel näher erläutern werde. Anzumerken ist allerdings, dass auch Gedächtnischemata noch nicht eindeutig belegt sind. Zusammen mit den mentalen Modellen liefern sie allerdings eine stringente Theorie menschlichen Lernverhaltens, die beobachtbare Phänomene nachvollziehbar erklärt.

Die Motivation für dieses Kapitel ist die Frage, wie Orientierung bei dem Anwender von Anwendungssystemen unterstützt werden kann, d.h. wie sich der Aufbau zweckmäßiger mentaler Modelle beim Anwender unterstützen lässt. Hier habe ich vier Instrumente vorgestellt: *konzeptionelle Modelle*, *Metaphern*, *Konsistenz* und *aktives Lernen*.

Diese vier Instrumente stehen dabei nicht unabhängig nebeneinander. *Konzeptionelle Modelle* und *Metaphern* unterscheiden sind im Wesentlichen dadurch, dass konzeptionelle Modelle neue Sachverhalte in Begriffen des Zielbereichs der Analogie ausdrücken, während Metaphern auf den bekannten Sachverhalt in Begriffen des Quellbereichs (dem bekannten Sachverhalt) zurückgreifen. Die vom Quellbereich in den Zielbereich übertragenen Attribute werden bei Metaphern intuitiv vom Lernenden ausgewählt. Insofern sind konzeptionelle Modelle bedeutungsschärfer, da ihre Attribute extern und damit explizit formuliert wurden.

Konsistenz beschreibt grundlegende Regelhaftigkeiten eines Anwendungssystems. Ausgebildete Erwartungen können auf unbekannte Systemteile übertragen werden. Hier gibt es eine Verbindung zu den Gedächtnisschemata, die jedem mentalen Modell zugrunde liegen. Konsistenz

beschreibt sozusagen Schemata im Anwendungssystem. Andersherum sind schematische Analogien in Anwendungssystemen mit Konsistenz gleichzusetzen.

Aktives Lernen rückt die anderen genannten Instrumente in die Nähe der Anwendung, vielleicht sogar in die Anwendung selbst. Anwender wollen beim Lernen aktiv handeln. Dazu sollten die konzeptionellen Modelle in der Anwendung selbst erfahrbar und *be-greifbar* sein. So kann ein Anwender Hypothesen über das System formulieren und Ergebnisse auswerten. Ein wichtiger Aspekt sind dabei nachvollziehbare Fehlermeldungen. Eine Anwendung sollte für diese Zwecke geeignete Werkzeuge anbieten.

Abschließend habe ich zwei Perspektiven auf Software vorgestellt: die *Entwickler-* und die *Benutzer-Perspektive*. Die wichtigste Aussage dieses Abschnitts ist, dass die Vorstellungen der Entwickler über die Verwendung der Anwendung ausschließlich über das Systembild dem Anwender vermittelt wird. Dabei wird das Systembild im Wesentlichen vom Anwendungssystem selbst geprägt.

Wie ich ausgeführt habe, sind die Entwickler nur eingeschränkt in der Lage, das Anwendungssystem in dieser Weise als „Kommunikationskanal" zu nutzen. Vielmehr sind sie bereits mit der technischen Konstruktion des Systems ausgelastet. Hierfür greifen sie allerdings selbst auf konzeptionelle Modelle zurück: die *Softwarearchitektur* und die *Modellarchitektur*. Auch hier gilt es, das Implementierungsmodell, die Softwarearchitektur und die Modellarchitektur konsistent zu halten. Ein *Rahmenwerk* kann die Konstruktion mit diesen konzeptionellen Modellen und das Einhalten von Konsistenz wesentlich unterstützen. Die WAM-Modellarchitektur, auf die ich mich in dieser Arbeit beziehen möchte, weist dabei ein wichtiges Merkmal auf: *Strukturähnlichkeit*. Sie stellt eine Verbindung zwischen Anwendungsbereich und Softwarearchitektur her.

Die Perspektive des Benutzers sollte durch ein *Benutzungsmodell* geprägt sein. Ich habe verschiedene Definitionen dieses Begriffs vorgestellt. Wesentlich für diese Arbeit ist jedoch der Charakter als *konzeptionelles Modell des Benutzers* von der Anwendung, also eine externe Vorlage für den Aufbau eines mentalen Modells. Als ein Beispiel für ein Benutzungsmodell habe ich das Object-Action Interaction Model von Ben Shneiderman vorgestellt. Genau betrachtet ist es kein Beispiel, sondern eine Vorlage für konkrete Benutzungsmodelle. Dies habe ich als *Benutzungsmodell-Rahmen* bezeichnet.

Zum Abschluss dieses Kapitels habe ich Probleme aufgelistet, die ich bei der Erstellung eines Benutzungsmodells sehe. Diese Probleme werde ich im nächsten Kapitel aufgreifen und einen Lösungsvorschlag erarbeiten.

Kapitel 5: Konzepte in Anwendungen

In diese Arbeit formulier ich einen Vorschlag, wie der in Kapitel 2 beschriebene Kontrollverlust für den Anwender von Softwaresystemen zur Unterstützung von Büroarbeit möglichst verringert werden kann. Insbesondere soll dabei Gestaltbarkeit für den Anwender im Hinblick auf seinen eigenen Arbeitsprozess ermöglicht werden. Die von mir im Kapitel 2 zuletzt vorgestellte zweistufige Definition von Kontrolle setzt der Gestaltbarkeit Verständnis voraus.

Gestaltbarkeit kann durch anpassbare Software umgesetzt werden. Im Kapitel 3 habe ich Dimensionen der Anpassbarkeit vorgestellt. Dabei nehmen die Gegenstände der Anpassung und der Geltungsbereich von Anpassungen eine zentrale Stellung ein. Wichtig für die weiteren Betrachtungen ist das Schaffen von nicht explizit vorweggenommenen Spielräumen. Gleichzeitig ist dies eine besondere Herausforderung für die Software-Entwickler.

Im Kapitel 4 habe ich den Begriff des mentalen Modells eingeführt, der in dieser Arbeit den Aspekt des Verstehens von Anwendungen durch den Anwender präzisieren soll. Ich habe verschiedene Instrumente vorgestellt, die den Aufbau geeigneter Modelle unterstützen können. Abschließend habe ich in diesem Kapitel die Perspektiven von Entwicklern und Benutzern auf ein Anwendungssystem gezeigt. Das Anwendungssystem selbst ist dabei die wesentliche Grundlage für den Aufbau mentaler Modelle.

Damit der Anwender die von der Anwendung angebotenen Gestaltungsspielräume wirklich nutzen kann, muss er verschiedene Dinge verstehen. Er muss die anpassbaren Gegenstände der Anwendungen erkennen können. Darüber hinaus muss er die Anwendung soweit verstanden haben, damit er die Auswirkungen der Anpassung dieser Gegenstände einschätzen kann (Vorhersehbarkeit). Um von Gestaltbarkeit sprechen zu können, muss der Anwender auch den umgekehrten Weg gehen können: Welchen Gegenstand muss ich wie anpassen, um ein bestimmtes Ziel zu erreichen?

5.1 Die Lösungsidee: Konzepte und konzeptionelle Modelle

Die Grundidee dieser Arbeit ist, dass die beiden eigentlich unterschiedlichen Forderungen *Orientierung* und *Gestaltbarkeit* von Anwendungen mit einem ähnlichen Mechanismus unterstützt werden können und dass es dabei sogar zu Synergie-Effekten kommen kann.

Ein weiterer wichtiger Aspekt ist, dass der Lösungsvorschlag praktikabel sein soll. Er soll auch auf Individualsoftware-Projekte anwendbar sein, denen im Allgemeinen nur eingeschränkte finanzielle Mittel zur Verfügung stehen.

Die Idee ist, die allgemeinen Prinzipien eines Benutzungsmodells, den Benutzungsmodell-Rahmen, softwaretechnisch in einem Rahmenwerk zu vergegenständlichen. Das Softwaresystem wird dann mit diesem Rahmenwerk konstruiert. Ausgewählte Teile des Systems selbst stellen dann das Benutzungsmodell dar. Auf diese Weise wird ein Benutzungsmodell bei der Konstruktion eines Systemteils automatisch mitkonstruiert. Das Rahmenwerk kann bereits konkrete Werkzeuge enthalten, mit denen der Anwender auf Basis der Benutzungsmodelle die Zusammenhänge der Anwendung erkunden und grundlegende Manipulationen ausführen kann. Zu diesen grundlegenden

Manipulationen können auch Anpassungen zählen. So wird aktives Lernen der allgemeinen Prinzipien durch be-greifen unterstützt.

Die Grundlage für diese Idee sind die in Abschnitt 4.4 vorgestellten konzeptionellen Modelle. Das Verhältnis von Modellarchitektur und konkreter Softwarearchitektur erscheint mir analog zu dem Verhältnis von Benutzungsmodell-Rahmen und Benutzungsmodell zu sein. Deshalb sollten die Vorteile eines Rahmenwerkes auch auf die Erstellung eines Benutzungsmodells übertragbar sein. In solch einem Rahmenwerk wären die allgemeinen Prinzipien des Benutzungsmodell-Rahmens vergegenständlicht.

Betrachtet man speziell WAM-Anwendungen, so kommt eine Besonderheit der WAM-Modellarchitektur hinzu: Strukturähnlichkeit. Sie stellt eine direkte Verbindung zwischen der Softwarearchitektur und dem Anwendungsbereich her. Wie der Anwendungsbereich in der Anwendung repräsentiert ist, ist jedoch auch wesentlich für die Benutzung der Anwendung und sollte deshalb Teil des Benutzungsmodells sein. Es gibt bei WAM-Anwendungen also eine direkte Verbindung zwischen Softwarearchitektur und Benutzungsmodell.

Diese Verbindung betrifft zwar nur ausgewählte Teile des Systems, trotzdem hilft es dem Anwender bei der Orientierung im System. Hier sehe ich noch einiges an Potenzial. Die explizite Vergegenständlichung wesentlicher Begriffe der Fachsprache betrifft nur das Implementierungsmodell und damit die Entwickler. Für den Anwender ist diese unmittelbare Vergegenständlichung nur bedingt wahrnehmbar.

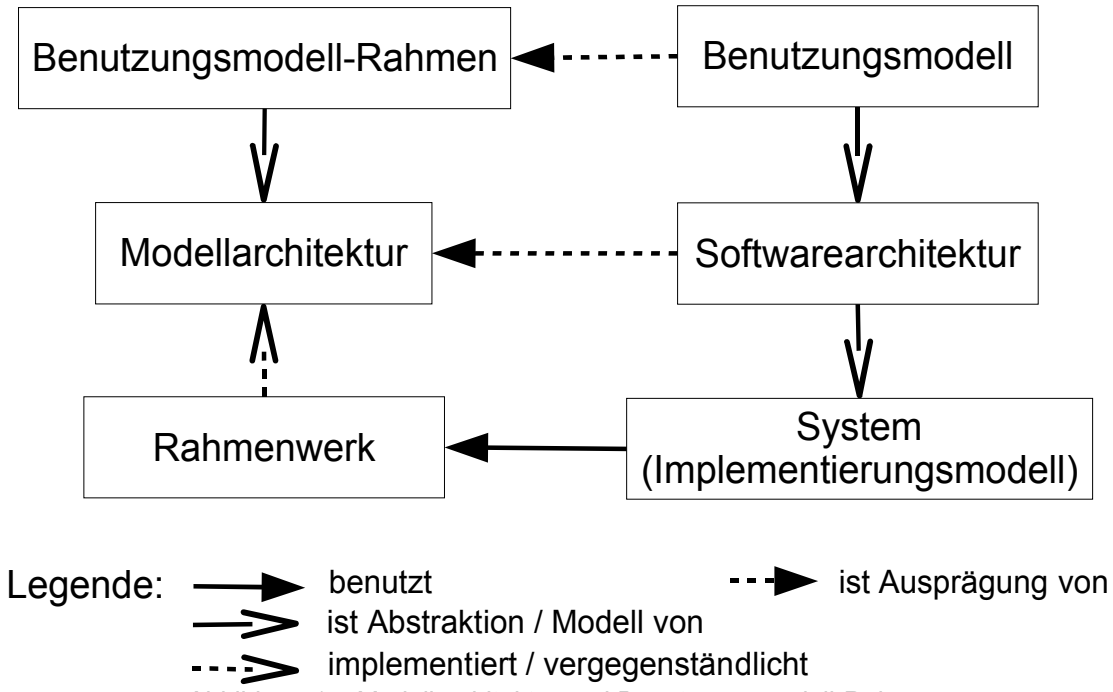

Abbildung 14: Modellarchitektur und Benutzungsmodell-Rahmen

Hier möchte ich ansetzen. Der Anwender soll mehr Zugriff auf bestimmte softwaretechnische Gegenstände des Anwendungssystems bekommen und mit ihnen unmittelbar umgehen können. Damit meine ich auch andere softwaretechnische Gegenstände als diejenigen, die das Begriffsmodell repräsentieren. Sicherlich sind dafür nicht alle softwaretechnischen Aspekte

geeignet. Welche jedoch dafür geeignet sein könnten und wie der Anwender sinnvoll mit ihnen umgehen kann, werde ich im Kapitel 6 und 7 ausführen.

Abbildung 14 gibt einen Überblick. Ein Benutzungsmodell ist eine Abstraktion über die konkrete Softwarearchitektur. Ausgewählte Elemente und Relationen werden übernommen. Über ausschließlich technische Aspekte wird abstrahiert. Das Benutzungsmodell ist dabei eine Ausprägung des Benutzungsmodell-Rahmens. Er ist wiederum eine Abstraktion der Modellarchitektur, nur ausgewählte Aspekte werden übernommen. Die Softwarearchitektur ist eine Abstraktion über das System (das Implementierungsmodell) und ist dabei eine Ausprägung der Modellarchitektur. Schließlich benutzt das Implementierungsmodell das Rahmenwerk, das wiederum die Modellarchitektur vergegenständlicht.

Wenn konzeptionelle Modelle (die Benutzungsmodelle) sozusagen automatisch erzeugt werden, ergeben sich zwangsläufig Nachteile gegenüber konzeptionellen Modellen, die explizit für einen bestimmten Zweck erstellt wurden. Streng genommen (nach der Definition aus Abschnitt 4.3.1) handelt es sich dabei nicht mehr um konzeptionelle Modelle. Der Zweck ist jedoch der gleiche, deshalb bleibe ich bei dieser Bezeichnung.

Die in diesem Abschnitt angesprochenen impliziten Benutzungsmodelle haben allerdings auch Vorteile. Zu jedem Teilsystem existiert automatisch ein Benutzungsmodell. Dies ist nicht der Fall, wenn man sich ein explizites Erstellen aus wirtschaftlichen Gründen nicht leisten kann. Das Benutzungsmodell ist auch immer mit dem tatsächlichen Systemverhalten konsistent. Mithilfe des Rahmenwerks kann schließlich die Konsistenz zwischen den verschiedenen Systemteilen unterstützt werden. Es ist allerdings auch denkbar, dass die automatisch erstellten Benutzungsmodelle im Nachhinein noch mit Informationen erweitert werden. Diesen Gedanken werde ich im Weiteren jedoch nicht verfolgen.

Wird solch ein Rahmenwerk für verschiedene Anwendungen benutzt, so kann mehr Aufwand in die Dokumentation fließen, z.B. in explizite konzeptionelle Modelle oder Schulungsunterlagen. In dieser Dokumentation werden dann die allgemeinen Prinzipien des Benutzungsmodell-Rahmens und die dazugehörigen Werkzeuge erklärt. Dies kommt allen auf diesem Rahmenwerk basierenden Anwendungen zugute.

Mit dieser Kern-Dokumentation und den Werkzeugen des Rahmenwerks wird Hilfe-zur-Selbsthilfe möglich. Mit Kenntnis der allgemeinen Prinzipien und den vom Rahmenwerk zur Verfügung gestellten Werkzeugen kann der Anwender selbst mehr über die Anwendungen herausfinden (aktives Lernen). Aufgrund der softwaretechnisch vergegenständlichten Benutzungsmodelle kann die Anwendung auf jede der gestellten Hypothesen konsistent zum Verhalten des Systems „antworten".

Das Rahmenwerk „Naked Objects" ([Pawson und Matthews 2002]) realisiert einen ähnlichen Ansatz, allerdings mit dem Ziel der schnellen Konstruktion von Prototypen (Rapid Prototyping). Der Entwickler muss lediglich Klassen für Geschäftsobjekte programmieren, wie z.B. `Person`. Das Rahmenwerk bietet generische Werkzeuge an, die ein einzelnes Geschäftsobjekt anzeigen und bearbeiten können. Zusätzlich existieren generische Werkzeuge, um sich alle Objekte einer Klasse auflisten zu lassen oder ein bestimmtes Objekt zu suchen. Die Metapher „Objektwelten" liegt hier zugrunde, die Stephan Dutke bereits

kritisiert hat (siehe [Dutke 1994]). Meines Erachtens greift die Metapher nicht weit genug, um damit aufgabengerechte Anwendungen zu konstruieren. Allerdings verdeutlichen „Naked Objects" meinen Ansatz, in einem Rahmenwerk allgemeine Prinzipien zu vergegenständlichen und konkrete Werkzeuge anzubieten, die sinnvoll auf diesen allgemeinen Prinzipien arbeiten können.

Die vorgestellte Lösungsidee löst fast alle in Abschnitt 4.4.5 angesprochenen Probleme. Entwickler erstellen beim Programmieren die Benutzungsmodelle implizit mit. Sie müssen lediglich in der Verwendung der neuen Rahmenwerks-Teile geschult werden. Dieser Aufwand erscheint aber vertretbar.

Das Rahmenwerk unterstützt die konsistente Programmierung bezüglich anderer Teile des Anwendungssystems. Unmittelbar verwendbare Werkzeuge aus dem Rahmenwerk können in unterschiedlichen Teilen der Anwendung wiederverwendet werden.

Durch die im Rahmenwerk enthaltenen Werkzeuge werden die Prinzipien unmittelbar be-greifbar und damit dem aktiven Lernen zugänglich. Eine gute Dokumentation und Schulungen über die dort vergegenständlichten Prinzipien ermöglicht eine Hilfe-zur-Selbsthilfe. Sie unterstützen auch eine Orientierung für nicht-vorweggenommene Spielräume.

Auf Grundlage dieser allgemeinen Prinzipien kann auch Anpassbarkeit realisiert werden. Spezielle Werkzeuge aus dem Rahmenwerk können anzeigen, welche Gegenstände eines Teilsystems anpassbar sind und wie diese Anpassungen durchgeführt werden können.

Eines der genannten Probleme bleibt jedoch ungelöst. In welcher *Form* sollen allgemeine Prinzipien des Benutzungsmodells vergegenständlicht werden. Hierfür schlage ich den Begriff des *Konzepts* vor, den ich im nächsten Abschnitt einführen werde.

5.2 Orientierung mit Konzepten

5.2.1 Zum Begriff Konzept

Konzepte sollen allgemeine Prinzipien und Regelhaftigkeiten des Benutzungsmodells vergegenständlichen. Auch hier möchte eine Analogie zu bekannten Sachverhalten ziehen. Mentale Modelle werden auf Basis von Gedächtnischemata aktiviert. Sie repräsentieren schematisches Wissen. Ihr Wert liegt darin, dass sie viele verschiedene Sachverhalte erklären können, da sie in vielen mentalen Modellen aktiviert werden können. Ich sehe bei ihnen allgemeine Prinzipien und Regelhaftigkeiten in der mentalen Welt des Anwenders repräsentiert.

Genauso wie ein konzeptionelles Modell als externe Vorlage für ein mentales Modell fungiert, soll ein *Konzept* als Vorlage für ein geeignetes Gedächtnischema dienen. Dies kann allerdings nicht unmittelbar geschehen, sondern nur über den „Umweg" des konzeptionellen und des mentalen Modells. Nur die Verbindung zwischen konzeptionellem Modell und mentalem Modell kann hier als möglicher „Eingabekanal" dienen.

Aus diesem Grund möchte ich den Begriff des Konzepts an dem des Gedächtnischemas ausrichten (siehe Abschnitt 4.2.3). Ein Konzept hat auf diese

Weise ähnliche Auswirkungen auf das konzeptionelle Modell wie ein Schema auf ein mentales Modell. Diese Merkmale können sich dann beim Verstehen auf das gebildete mentale Modell übertragen, sodass die gleichen mentalen Mechanismen greifen können, die aus einem mentalen Modell ein Schema extrahieren bzw. ein existierendes anpassen.

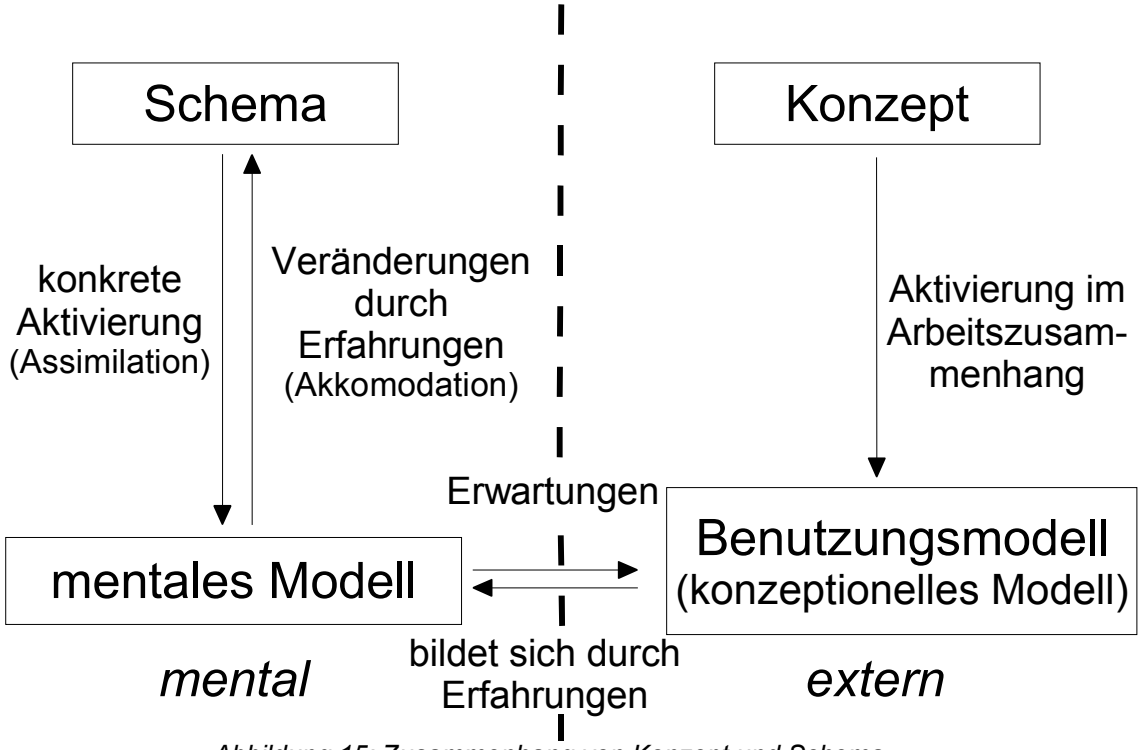

Abbildung 15: Zusammenhang von Konzept und Schema

Das Benutzungsmodell eines Anwendungsteils ergibt sich somit durch die Aktivierung einer oder mehrerer Konzepte analog zu der Aktivierung von Gedächtnischemata in mentalen Modellen. Abbildung 15 zeigt den Zusammenhang von Schema und Konzept. Die Strukturen, die von den Konzepten vergegenständlicht werden, sind in allen Aktivierungen gleich. Dadurch können Gemeinsamkeiten in verschiedenen Teilen der Anwendung realisiert werden. Der Anwender hat dadurch aber auch die Möglichkeit, allgemeine Prinzipien aus den unterschiedlichen, konkreten Benutzungsmodellen abzuleiten und Schemata für diese Prinzipien aufzubauen.

Eine ähnliche Funktion in der Softwaretechnik erfüllen Entwurfsmuster ([Gamma et al. 1995]). Sie beschreiben eine überschaubare Menge an softwaretechnischen Einheiten und ihre Verbindung untereinander. Mit ihnen können die zugrunde liegenden Strukturen einer Anwendung beschrieben werden. Der Name eines Entwurfsmusters hat dabei durchaus den Charakter einer Metapher.

Ein wesentlicher Unterschied zu Konzepten besteht jedoch darin, dass die Adressaten von Konzepten die Benutzer einer Anwendung sind und nicht die Entwickler wie bei Entwurfsmustern. Da der Anwender nur über Werkzeuge mit der Anwendung interagieren kann, müssen Konzepte softwaretechnisch so scharf beschrieben werden, dass Software-Werkzeuge auf ihnen arbeiten können. Mit ihnen kann dann der Benutzer die Anwendung erkunden und be-greifen. Insofern sind Konzepte keine „Muster”.

Ich fasse zusammen: Konzepte verbinden konzeptionelle Modelle (Benutzungsmodelle), Konsistenz und aktives Lernen. Sie vergegenständlichen softwaretechnische Regelhaftigkeiten des Anwendungssystems auf eine Art und Weise, dass sie für den Anwender erfahrbar werden. Diese Regelhaftigkeiten beschreiben in erster Linie fundamentale Strukturen des Systems.

Um sich dem Begriff *Konzept* weiter zu nähern, möchte ich jetzt die Merkmale von Gedächtnischemata daraufhin untersuchen, wie sie sich sinnvoll auf Konzepte übertragen lassen.

5.2.2 Umsetzung von Konzepten

Bevor ich die auf die Merkmale von Konzepten eingehe, möchte ich eine mögliche Umsetzung skizzieren. Dies soll helfen, die genannten Merkmale verständlicher zu erläutern. Für eine genauere Definition der kursiv gesetzten Begriffe aus diesem Abschnitt verweise ich auf [Schmolitzky 2000].

Sollen Konzepte in einem objektorientierten Rahmenwerk vergegenständlicht werden, so schränkt dies die mögliche Implementierung bereits ein. Die wesentlichen Bausteine objektorientierter Rahmenwerke sind *Klassen*. Mit Blick auf JWAM kommen noch *Schnittstellen* als Besonderheit der Programmiersprache Java hinzu. Schnittstellen abstrahieren über konkrete Implementierungen von Methoden und repräsentieren eine Sammlung von Methoden-Signaturen, was im Wesentlichen einem *Typ* entspricht. *Methodensignaturen* bestehen im Wesentlichen aus Methodennamen, Rückgabetyp, Parametertypen und der Reihenfolge der Parametertypen. Klassen können Schnittstellen implementieren, d.h. Implementierungen für die einzelnen Methoden anbieten. Klassen können auch unmittelbar einen Typ definieren, d.h. ohne eine explizite Schnittstelle zu implementieren.

Für die Umsetzung von Konzepten sind dabei zwei Aspekte wichtig. Zum einen betrifft dies den Modellcharakter von Typen. Sieht man Typen als Elemente eines Modells an, so steht ein Typ über seine Methodensignaturen in Relation zu anderen Typen, d.h. Rückgabetyp oder Parametertypen. Zum anderen betrifft dies die Subtyp-Beziehung. Typen können andere Typen *erweitern*, d.h. sie fügen weitere Methodensignaturen hinzu. Den erweiternden Typ nennt man *Subtyp*, den erweiterten Typ *Supertyp*. Konkrete Objekte als Rückgabe oder aktuelle Parameter bei einem Methodenaufruf dürfen Objekte eines Subtyps der in der Methodensignatur spezifizierten Typen sein. Dies nennt man *Polymorphie*.

Ein Beispiel: Gegeben seien zwei Klassen `Sparbuch` und `Girokonto`. Beide Klassen implementieren die Schnittstelle `Konto` und sind damit Subtypen von `Konto`. Eine Operation `berechneZinsen(Konto k)` kann mit einem aktuellen Parameter vom Typ `Konto` gerufen werden. In unserem Beispiel können dies ein Objekte der Klassen `Girokonto` oder `Sparbuch` sein.

Darüber hinaus möchte ich noch die *Vererbungs-Beziehung* ansprechen. Sie kann als eine spezielle Subtyp-Beziehung gesehen werden, die nur auf Klassen definiert ist. Eine *Subklasse* erbt von einer *Superklasse*. Die Subklasse ist dabei immer auch Subtyp der Superklasse. Darüber hinaus wird das konkrete Verhalten der Methoden mitvererbt.

Von einer konkreten Klasse kann ein *Objekt* erzeugt werden. Ein Objekt hat einen konkreten *Zustand*. An ihm können alle Methoden der erzeugenden Klasse

gerufen werden, wenn die benötigten Parameter-Objekte (und -Werte) mit übergeben werden.

Im Folgenden möchte ich unter der *softwaretechnischen Realisierung eines Konzepts* eine (überschaubare) Menge von Schnittstellen und Klassen verstehen, die ein Modell bilden. Als *Aktivierung* eines Konzepts kann man dann eine Menge von Objekten sehen, die die Typen des Konzepts implementieren und die gemäß des Konzepts in Relation zueinander stehen.

Wichtig ist aber auch, dass nicht jede Klasse und jede Schnittstelle zu der Vergegenständlichung eines Konzepts gehören. Es gibt auch Typen mit rein technischer Bedeutung.

Dieser Umsetzungsvorschlag dient in erster Linie dem besseren Verständnis des Begriffs *Konzept* und erhebt nicht den Anspruch einer in jeder Hinsicht angemessenen Umsetzung.

Vor diesem Hintergrund möchte ich nun auf die Merkmale von Konzepten eingehen.

5.2.3 Merkmale von Konzepten

Im Abschnitt 4.2.3 habe ich die Merkmale von Gedächtnischemata vorgestellt. An dieser Stelle möchte ich erläutern, welche dieser Merkmale sich sinnvoll auf den Begriff Konzept übertragen lassen und welche nicht.

Sinnvoll übertragbare Merkmale:

- **Konzepte sind Strukturen allgemeinen Wissens (über die Anwendung)**
- **Konzepte sind Abstraktionen über konkrete konzeptionelle Modelle**
- **Konzepte enthalten Leerstellen**
- **Konzepte enthalten Voreinstellungen für Leerstellen**
- **Konzepte sind hierarchisch verschachtelt**

Nicht sinnvoll übertragbare Merkmale:

- **Schemata unterliegen Veränderung**
- **Schemata beeinflussen kognitive Leistungen**

Im Folgenden möchte ich die Merkmale im Detail erläutern.

Konzepte sind Strukturen allgemeinen Wissens (über die Anwendung)

Konzepte können Wissen über ein Systemteil auf verschiedene Weise verallgemeinern. Der Systemteil kann Objekte nutzen, die unmittelbar von Klassen erzeugt wurden, die Teil der Vergegenständlichung des Konzepts sind. Es kann aber auch Objekte nutzen, die Subklassen der Konzept-Klassen sind. In beiden

Fällen weist das System ein konsistentes Verhalten mit dem Konzept auf.

Dies trifft insbesondere auch auf Werkzeuge zu, die auf den Konzepten eines Systemteils arbeiten. Aufgrund der softwaretechnischen Vergegenständlichung von Konzepten kann so ein Werkzeug auf allen Systemteilen arbeiten, die ein bestimmtes Konzept aktivieren.

Konzepte sind Abstraktionen über konkrete konzeptionelle Modelle

Technisch gesehen verhält es sich andersherum. Konzeptionelle Modelle sind aktivierte Konzepte. Aus Sicht des Anwenders kann man dieser Aussage jedoch zustimmen.

Konzepte enthalten Leerstellen

Die Leerstellen von Schemata können in Konzepten als Typen interpretiert werden, die mit dem Konzept in Relation stehen. Als Ausfüllen dieser Leerstelle kann die Belegung mit einem konkreten Objekt gesehen werden. Der Freiraum im Ausfüllen der Stelle kann dabei sowohl in den möglichen Zuständen des Objekts als auch des konkreten Subtyps gesehen werden. Möglicherweise können dabei auch ausschließlich Objekte mit einem bestimmten Zustand akzeptiert werden.

Konzepte enthalten Voreinstellungen für Leerstellen

Als „Voreinstellungen" können Standardimplementierungen von Konzepten gesehen werden, also Klassen, die ein Konzept auf eine bestimmte typische Weise implementieren. Ebenso können auch bereits konkrete Aktivierungen („Vorlagen") als Voreinstellungen gelten.

Konzepte sind hierarchisch verschachtelt

Zwischen Schemata gibt es eine Spezialisierungsbeziehung. Dies kann bei Konzepten mit der Subtyp-Beziehung abgebildet werden. Ein Konzept ist dann ein Subkonzept eines Superkonzepts, wenn seine Typen Subtypen der Typen des Superkonzepts sind.

Neben der Subtyp-Beziehung können Konzepte auch durch Komposition verschachtelt sein, d.h. ein Konzept lässt sich so betrachten, dass es sich aus anderen Konzepten zusammensetzt. Dies werde ich im nächsten Kapitel am Beispiel des Konzepts *Werkzeug* erläutern.

Schemata unterliegen Veränderung

Dieses Merkmal kann nicht auf Konzepte übertragen werden. Konzepte sollen ganz im Gegenteil stabil sein, damit der Anwender auf ihrer Basis geeignete Gedächtnischemata ausbilden kann.

Schemata beeinflussen kognitive Leistungen

Dieses Merkmal trifft nur indirekt über den Umweg des konzeptionellen zum mentalen Modell zu. Konzepte sollen aus Vorlage für das Ausbilden von Schemata dienen. Diese Schemata sollen dann die Orientierung in der Anwendung erleichtern. Konzepte sind deshalb zwar externe Vorlagen, sie beeinflussen kognitive Leistungen jedoch nicht unmittelbar.

Damit habe ich gezeigt, dass Konzepte, so wie ich ihre softwaretechnische Umsetzung beschrieben habe, den wesentlichen Merkmalen von Gedächtnisschemata gerecht werden. Man kann also annehmen, dass sie – über den Umweg des Benutzungsmodells und des mentalen Modells des Benutzers – als Vorlagen für Gedächtnisschemata dienen können.

5.3 Gestaltbarkeit: Anpassbarkeit als Grundprinzip

Bisher habe ich in diesem Kapitel die Wirkung von Konzepten auf *Orientierung* angesprochen. In diesem Abschnitt möchte ich die Wirkung auf *Gestaltbarkeit* und damit *Kontrolle* von Anwendungen durch den Benutzer erläutern.

5.3.1 Konzepte und Anpassbarkeit

Konzepte können in Bezug auf die Anpassbarkeit zwei Aspekte einer Anwendung unterstützen: die Orientierung für Anpassungen und eine *konzeptionelle Anpassbarkeit*.

Orientierung für Anpassungen meint, dass der Anwender anhand von bestimmten Konzepten erkennen kann, welche Gegenstände eines anpassbar sind und welche Auswirkungen die Anpassungen haben. Gleichzeitig können konkrete Werkzeuge angeboten werden, die auf einem bestimmten Anpassungs-Konzept arbeiten, um eine Anpassung durchzuführen. Diese Werkzeuge sind überall dort anwendbar, wo dieses Konzept aktiviert wird.

Als Beispiel möchte ich Tastenkombinationen für die Funktionen eines Werkzeugs nennen. Häufig können die Funktionen aus den Menüs eines Werkzeugs auch über Tastenkombinationen aufgerufen werden, z.B. <Steuerung>-<x> für das Ausschneiden von Text. Solche unmittelbar ausführbaren „Funktionen” von Werkzeugen möchte ich im Folgenden *Aktionen* nennen. Hierfür könnte ein spezielles Konzept *Aktivierung von Aktionen* vergegenständlicht werden. Auf diesem Konzept kann dann ein Anpassungs-Werkzeug arbeiten, das alle Aktionen eines Werkzeug auflistet und einer Aktion ein vom Benutzer gewähltes Tastenkürzel zuweisen kann.

Weiß der Anwender um dieses Konzept und dass ein bestimmtes Werkzeug dieses Konzept aktiviert, so weiß er auch, dass die Tastenkürzel dieses Werkzeugs anpassbar sind. Hat er einmal das Werkzeug zum Anpassen der Tastenkombinationen verwendet, so kann er mit demselben Anpassungs-Werkzeug die Tastenkombination eines jeden Werkzeugs anpassen, das das genannte Konzept aktiviert.

Dies kommt bereits dem Aspekt nahe, den ich mit *konzeptioneller Anpassbarkeit* bezeichne. Wird sichergestellt, dass jedes Werkzeug dieses Konzept aktiviert, z.B. durch eine gemeinsame Oberklasse, kann auch jedes Werkzeug entsprechend angepasst werden. Diese Art von Anpassbarkeit ist somit nicht willkürlich auf bestimmte Werkzeuge beschränkt, sondern basiert auf einem allgemeinen Konzept der Anwendung bzw. des Benutzungsmodell-Rahmens.

5.3.2 Anpassbarkeit als Grundprinzip

Ein wesentlicher Aspekt, um *Gestaltbarkeit* einer Anwendung zu erreichen, sind

nicht-vorweggenommene Spielräume (siehe Abschnitt 3.5). *Konzeptionelle Anpassbarkeit* ist ein wesentliches Prinzip, um solche nicht-vorweggenommenen Spielräume zu schaffen. Werden durch konstruktive Maßnahmen, wie z.B. eine gemeinsame Oberklasse, Anpassungskonzepte über die gesamte Anwendung „gestreut", so werden sehr viele, beinahe schon unzählige Spielräume geschaffen.

Mit einem Begriff wie *konzeptioneller Anpassbarkeit* stellt sich jedoch auch eine grundsätzliche Frage: Welche Teile einer Anwendung sollen überhaupt anpassbar sein und welche nicht? In dieser Arbeit fordere ich die Kontrolle des Anwenders über die Anwendung, insbesondere in Form von Gestaltbarkeit. Bei nicht-gestaltbaren Anwendungen entscheiden die Entwickler explizit über die Spielräume des Anwenders. Gestaltbarkeit sehe ich erst dann, wenn der Anwender aber auch *nicht-vorweggenommene* Spielräume nutzen kann. Daraus folgt, dass eine Anwendung *maximal anpassbar* sein sollte. Anpassbarkeit soll somit zum Prinzip erhoben werden. Alles soll prinzipiell anpassbar sein.

Dem sind natürlich Grenzen gesetzt. Die Entwickler müssen garantieren, dass die Anwendung stabil läuft und ihre Funktionen erfüllen kann. Insofern kann nicht wirklich *alles* anpassbar gemacht werden. Oft soll auch nicht jeder Anwender *jede* Funktion nutzen können, beispielsweise aus Kompetenz- oder aus Datenschutzgründen. Auch hier werden der Anpassbarkeit Grenzen gesetzt. Schließlich kann auch der Anwender mit so einer uneingeschränkten Fülle an Anpassungsmöglichkeiten überfordert sein.

Was die ersten beiden Argumente betrifft, garantierte Funktion der Anwendung und Benutzungsrechte, so möchte am Ende von Kapitel 6 anhand der vorgestellten Konzepte diskutieren, wie eine maximale Anpassbarkeit aussehen könnte. Was eine Überforderung des Anwenders anbelangt, so möchte ich dies in dieser Arbeit nur am Rande betrachten. Ich möchte die Möglichkeiten vorstellen, die ein Experten-Anwender nutzen könnte. Meines Erachtens gibt es auch einen gangbaren Weg von einem Neuling hin zu einem Anwendungs-Experten. Diesen Weg möchte ich in dieser Arbeit allerdings nicht aufzeigen. Lediglich möchte ich darauf hinweisen, dass ein Anwender ein Konzept nach dem anderen lernen kann. Außerdem kann er dabei mit allgemeinen Konzepten anfangen und Subkonzepte erst einmal ausschließen. Auf diese Weise sollte ein stetiger Lernpfad möglich sein.

Um wirklich zu nicht-vorwegnehmbaren Spielräumen zu kommen, benötigen wir allerdings noch ein fundamentales Prinzip: Kombinierbarkeit von Konzepten.

5.3.3 Kombinierbarkeit von Konzepten

Damit man von nicht-vorweggenommenen Spielräumen sprechen kann, muss die Anzahl möglicher Anpassungen sehr groß werden. Konzeptionelle Anpassbarkeit ist ein erster Schritt in die richtige Richtung. Die Anzahl möglicher Anpassungen wächst jedoch exponentiell, wenn man verschiedene Anpassungen kombinieren kann. Konzepte ermöglichen solche *kombinierbaren Anpassungen*.

Als Beispiel möchte ich das oben genannte Anpassungs-Konzept *Aktivierung von Aktionen* zum Definieren von Tastenkombinationen aufgreifen. Nimmt man ein zweites Anpassungs-Konzept „Makro" an, das dem Anwender ermöglicht, Makros zu definieren, so könnten beide auf folgende Arten kombiniert werden:

Es ist denkbar, dass Makros als zusätzliche Funktion an einem Werkzeug definiert

werden. Dann greift die Polymorphie von Konzepten. Das Konzept *Makro* wäre dann ein Subkonzept von *Aktion* und kann somit mit einer Tastenkombination versehen werden.

Es ist allerdings noch ein anderer Kombinationsmechanismus denkbar: Adaption. Angenommen das Makro wäre parametrisiert und würde die Belegung mit aktuellen Parametern fordern, bevor es aufgerufen werden kann, z.B. `berechneFakultätVon(int n)`. Dieses Makro wäre kein Subkonzept von *Aktion*, wie ich es oben definiert habe, da Aktionen wie ein Menüeintrag unmittelbar aktivierbar sind. Denkbar wäre nun ein Adapter, der aus einem parametrisierten Makro eine Funktion macht, z.B. indem vorher ein Dialog zur Eingabe der Parameter erscheint oder aber, indem die Parameter mit Standardwerten vorbelegt werden.

Mit solch *kombinierbaren Anpassungen* ist eine sehr große Anzahl von Anpassungen denkbar, sodass man von *nicht-vorweggenommenen Spielräumen* sprechen kann und damit von *Gestaltbarkeit*.

5.4 Zusammenfassung

In diesem Kapitel habe ich meine Lösungsidee skizziert, die den Kontrollverlust von Anwendern bei Computer-unterstützter Büroarbeit möglichst verringern soll. Sie besteht darin, allgemeine Prinzipien des Benutzungsmodell-Rahmens in einem Rahmenwerk softwaretechnisch zu vergegenständlichen. Teile des Anwendungssystems werden mithilfe des Rahmenwerks konstruiert und Verhalten sich deshalb konsistent zu den vergegenständlichten Prinzipien. Das Benutzungsmodell kann so als Abstraktion über die Implementierung eines Systemteils gesehen werden. Das Rahmenwerk bietet neben den Vergegenständlichungen Werkzeuge an, die auf den allgemeinen Prinzipien arbeiten, um sie für den Anwender erfahrbar zu machen.

Die Vorteile liegen darin, dass für jedes Systemteil ein Benutzungsmodell erstellt wird, das konsistent zum Verhalten des Systems ist. Da das Rahmenwerk wiederverwendet werden kann, kann es ausführlicher dokumentiert werden als die Anwendung selbst. Zusammen mit den Werkzeugen des Rahmenwerks wird so Hilfe-zur-Selbsthilfe möglich.

Konzepte sind das Mittel, um die allgemeinen Prinzipien des Benutzungsmodell-Rahmens zu vergegenständlichen. Sie sind eine externe Vorlage für das Ausbilden von geeigneten Gedächtnisschemata. Deshalb orientieren sich die Merkmale von Konzepten an denen von Gedächtnisschemata. Wesentlich an Konzepten ist, dass ihre Adressaten die Benutzer einer Anwendung sind. Es müssen deshalb für jedes vergegenständlichte Konzept Werkzeuge im Rahmenwerk angeboten werden, mit denen der Anwender die Konzepte begreifen kann. Zur Verdeutlichung des Begriffs habe ich auch bereits eine objektorientierte Umsetzungs-Möglichkeit für Konzepte skizziert.

Konzepte dienen aber nicht allein der Orientierung. Auch für die Gestaltbarkeit und damit die Anpassbarkeit von Anwendungen spielen sie eine wichtige Rolle. Es können nämlich allgmeeine Anpassungs-Konzepte vergegenständlicht werden, die es dem Anwender ermöglichen, die grundlegenden Aspekte der Anwendung anzupassen. Erst auf Basis von Konzepten wird so eine konzeptionelle bzw. kombinatorische Anpassbarkeit möglich. Auf diese Weise kann man nicht-

vorweggenommene Spielräume schaffen, die erst Gestaltbarkeit ermöglichen. Gestaltbarkeit heißt dabei auch, Anpassbarkeit zum Prinzip zu erheben. Diesen Aspekt werde ich am Ende des nächsten Kapitels noch genauer diskutieren.

Als nächstes möchte Konzepte vorstellen, die ich im WAM-Ansatz identifiziert habe; allen voran das Konzept des Arbeitszusammenhangs, dem ein zentrale Rolle bei der Orientierung und Gestaltbarkeit von WAM-Anwendungen zukommt.

Kapitel 6: Der Arbeitszusammenhang

Ziel dieser Arbeit ist es, einen Lösungsvorschlag zu erarbeiten, wie Software *gestaltbar* (siehe Abschnitt 2.5) konstruiert werden kann. Im vorherigen Kapitel habe ich hierzu bereits die Grundidee skizziert. In diesem Kapitel möchte ich nun diese Idee konkretisieren.

Hierfür werde ich in diesem Kapitel die zentralen Konzepte von WAM-Anwendungen erläutern. Ich greife dabei auf Konzeptions- und Entwurfsmuster aus [Züllighoven et al. 1998] zurück, die ich im Sinne von *Konzepten* umdeuten werde. Dazu werde ich Muster in Teilen abändern oder neue hinzufügen. Zugunsten einer verständlichen Darstellung verzichte ich auf eine detaillierte Gegenüberstellung von Konzepten und WAM-Mustern. Vereinzelt werde ich jedoch einen expliziten Zusammenhang zu WAM-Mustern herstellen.

Zunächst werde ich das Java-Zeiterfassungssystem grob skizzieren. Diese Anwendung soll in der restlichen Arbeit als Anwendungsbeispiel dienen. Danach werde ich die Konzepte einzeln im Detail erläutern. Der *Arbeitszusammenhang* als Kernkonzept dieser Arbeit dient dazu, all die anderen Konzepte in einen gemeinsamen Rahmen zusammenzuführen. Nach dem Einführen Klassifizierung von Konzepten gehe ich – wie im vorausgegangenen Kapitel angekündigt – auf den Begriff *maximale Anpassbarkeit* von WAM-Anwendungen ein.

6.1 JaZe - das Java-Zeiterfassungssystem

Um die in diesem Kapitel vorgestellten Konzepte zu verdeutlichen, aber auch als Grundlage für die Anwendungsfälle im nächsten Kapitel, möchte ich die Anwendung „Java-Zeiterfassungssystem" (kurz JaZe) als Beispiel verwenden. JaZe dient dazu, geleistete Arbeitszeiten von Mitarbeitern einer IT-Beratungsfirma zu erfassen. Die erfassten Zeiten dienen sowohl als Grundlage für die Lohn- und Gehaltsabrechnungen der Mitarbeiter, als auch für Rechnungen an Kunden.

Ich möchte an dieser Stelle nicht weiter auf den Leistungsumfang von JaZe eingehen, sondern werde in diesem und dem folgenden Kapitel die Kernkonzepte am Beispiel von JaZe und JaZe mit den hier vorgestellten Kernkonzepten erläutern.

Für das Verständnis ist es jedoch wichtig, dass JaZe im Wesentlichen von drei Benutzergruppen verwendet wird. Zuerst sind die *Mitarbeiter* zu nennen, die ihre geleisteten Zeiten auf Projekte buchen. *Projektleiter* können Berichte über die auf ihr Projekt gebuchten Zeiten erstellen. *Administratoren* schließlich übernehmen Verwaltungsaufgaben, wie z.B. das Erstellen neuer Projekte. Die Administratoren sind im Wesentlichen mit der Assistenz der Geschäftsführung identisch. Sie sind auch für die Gehaltsabrechnungen und die Rechnungsstellung an die Kunden verantwortlich. Für alle drei Benutzergruppen treffen die in Abschnitt 2.2 herausgearbeiteten Merkmale von Büroarbeit zu.

6.2 Ein erster Blick auf den Arbeitszusammenhang

Das Kernkonzept meiner Lösungsidee ist der *Arbeitszusammenhang*. Er repräsentiert eine Arbeitssituation in Form eines konzeptionellen Modells (eines Benutzungsmodells). Ein Arbeitszusammenhang bildet den Rahmen für die Erledigung einer Aufgabe. Hierfür hält er die benötigten Arbeitsobjekte und

Arbeitsmittel bereit und koordiniert sie entsprechend der Aufgabe. Gleichzeitig repräsentiert er eine bestimmte Abstraktionsebene. Die enthaltenen Arbeitsmittel können dabei wieder Arbeitszusammenhänge sein, die zur Erledigung spezieller Teilaufgaben dienen. Arbeitszusammenhänge lassen sich also hierarchisch gliedern.

Betrachtet man den Charakter des Arbeitszusammenhangs als konzeptionelles Modell, so besteht er aus Elementen und Relationen. Die Relationen möchte ich im folgenden *Verbindungen* nennen. Die Arbeitsobjekte und Arbeitsmittel werden durch Elemente repräsentiert. Diese Elemente sind entweder *Zeug-Exemplare* oder (*Fach*-)*Werte.*

Der Arbeitszusammenhang ist der Ort, an dem die einzelnen Konzepte zusammengestellt und kombiniert werden können. Aus diesem Grund möchte ich zunächst die anderen WAM-Konzepte besprechen. Mit ihnen kann ich dann gegen Ende des Kapitels das Konzept des Arbeitszusammenhangs besser erläutern.

6.3 Zeug und Werte

Zeug-Exemplare repräsentieren Gegenstände in einem WAM-Anwendungssystem, mit denen der Anwender bei seiner Arbeit umgeht. Sie haben eine *Identität*, einen *Zustand* und *Umgangsformen*. Umgangsformen definieren, wie mit einem Zeug-Exemplar umgegangen werden kann, was damit getan werden kann. Umgangsformen (*Operationen*) können *gerufen* werden. Der Anwender kann den Zustand abfragen (*Funktionen*) oder verändern (*Prozeduren*). Funktionen liefern dabei einen *Rückgabewert*. Umgangsformen nutzen das *Vertragsmodell*. Das Rufen einer Umgangsform setzt bestimmte Vorbedingungen voraus. Dies kann den Zustand des Exemplars betreffen, die aktuellen Parameter beim Aufruf der Umgangsform oder eine Kombination von beidem. Sind Vorbedingungen nicht eingehalten wird der Aufruf abgewiesen und statt dessen eine Fehlermeldung erzeugt.

Zeug-Exemplare mit gleichem Zustand können durch ihre verschiedenen Identitäten unterschieden werden. Zeug muss *erzeugt*, kann danach *verändert* und schließlich *zerstört* werden. Den Zeitraum von der Erzeugung bis hin zur Zerstörung nennt man *Lebensdauer* des Zeugs. Ein Beispiel für Zeug in JaZe ist ein Projekt. Trotzdem sollte dies konzeptionell unterschieden werden.

Im Gegensatz dazu haben (*Fach*-)*Werte* weder Identität noch Zustand, sondern nur einen konstanten Wert. Werte werden nicht erzeugt oder zerstört, sondern *selektiert*. Einen Wert zu selektieren kann, sich für den Anwender allerdings ähnlich präsentieren, wie das Erzeugen eines neuen Zeug-Exemplars.

Aufgrund einer fehlenden Identität kann nicht davon gesprochen werden, dass ein Wert verändert wird. Verändernde Umgangsformen gibt es an einem Wert nicht, sondern nur weitere Selektoren. In JaZe ist z.B. das Datum ein Wert. An einem Datum gibt aber die Umgangsform `gibZukuenftigesDatum(int tage)` ein Datum zurück, das eine bestimmte Anzahl von Tagen in der Zukunft liegt. Dabei wird das neue Datum selektiert. Das „alte" Datum bleibt unverändert. Aus Komplexitätsgründen und weil die in dieser Arbeit vorgestellten Konzepte mit geringem Aufwand auch auf Werte anwendbar sind, werde ich mich beim Vorstellen der Konzepte im Wesentlichen auf Zeug konzentrieren.

6.4 Konzepte

Zeug ist immer Exemplar eines *Konzepts*. Ein Konzept legt fest, in welcher Beziehung Zeug zu anderem Zeug stehen kann. Im Wesentlichen geschieht dies über die Umgangsformen. Ein Konzept definiert Umgangsformen. Unterschiedliche Exemplare des gleichen Konzepts müssen mindestens die Umgangsformen des Konzepts anbieten. Aufgrund ihres Zustandes stehen die Exemplare dieses Konzepts konkret mit anderen Zeug-Exemplaren in Verbindung. Auch Konzepte sind Zeug, nämlich Exemplare des Konzepts *Konzept*.

Konzepte sind auch untereinander verbunden. Zum ehier ein dauerhafte Anpassung zu inen über Umgangsformen, die selbst mit Konzepten definiert werden (z.B. ist die Rückgabe von Funktionen Zeug eines bestimmten Konzepts). Zum anderen können Konzepte auch in einer *Spezialisierungs*-Beziehung zueinander stehen: Subkonzepte bieten mindestens alle Umgangsformen eines Superkonzepts an. Diese Beziehung hat gewisse Ähnlichkeiten mit der programmiersprachlichen Subtyp-Beziehung. In Java mittels Schnittstellen (Schlüsselwort *Interface*) realisiert. Ein Konzept kann Subkonzept von unterschiedlichen Superkonzepten sein.

Für den Anwender können Konzepte als Filter oder zum Auswählen einer Untermenge dienen. Denkbar ist ein Konzept *Projektleiter* mit dem Superkonzept *Mitarbeiter*. Ein Anwender könnte mithilfe des Konzepts *Projektleiter* alle Projektleiter aus einer Menge von Mitarbeitern herausfiltern bzw. auswählen. Ein weiterer Anwendungsfall wäre das Auflisten von Werkzeugen, die ein bestimmtes Konzept bearbeiten können.

Konzepte als Zeug können auf diese Weise zum Verständnis einer Anwendung beitragen. Alle Exemplare, mit denen der Anwender umgehen kann, sind (per Definition) vom Konzept *Zeug*. Auch Konzepte sind Exemplare vom Konzept *Konzept*, das eine Spezialisierung des Konzepts *Zeug* ist. Neben Zeug gibt es auch *Nicht-Zeug*. Dies sind Klassen und Objekte des Systems, die ausschließlich der technischen Realisierung dienen sind und mit denen der Anwender nicht sinnvoll umgehen kann, wie z.B. Mapper von Zeug auf Tabellen in einer Datenbank. Nicht-Zeug sollte für den Anwender unsichtbar sein.

Mit Hilfe von Konzepten lässt sich eine Anwendung auf verschiedenen Abstraktionsebenen betrachten. Betrachtet man ausschließlich eine bestimmte Menge von Konzepten, so kann der Anwender seine Betrachtung auf bestimmte Aspekte konzentrieren und andere Aspekte ausblenden. Gerade für diesen Zweck haben die Konzepte *Material*, *Werkzeug* und *Automat* eine besondere Bedeutung.

Mit *Zeug* und *Konzept* greife ich im Wesentlichen die Konzepte Objekt und Typ der objektorientierten Programmierung auf. Wichtig ist aber, dass bei den Objekten zwischen Zeug und Nicht-Zeug unterschieden wird. Diese Zuordnung sollte sich aus der Unterscheidung von Benutzungsmodell-Rahmen und Modellarchitektur ableiten lassen.

6.5 Materialien, Werkzeuge, Automaten und Dienste

Die Konzepte *Werkzeug*, *Automat, Material* und *Dienst* adressieren grundlegend verschiedene Aspekte in einem Anwendungssystem. Sie repräsentieren disjunkte

Mengen von Zeug. Ein *Material*-Exemplar ist niemals gleichzeitig ein Exemplar des Konzepts *Werkzeug* oder *Automat*. Dies gilt auch umgekehrt. Interessiert sich ein Anwender für einen bestimmten Aspekt des Anwendungssystems, z.B. möchte er eine Tastenkombination ändern, so muss er dies an einem Werkzeug machen. Auf dieser Weise können diese Konzepte dem Anwender eine grobe Orientierung über das System geben. Im Folgenden möchte ich auf die einzelnen Konzepte im Detail eingehen.

6.5.1 Materialien

Materialien repräsentieren ausschließlich Arbeitsobjekte. Um eine Aufgabe zu erledigen, werden ausgewählte Materialien mithilfe ihrer Umgangsformen sondiert und manipuliert. An ihrem endgültigen Zustand zeigt sich das Arbeitsergebnis. Dabei sind Materialien passiv, d.h. Änderungen am Zustand werden ausschließlich durch den Anwender ausgelöst. Ist der Anwender selbst passiv, so bleibt der Zustand von Materialien unverändert und damit stabil.

Betrachtet man allein die Materialien eines Systems, so werden bereits hier grundlegende Aspekte der Arbeitserledigung deutlich. Aufgaben können bereits darauf abgebildet werden, *welche* Materialien verändert werden müssen. *Wie* und *womit* genau dies geschieht wird bei dieser Betrachtungsebene abstrahiert.

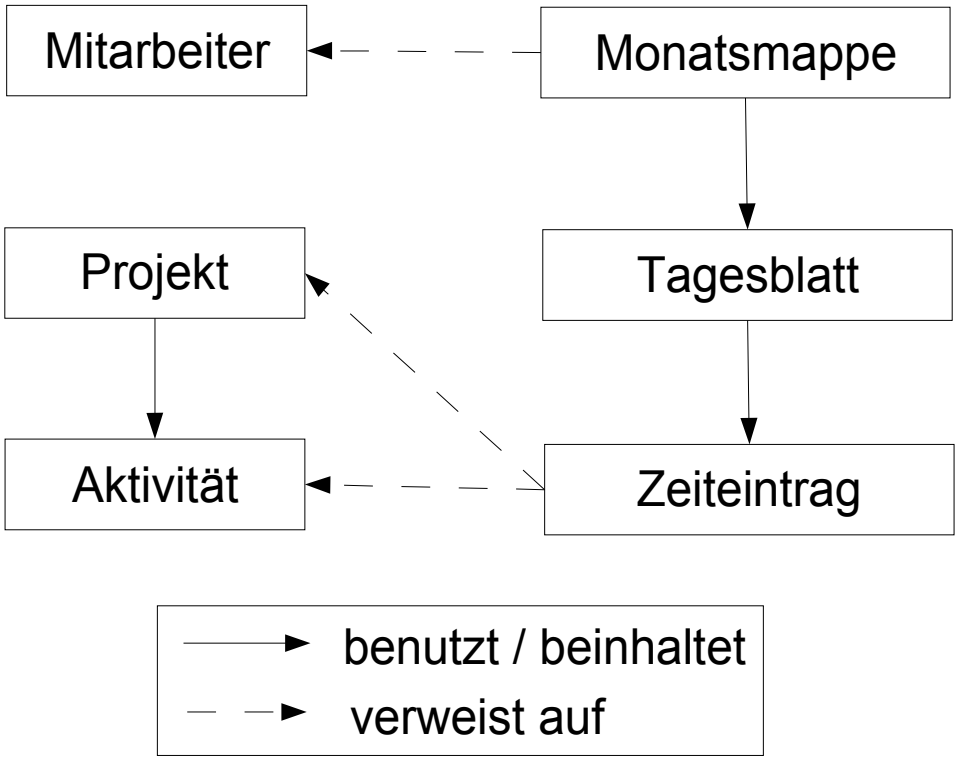

Abbildung 16: die Materialien in JaZe

Abbildung 16 zeigt die Materialien von JaZe. Projekte beinhalten Aktivitäten. Monatsmappen beinhalten Tagesblätter für einen bestimmten Monat. Tagesblätter beinhalten wiederum die Zeiteinträge für einen bestimmten Tag. Monatsmappen werden für einen bestimmten Monat und einen bestimmten Mitarbeiter angelegt. Auf diesen Mitarbeiter verweist die Monatsmappe. Zeiteinträge verweisen auf das

Projekt und die jeweilige Aktivität, auf die sie gebucht sind.

Bereits auf dieser sehr groben Betrachtungsebene für ein Anwendungssystem, die über jede Interaktion abstrahiert, kann man einige grundlegende Aussagen treffen. Die Aufgabe „Erfassen von Zeiten" für Mitarbeiter z.B. wird im Wesentlichen darin bestehen, Zeiteinträge zu erzeugen. Die Aufgaben von Administratoren wird im Wesentlichen vom Erzeugen und Manipulieren von Mitarbeitern, Projekten und Aktivitäten geprägt sein.

6.5.2 Werkzeuge

Werkzeuge dienen der interaktiven Arbeitserledigung. Manipulationen am System kann der Anwender nur mittels eines Werkzeugs durchführen. Werkzeuge haben als einziges Zeug eine sichtbare *Oberfläche*. Jede Manipulation an einem WAM-Anwendungssystem muss der Anwender mithilfe eines Werkzeugs umsetzen. Die Aktivität des Anwenders wird über die Oberfläche und durch das Werkzeug in das System moderiert. Dabei ist der Wirkungsbereich eines Werkzeugs über die *Arbeitet-auf-Beziehung* klar definiert. Ein Werkzeug arbeitet auf Zeug, typischerweise auf einem oder mehreren Materialien. Dies bedeutet, dass das Werkzeug ausschließlich diese Materialien darstellen und manipulieren kann.

Mit dieser Definition stellen Werkzeuge typische Arbeitsmittel dar. Anpassungen, wie die in Abschnitt 3.3.1 beschriebenen, werden deshalb typischerweise an Werkzeugen durchgeführt. Dies betrifft vor allem die Anpassungsgegenstände a) Synonyme, d) Interaktionsformen, g) Einschränkungen des verfügbaren Funktionsvorrats und h) Modifikation und Erweiterung der Funktionalität.

Einige Anpassungsmöglichkeiten auf Ebene des Werkzeugs möchte ich im Folgenden grob skizzieren. Abbildung 17 zeigt den Aufbau des Konzepts *Werkzeug*.

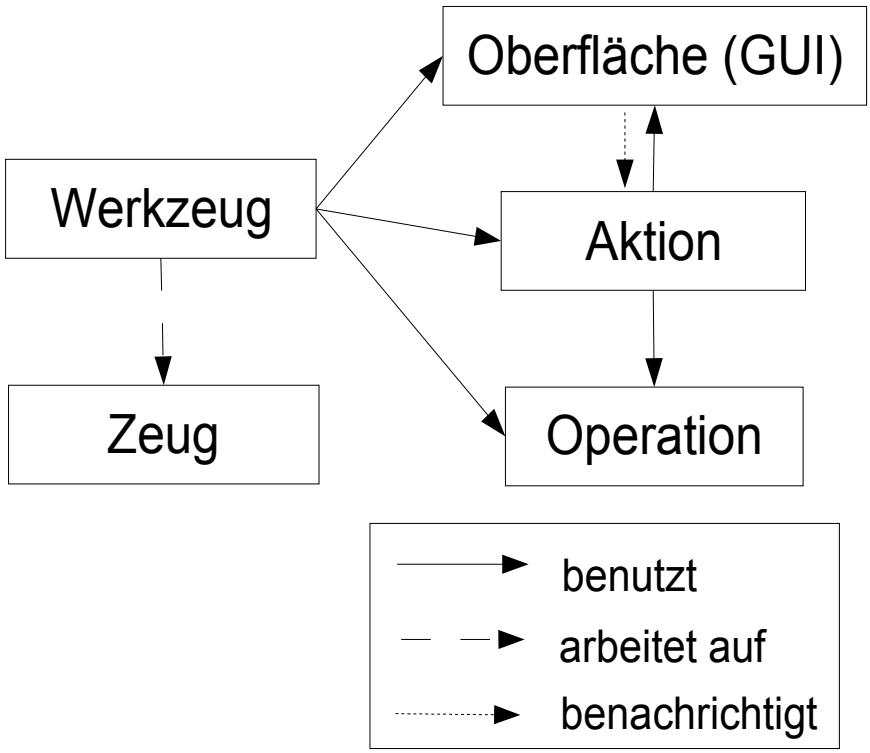

Abbildung 17: das Werkzeug-Konzept

Ein Werkzeug stellt das bearbeitete Zeug an der Oberfläche dar. Über Interaktionsformen, z.B. einen Knopf, löst der Anwender eine zugeordnete Aktion aus, z.B. sichern des veränderten Materials. Die Aktion greift auf die Oberfläche zu, um benötigte Parameter zu ermitteln, in unserem Beispiel die geänderten Werte des Materials und ruft dann eine fachliche Operation des Werkzeugs auf, um die Aktion auszuführen, und übergibt dabei die ermittelten Parameter.

An der Darstellung in Abbildung 17 wird auch die Komposition von Konzepten deutlich, wie ich sie im vorhergehenden Kapitel beim Merkmal „Konzepte sind hierarchisch verschachtelt" bereits erwähnt habe. Werkzeuge *bestehen* aus Oberfläche, Aktionen und Operationen. Der Anwender hat die Wahl zwischen zwei Abstraktionsebenen. Auf der abstrakteren Ebene kann man allein von dem Gegenstand *Werkzeug* sprechen, das auf einem bestimmten Material (Zeug) arbeitet. Auf der detaillierteren Abstraktionsstufe kann man Oberfläche, Aktionen und Operationen „einblenden". Der Anwender kann eine für ihn und die gesetzte Aufgabe geeignete Abstraktionsebene wählen.

Auf Grundlage des Konzepts *Werkzeug* sind folgende Anpassungen denkbar. Angenommen, die Oberfläche setzt sich aus einzelnen Interaktionsformen zusammen. Der Anwender kann dann eine Aktion einer zusätzlichen Interaktionsform zuordnen, z.B. einer Tastenkombination. Beim Drücken der Tastenkombination wird die Aktion ausgelöst.

Eine andere Anpassung könnte darin bestehen, dass der Anwender eine neue Aktion hinzufügt, wobei er eine bestehende Aktion als Vorlage benutzt. Für diese neue Aktion kann er das Ermitteln der Parameter oder die aufgerufene Operation ändern. Schließlich ist auch das Erzeugen einer neuen Operation denkbar. Eine solche benutzerdefinierte Operation würde einem „Makro" entsprechen (siehe

Abschnitt 3.3.1).

Als weitere Möglichkeit ist auch eine Anpassung der Oberfläche denkbar: umbenennen von Bezeichnungen oder Einschränken der in Menüs angezeigten Aktionen.

Die beschriebenen Anpassungen müssen wie alle Manipulationen mithilfe von Werkzeugen vorgenommen werden. Diese Werkzeuge arbeiten allerdings nur auf Ebene des in Abbildung 17 beschriebenen Konzepts. Sie können also generisch für alle Werkzeuge implementiert werden. Sollte ein Werkzeug darüber hinaus angepasst werden können, so müssten konkretere Werkzeuge angeboten werden.

Auf diese Möglichkeiten zur Anpassung von Werkzeugen möchte ich in dieser Arbeit nicht vertiefend eingehen, da ich den Fokus auf die Koordination verschiedener Werkzeuge mithilfe des Arbeitszusammenhangs setzen möchte. Mit dieser kurzen Skizze möchte ich jedoch verdeutlichen, welche Umgangsformen und Anpassungsmöglichkeiten mit einem allgemeinen Konzept *Werkzeug* bereits möglich sind.

6.5.3 Automaten

Automaten dienen wie Werkzeuge der Arbeitserledigung. Im Gegensatz zu Werkzeugen sind Automaten jedoch nicht interaktiv. Vielmehr werden sie mit einem Werkzeug eingestellt, d.h. ihr Zustand wird verändert und anschließend gestartet. Nach dem Start erledigt dieser Automat, sozusagen im Auftrag des Anwenders, gemäß den Einstellungen die vorgesehene Arbeit. Nach dem Erledigen der Arbeit, die eine gewisse Zeit in Anspruch nehmen kann, meldet der Automat den Vollzug.

Ich erwähne Automaten an dieser Stelle hauptsächlich der Vollständigkeit halber. In der weiteren Arbeit werde ich nur am Rande auf Automaten eingehen.

6.5.4 Dienste

Dienste sind nur in Ausnahmefällen für Anwender von Interesse. Sie repräsentieren Dienstleistungen, die von Werkzeugen oder Automaten in Anspruch genommen werden. Zum einen nimmt der Anwender sie als Operations-Sammlungen wahr, die er im Rahmen von Makro-Programmierung verwenden kann. Zum anderen repräsentieren sie technische Dienstleistungen wie z.B. das Bereitstellen von Persistenz durch *Ablagen*. Dienste sind i.d.R. passiv, können aber Aktivitäten von außen weiterleiten, wie z.B. das Eintreffen einer Email bei einem Email-Dienst. Auch Dienste möchte ich in dieser Arbeit nur am Rande behandeln.

6.5.5 Zusammenfassung

Zusammenfassend kann man sagen, dass Materialien und Dienste passive, Werkzeuge und Automaten aktive Elemente sind. Werkzeuge moderieren die Aktivitäten des Anwenders in das Anwendungssystem, während Automaten stellvertretend für den Anwender aktiv agieren. Materialien werden von Werkzeugen und Automaten bearbeitet, wobei sie auf Dienste zurückgreifen können. Ihr endgültiger Zustand repräsentiert das Arbeitsergebnis. Der Zustand von Werkzeugen ist dabei z.T. Ergebnis von Anpassungen. Dabei können bereits

auf Ebene des Konzepts *Werkzeug* konkrete Werkzeuge definiert werden, die solch typische Anpassungen vornehmen können.

6.6 Der Materialfluss: Orte, Ablagen, Behälter und Verpackungen

Ein wichtiger Aspekt in WAM-Anwendungssystem, insbesondere im Hinblick auf *Gestaltung* durch den Anwender, ist der *Materialfluss*. Arbeit wird unter Verwendung von Werkzeugen an Materialien vollbracht. Materialien werden aber nicht in Werkzeugen gelagert, sondern in *Behältern*. Den Weg vom Behälter zum Werkzeug und ggf. wieder zurück bezeichne ich als *Materialfluss*. In einem Arbeitszusammenhang vergegenständlicht er eine *Verbindung* zwischen Behältern und Werkzeugen.

Ein WAM-Anwendungssystem verwaltet typischerweise Mengen gleichartiger Materialien. JaZe z.B. verwaltet Monatsmappen, Tagesblätter und Zeiteinträge. Materialien, die eine Menge gleichartigen Zeugs bzw. Materialien verwahren, heißen *Behälter*. Sie verwahren Materialien. In JaZe ist z.B. ein Projekt ein Behälter für Aktivitäten. Spezielle Behälter sind *Ablagen*. Sie nehmen eine besondere Stellung ein, auf die ich später noch eingehen möchte. Die Projekt-Ablage in JaZe verwahrt z.B. Projekte.

Ein Behälter kann einzelne oder alle verwahrten Materialien zum Transport *herausgeben*. Er kann ein verwahrtes Material *aktualisieren* oder *entfernen*. So können Behälter auf das Einhalten von Konsistenzkriterien über die Menge an verwalteten Materialien achten. Die Projekt-Ablage von JaZe z.B. achtet darauf, dass die Projektnummern der Projekte über alle verwahrten Projekte hinweg eindeutig sind. Speziell solche Eindeutigkeitskriterien, die sich auf Werte der verwahrten Materialien beziehen, sind häufig. Aufgrund der Eindeutigkeit kann man anhand eines Wertes genau ein Material aus dem Behälter referenzieren. Der Projektbehälter von JaZe kann z.B. Projekte anhand einer gegebenen Projektnummer eindeutig identifizieren und herausgeben.

Materialien werden nur zur Bearbeitung herausgeben. Auch nachdem ein Material herausgegeben wurde, wird es immer noch vom Behälter verwahrt. Dies ist mit einer physischen Registratur vergleichbar, die immer eine Sicherheitskopie der herausgegebenen Akten bereithält. Materialien können deshalb nur dann endgültig zerstört werden, wenn sie explizit aus einem Behälter entfernt werden.

Behälter können Sub-Materialien von anderen Materialien sein oder selbst wieder von Behältern verwahrt werden. Hieraus ergibt sich eine Hierarchie. Der Behälter an der Spitze dieser Hierarchie sind *Ablagen*. Sie repräsentieren das eigentliche Materiallager. Aus technischer Sicht sind Ablagen für die Persistenz verantwortlich.

Ablagen sind zentrale Zugriffspunkte für Anwender, um an Materialien zu gelangen, insbesondere bei kooperativen Anwendungssystemen. Für solche Anwendungssysteme müssen Ablagen für alle Anwender zugreifbar sein und befinden sich deshalb an einem öffentlichen, d.h. für alle Mitarbeiter zugreifbaren, Ort. Für JaZe wäre dieser Ort der *JaZe-Server*. Ablagen an öffentlichen Orten sind typischerweise Registraturen. *Registratur* sehe ich als ein Subkonzept von Ablage mit einem bestimmten expliziten Kooperationsmodell (siehe auch [Züllighoven et al. 1998]).

Neben öffentlichen Orten gibt es für jeden Mitarbeiter einen eigenen, nur für ihn

zugreifbaren *Ort*, seinen *Arbeitsplatz*. Hier übt er die alleinige Kontrolle über seine Arbeitsmittel und Arbeitsobjekte aus. Diese Aufteilung ist in Hinsicht auf den in Abschnitt 2.4 beschriebenen Kontrollbegriff wichtig und findet ihre physische Analogie im Schreibtisch bzw. Büro eines Mitarbeiters und der frei zugänglichen Registratur bzw. Ablagen. Auch am Schreibtisch sind die Arbeitsmittel und Arbeitsobjekte (aufgrund sozialer Konventionen) vor dem Zugriff von Mitarbeitern geschützt.

Aus dieser Trennung von Arbeitsplatz und öffentlichen Orten folgt, dass zu bearbeitende Materialien von der Ablage am öffentlichen Ort zum Werkzeug am Arbeitsplatz des Anwenders transportiert werden müssen. Dies geschieht mit *Materialflüssen*. Abbildung 18 zeigt einen typischen Materialfluss in JaZe. Die Ablage verbleibt dabei am öffentlichen Ort (deshalb grau hinterlegt). Der ausgewählte Inhalt wird an den Arbeitsplatz transportiert und zum Werkzeug befördert. Da auch hier Mengen von Materialien transportiert werden müssen, werden geeignete Behältnisse für den Transport benötigt, die ich *Verpackungen*[9] nenne, z.B. *Listen*. Auf den Aufbau von Materialflüssen und die dafür benötigten Verpackungen werde ich bei den Anwendungsfällen im nächsten Kapitel genauer eingehen.

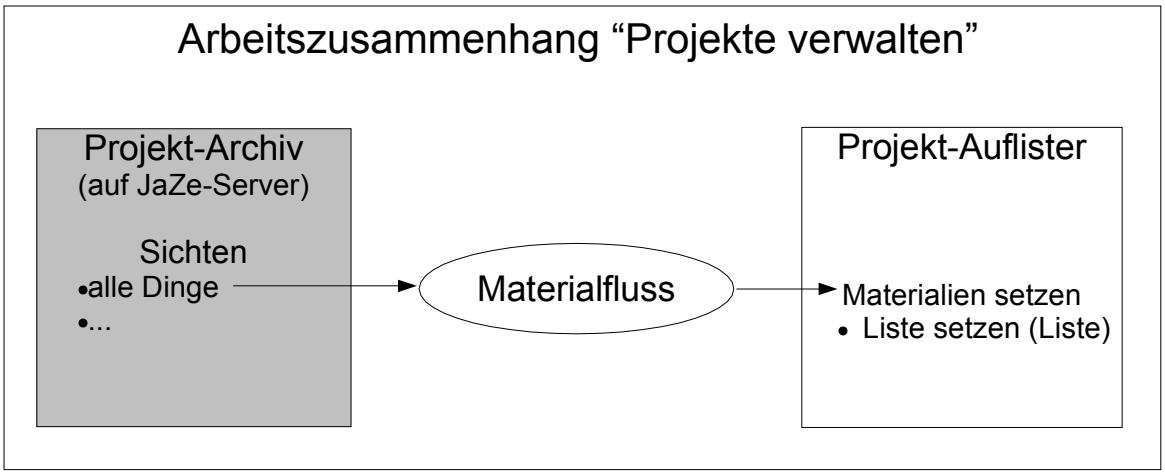

Abbildung 18: ein typischer Materialfluss in JaZe

An dieser Stelle möchte ich noch hinzufügen, dass veränderte Materialien wieder zur Ablage zurücktransportiert werden müssen, um dort persistent gemacht zu werden. Materialflüsse verfügen also auch über einen „Rücklauf”. (In der Abbildung nicht eingezeichnet.)

Ich fasse zusammen: Gleichartige Materialien werden in Behältern aufbewahrt. Behälter können wiederum von anderen Behältern verwahrt oder als Sub-Materialien verwaltet werden. Ablagen stehen dabei an der Spitze der sich ergebenden Hierarchie. Ablagen befinden sich i.d.R. an einem öffentlichen Ort. Materialflüsse sind für den Transport von der Ablage am öffentlichen Ort zum Werkzeug am Arbeitsplatz verantwortlich.

9 In der WAM-Terminologie sind Verpackungen mit technischen Behältern vergleichbar. Wie ich bei den Anwendungsfällen in Kapitel 7 jedoch noch zeigen werden, ist es sinnvoll, dass Anwender auch mit Verpackungen umgehen können. Verpackungen sind also Zeug.

6.7 Vorlage, Kopie und Original

Neben dem Transport von Materialien zum Werkzeug ist das Erzeugen von neuem Zeug am Arbeitsplatz ein wichtiger Aspekt der Arbeitserledigung. In JaZe z.B. erzeugen Mitarbeiter in erster Linie neue Zeiteinträge. Administratoren erzeugen neue Mitarbeiter, Projekte oder Aktivitäten.

Das Erzeugen neuer Exemplare ist auch typisch für andere Systeme, selbst wenn es sich nur um „Informationssysteme" handelt, bei denen es ausschließlich um Auswertungen geht. Das Erstellen von Projektberichten in JaZe kann beispielsweise als das Erzeugen neuer Berichts-Materialien auf Basis der gebuchten Zeiten verstanden werden.

Das Erzeugen neuer Exemplare kann man konzeptionell als das Erstellen einer *Kopie* verstehen. Das kopierte Exemplar ist das *Original*. Der Zustand von Kopie und Original sind gleich, allerdings ist die Identität unterschiedlich. Die Kopie kann dann entsprechend weiterverarbeitet werden. Zeug-Exemplare, die speziell dem Zweck dienen, kopiert zu werden, nenne ich *Vorlagen*.

Wichtig ist dabei, dass Kopie und Original nach dem Kopieren nicht mehr miteinander verbunden sind. Veränderungen an dem Original wirken nicht auf die Kopie. Der Wert des Konzepts Kopie liegt auch gerade in diesem einfachen und nachvollziehbaren Verhalten.

Das Erzeugen neuer Exemplare ist auch hinsichtlich der Arbeitsmittel interessant. Soll ein Material in einem Werkzeug bearbeitet werden, besteht die Möglichkeit, ein existierendes Werkzeug-Exemplar zu verwenden, um das ausgewählte Material zu bearbeiten. Dies bedeutet aber gleichzeitig, dass bei der Auswahl eines anderen Materials wieder dasselbe Werkzeug-Exemplar verwendet wird. Die Bearbeitung erfolgt so nur in einem einzelnen Werkzeug-Exemplar.

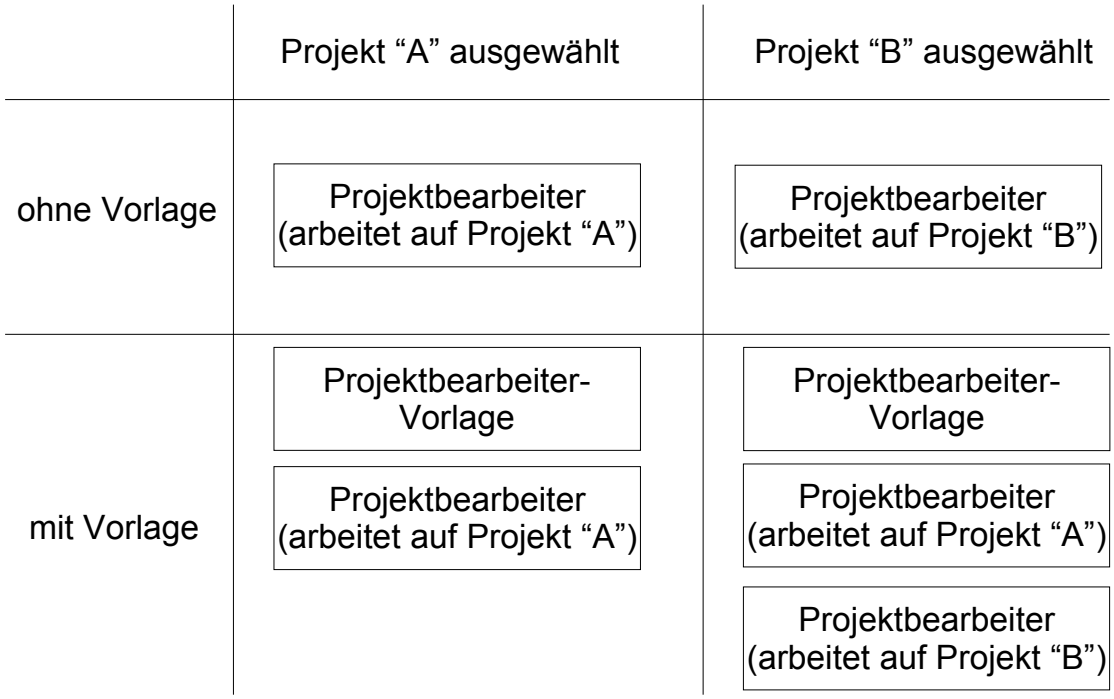

Abbildung 19: Bearbeitung von Projekten mit und ohne Werkzeug-Vorlage

Wird zunächst aber ein neues Werkzeug-Exemplar erzeugt und dann erst das Material eingespannt, so wird bei Auswahl eines neuen Materials ein neues Werkzeug erzeugt. Auf diese Weise können z.B. zwei Materialien miteinander verglichen werden oder es kann schnell zwischen diesen beiden Werkzeugen hin- und hergewechselt werden. Das Erzeugen neuer Werkzeug-Exemplare hat also Auswirkungen auf die Handhabung und sollte deshalb auch Gegenstand der Anpassung sein können. Über das Konzept der Werkzeug-Vorlage ist es für den Anwender manipulierbar. Abbildung 19 zeigt eine schematische Gegenüberstellung.

Auch in Bezug auf Anpassungen ist das Konzept der Vorlage mit seiner klaren Trennung von Original und Kopie hilfreich. Anpassungen bleiben bei dem einzelnen Projekt-Bearbeiter im Beispiel auch bei Auswahl eines neuen Materials erhalten. Es handelt sich ja um dasselbe Werkzeug-Exemplar. Wird bei Auswahl eines neuen Projekts allerdings ein neues Werkzeug erzeugt, so haben Anpassungen an der einen Kopie keinerlei Auswirkungen auf Anpassungen an der zweiten Kopie. Um hier eine dauerhafte Anpassung zu erreichen, muss das Original, die Vorlage, angepasst werden. Kopie und Original können also den Geltungsbereich (siehe Abschnitt 3.3.5) von Anpassungen deutlich machen.

Aufgrund der hier vorgestellten Möglichkeiten kann das Anlegen verschiedener Vorlagen als ein wichtiger Aspekt in der Anpassung von WAM-Anwendungssystemen angesehen werden. Für jeden Zweck kann der Anwender ein Exemplar als Vorlage definieren, das dem Ziel-Exemplar möglichst nahe ist und deshalb nach dem Kopieren einen minimalen Anpassungsaufwand erfordert. Das Anwendungssystem sollte ihn allerdings dabei unterstützen, solche Vorlagen zu verwalten, z.B. in speziellen Behältern.

6.8 Der Arbeitszusammenhang im Detail

Mit den bisher in diesem Kapitel besprochenen Konzepten kann ich nun das Konzept des Arbeitszusammenhangs näher erläutern. Wie bereits am Anfang dieses Kapitels erwähnt, repräsentiert ein Arbeitszusammenhang eine Arbeitssituation, d.h. die Erledigung einer bestimmten Aufgabe unter bestimmten Rahmenbedingungen.

Für diese Arbeitssituation beinhaltet er alle benötigten Arbeitsmittel und Arbeitsobjekte. Dies kann Zeug der bisher beschrieben Konzepte sein: Werkzeuge, Automaten, Materialien, Werte, Konzepte, Dienste, Ablagen, Behälter, Vorlagen, usw. Ein Arbeitszusammenhang verwahrt somit die *Elemente* der Arbeitserledigung. Um dem Anwender einen konzeptionellen Zugang zu ermöglichen, sind die wesentlichen Arbeitsmittel und Arbeitsobjekte *benannt*.

Darüber hinaus koordiniert der Arbeitszusammenhang die enthaltenen Arbeitsmittel und Arbeitsobjekte. Er kennt z.B. die benötigten Materialflüsse, die die benötigten Materialien automatisch aus ihren Ablagen in die entsprechenden Werkzeuge transportieren. Zum Koordinieren muss ein Arbeitszusammenhang um die *Verbindungen* zwischen den Elementen der Arbeitserledigung wissen.

Mit dem Wissen um die Elemente der Arbeitserledigung und ihrer Verbindungen vergegenständlicht ein Arbeitszusammenhang einen wesentlichen Teil des Benutzungsmodells. Aus diesem Grund sollte der Anwender ihn anpassen können.

Arbeitszusammenhänge können andere Arbeitszusammenhänge als Arbeitsmittel verwahren. Sie repräsentieren die Erledigung von Teilaufgaben. Das Wechseln zwischen verschiedenen Arbeitszusammenhängen kann man als Navigation bezeichnen. Abbildung 20 zeigt eine Übersicht der für Administratoren relevanten Arbeitszusammenhänge in JaZe.

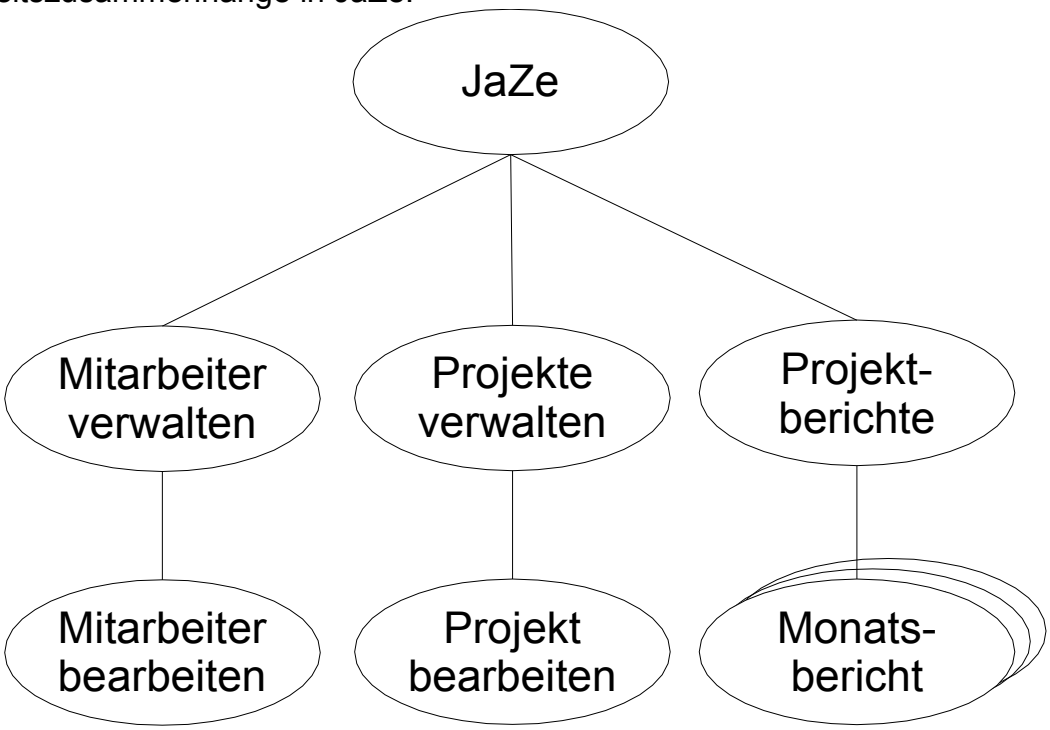

Abbildung 20: die Arbeitzusammenhänge für Administratoren in JaZe

Damit repräsentiert die Hierarchie der Arbeitszusammenhänge die grundlegende Struktur einer Anwendung. Im Gegensatz zum OAI-Modell (siehe 4.4.4) gibt es hier lediglich *eine* Hierarchie. Die vier verschiedenen im OAI-Modell als Hierarchien repräsentierten Aspekte finden sich als Querbezüge zwischen den Konzepten wieder, z.B. die Materialien als Objekte des Aufgabenbereichs oder Werkzeuge, die zusammengenommen die Oberfläche einer Anwendung ausmachen.

Jede Art von Arbeitserledigung durch den Anwender findet innerhalb eines Arbeitszusammenhanges statt. Ein Arbeitszusammenhang kann *aktiviert* und *unterbrochen* werden. Nach einer Unterbrechung kann er *reaktiviert* werden. Wird ein Arbeitszusammenhang aktiviert, stehen alle Arbeitsmittel und Arbeitsobjekte die für die Arbeitssituation benötigt bereit. Die Werkzeuge sind im richtigen Zustand voreingestellt, und die Materialflüsse haben für die benötigten Materialien gesorgt.

Wird ein Arbeitszusammenhang unterbrochen, so werden alle Werkzeuge unsichtbar. Wird er dann nach der Arbeitsunterbrechung reaktiviert, befinden sich alle Arbeitsmittel und Arbeitsobjekte im gleichen Zustand wie vor der Unterbrechung.

Ein Arbeitszusammenhang macht alle beteiligte Gegenstände der Arbeitserledigung für Anwender zugreifbar und damit manipulierbar. Außerdem

kann der Anwender benötigte Materialien und Werkzeuge im Arbeitszusammenhang ablegen. Außerdem kann er mithilfe von Platzhaltern Gegenständen benennen, die selbst nicht im Arbeitszusammenhang verwahrt werden, sondern erst aus lokalen oder entfernten Ablagen automatisch herausgesucht werden müssen.

6.9 Basiskonzepte, universelle Konzepte und fachliche Konzepte

Nachdem ich in diesem Kapitel diverse Konzepte vorgestellt habe, möchte ich auf eine hilfreiche Klassifizierung von Konzepten eingehen: *Basiskonzepte*, *universelle Konzepte* und *fachliche Konzepte*.

Die in diesem Abschnitt vorgestellten Konzepte sind größtenteils *Basiskonzepte*. Sie definieren die grundlegende (konzeptionelle) Struktur einer Anwendung. Der Nutzen für den Anwender liegt in einer besseren Orientierung. Allein mit ihnen kann er noch keine Arbeit erledigen. Basiskonzepte werden im System durch Spezialisierungen des Konzepts *Konzept* repräsentiert.

Im Gegensatz dazu umfassen die *universellen Konzepte* bereits konkretes Zeug, wie z.B. Verpackungen für Materialflüsse oder Werkzeuge zum Bearbeiten von Materialflüssen. Mit ihnen kann der Anwender im Prinzip bereits Arbeit erledigen, allerdings weisen sie keine fachlichen Bezüge auf. Ein universeller Auflister z.B. kann Listen von Zeug anzeigen, er hat allerdings keinen fachlichen Bezug z.B. zu Projekten.

Die hierfür benötigten Konzepte fallen in die dritte Klasse, die *fachlichen Konzepte*. Hierzu zählt konkretes Zeug, wie z.B. das Projekt-Material und das Projekt-Bearbeiter-Werkzeug aus JaZe. Eine Anwendung definiert sich im Wesentlichen über fachliche Konzepte.

Auf Grundlage der Basiskonzepte können Werkzeuge definiert werden, die universelle Arbeiten ausführen können. Dies betrifft Bereiche wie die Navigation, das Anzeigen von Mengen (Verpackungen) von Zeug und das Anpassen von Arbeitszusammenhängen und Werkzeugen. Als universelle Konzepte können sie allgemein und damit für eine große Klasse von Anwendungen definiert werden. In dieser Arbeit spreche ich die Klasse der Anwendungen zur Computer-unterstützten Büroarbeit an.

Ein wesentlicher Aspekt universeller Konzepte ist jedoch, dass von ihnen fachliche Subkonzepte gebildet werden können. Es kann sich z.B. herausstellen, dass die universelle Implementierung eines Auflister-Werkzeugs sich als nicht geeignet für das Auflisten von Projekten erweist. In diesem Fall kann ein aufgabengerechter Projektauflister als Subkonzept des Auflisters konstruiert werden. Aus Sicht der Orientierung ist dabei wichtig, dasd der Anwender auf Wunsch trotzdem auf den universellen Auflister zugreifen kann, um Projekte aufzulisten. Das Auflisten der Projekte mit den beiden verschiedenen Werkzeugen verdeutlicht dem Anwender die Subkonzept-Beziehung. Möglicherweise bietet der universelle Auflister auch Funktionen, die aus Zeitgründen im Projektauflister nicht implementiert wurden oder deren Implementierung für Projekte nicht sinnvoll erschien (vorweggenommene Spielräume). Der Anwender hat so die Möglichkeit, diese Funktionen trotzdem, auf universeller Ebene zu nutzen.

Basiskonzepte und universelle Konzepte eignen sich deshalb, um in einem

(anwendungsübergreifenden) Rahmenwerk zusammengestellt zu werden. Wenn diese Konzepte unmittelbar und an allen Stellen der Anwendung wiederverwendet werden, unterstützen sie die Konsistenz der Anwendung. Hat der Anwender gelernt, ein Werkzeug mit einem universellen Anpassungs-Werkzeug anzupassen, kann er dieses Wissen auf noch unbekannte Werkzeuge der Anwendung übertragen. Der Anwender muss deshalb nur einmal ihre Handhabung erlernen. Über diesen unmittelbaren Vorteil hinaus können auch Dokumentation, Online-Hilfe oder Schulungsmaterialien für universelle Konzepte wiederverwendet werden.

6.10 Maximale Anpassbarkeit von WAM-Anwendungen

Im vorhergehenden Kapitel habe ich den Begriff *maximale Anpassbarkeit* angesprochen. *Gestaltbarkeit* bedeutet für Anwender Kontrollgewinn und damit für Entwickler automatisch Kontrollverlust. Wie weit ist dieser Kontrolltransfer aber prinzipiell denkbar? Wo sind die maximalen Grenzen der Anpassbarkeit?

Bei genauer Betrachtung werden zwei unterschiedliche Grenzen erkennbar. Die erste Grenze markiert ab wann die Funktionalität einer Anwendung nicht mehr garantiert werden kann. Ein Werkzeug wurde z.B. so angepasst, dass es seine Funktion nicht mehr erfüllt. Kann der Anwender die Anpassung rückgängig machen, z.B. indem er das Werkzeug auf eine Standardeinstellung oder eine frühere Version zurücksetzen kann, so ist dies unproblematisch. Die zweite Grenze markiert die Stelle, ab der der Anwender Schäden anrichten kann, die nicht ohne Weiteres zu beheben sind. Beispielsweise, wenn ein Anwender in einem Banksystem Geld von einem Konto „verschwinden” lassen kann.

Vor dem Hintergrund der in diesem Kapitel vorgestellten Konzepte erscheint die zweite Grenze erreichbar, wenn lediglich *Materialien* und *Dienste* von Anwender Veränderungen durch den Anwender ausgeschlossen werden. Durch das *Vertragsmodell* ist dann sichergestellt, dass Materialien und *Dienste* korrekt verwendet werden. *Werkzeuge* und *Automaten* können dann beliebig verändert oder im Arbeitszusammenhang durch andere Werkzeuge ersetzt werden. Materialien lassen sich trotzdem nicht in fachlich unerwünschte Zustände überführen. Ablagen können ein Aktualisieren verweigern, wenn ein Material-Zustand nicht den Anforderungen der Ablage genügt.

Mit dieser Sichtweise repräsentieren die fachlichen Werkzeuge einer Anwendung lediglich *eine* Möglichkeit unter vielen, die Materialien zu manipulieren. Sie sind durch die Entwickler so gestaltet, dass sie bestimmte Aufgaben gezielt unterstützen. Der Anwender hat aber die Möglichkeit, die Materialien auch auf andere Art zu manipulieren. Er kann universelle Werkzeuge verwenden, universelle und fachliche Werkzeuge neu verknüpfen oder aber - was in dieser Arbeit allerdings nur skizziert wird - Werkzeuge *umzugestalten*, indem der Anwender z.B. Aktionen und Interaktionsformen anpasst.

Dies hat Auswirkungen auf das Rollenverständnis von Entwickler und Anwender. Der Entwickler konstruiert kein Fertigprodukt, das der Anwender nur in vorbestimmten Bahnen verwenden kann. Vielmehr bietet er eine Menge von Bausteinen an, die bereits zu einem funktionsfähigen Ganzen zusammengefügt wurden. Der Anwender ist jetzt nicht mehr „Benutzer” einer Anwendung, sondern er kann die einzelnen Bausteine erkennen und sie neu zusammenfügen. Insbesondere für Aufgaben, die die Entwickler nicht vorweggenommen haben, ist

dies von Vorteil.

Dabei spielen universelle Konzepte eine große Rolle. Sobald ein fachliches Werkzeug zu speziell ist, hat der Anwender die Möglichkeit auf universelle Konzepte auszuweichen. Es ist aber wichtig, dass der Anwender dabei trotzdem auf die fachlichen Konzepte zugreifen kann, damit der Bezug zu seiner fachlichen Tätigkeit bestehen bleibt. Gerade in der Kombination von fachlichen und universellen Konzepten ergeben sich *nicht-vorweggenommene Spielräume* und damit *Gestaltbarkeit*.

6.11 Zusammenfassung

Nachdem ich im vorausgegangenen Kapitel den Begriff *Konzept* allgemein eingeführt habe, habe ich in diesem Kapitel verschiedene Konzepte vorgestellt. Ich habe dabei auf Konzeptions- und Entwurfsmuster aus dem WAM-Ansatz zurückgegriffen, die ich im Sinne von Konzepten neu interpretiert habe.

Das grundlegende Konzept ist *Zeug*. Alle Gegenstände, mit denen der Anwender in WAM-Anwendungen umgeht, sind Zeug. Zeug definiert sich im Wesentlichen über seine *Umgangsformen*.

Das Konzept *Konzept* schlägt eine Brücke für den Benutzer von den konkreten Gegenständen einer Anwendung zu ihren allgemeinen Konzepten. Zeug ist immer Exemplar eines Konzepts. Das Konzept eines Zeug-Exemplars legt fest bzw. schränkt ein, wie ein Zeug-Exemplar mit anderen Zeug-Exemplaren in Verbindung stehen kann. Neben Verbindungen, die über Umgangsformen hergestellt werden, können verschiedene Konzepte in einer *Subkonzept-Beziehung* stehen. Auf diese Art und Weise können Taxonomien entstehen, die dem Anwender helfen, sich im System zu orientieren und es besser zu nutzen, z.B. um Untermengen von Zeugmengen herauszufiltern oder auszuwählen.

In Bezug auf die Arbeitserledigung sind zwei Subkonzepte von Zeug besonders wichtig: *Werkzeug* und *Material*. Werkzeuge repräsentieren Arbeitsmittel, Materialien repräsentieren Arbeitsobjekte. Werkzeuge arbeiten (typischerweise) auf Materialien. Diese beiden Konzepte sind unmittelbar an der Arbeitserledigung durch den Anwender beteiligt. Der Anwender kann nämlich nicht unmittelbar auf Materialien zugreifen, da sie für sich genommen unsichtbar sind. Ausschließlich Werkzeuge haben eine Oberfläche und können so Materialien (oder Zeug allgemein) für den Anwender erfahrbar und manipulierbar machen.

Ein weiterer wichtiger Aspekt für die Arbeitserledigung ist die Quelle von Materialien. Materialien werden in *Ablagen* gelagert, die sich typischerweise an anderen *Orten* befinden. Von dort müssen sie erst einmal auf den *Arbeitsplatz* transportiert und in die betreffenden Werkzeuge zum Bearbeiten eingespannt werden. Diese Aufgabe übernehmen *Materialflüsse*. Sie sorgen auch dafür, dass die geänderten Materialien wieder zurück in die Ablagen gelangen.

Mit den Konzepten *Vorlage*, *Kopie* und *Original* habe ich eine einfache Möglichkeit vorgestellt, wie der Anwender neue Zeug-Exemplare erzeugen kann. Außerdem kann man das Konzept der Vorlage auch auf Werkzeuge anwenden. Verwendet man ein Werkzeug als Vorlage, so führt dies zu einem anderen Verhalten, als wenn man ein Werkzeug unmittelbar verwendet. Der Anwender kann mithilfe des Konzepts *Vorlage* entscheiden, welche dieser Varianten er für eine bestimmte

Aufgabe für angemessen hält.

Als das zentrale Konzept der vorliegenden Arbeit, nimmt der *Arbeitszusammenhang* eine herausragende Stellung unter den anderen Konzepten ein. Ein Arbeitszusammenhang repräsentiert eine bestimmte Arbeitssituation und bietet dem Anwender die benötigten Arbeitsobjekte und Gegenstände. Jede Form von Aufgabenerledigung vollzieht der Anwender innerhalb eines Arbeitszusammenhangs. Aus diesem Grund erscheint es sinnvoll, ihn anpassbar zu konstruieren.

Da ich in diesem Kapitel sehr viele verschiedene Konzepte vorgestellt habe, erschien es mir sinnvoll, eine Klassifizierung von Konzept einzuführen: *Basiskonzepte*, *universelle Konzepte* und *fachliche Konzepte*. *Basiskonzepte* definieren allgemeine Zusammenhänge eines Anwendungssystems. Universelle Konzepte stellen zwar allgemeine Konzepte dar, die allerdings schon konkret vergegenständlicht sein können, wie z.B. Verpackungen oder generische Werkzeuge. *Fachliche Konzepte* definieren schließlich fachlich konkretes Zeug, das die Grundlage für eine fachliche Aufgabenerledigung in einer Anwendung bildet, z.B. das *Projekt*-Konzept in JaZe.

Abschließend bin ich auf den Begriff der *maximalen Anpassbarkeit* eingegangen. *Kontrolle* einer Anwendung durch den Anwender heißt für mich, *Gestaltbarkeit* zum Prinzip zu erheben; alles anpassbar zu machen, was nicht die grundlegende Funktionalität einer Anwendung gefährdet. Für die in diesem Kapitel vorgestellten Konzepte sind nur die konkreten fachlichen *Materialien* und *Dienste* für das Funktionieren einer Anwendung notwendig. Der eigentliche Umgang, das Arbeiten mit den Materialien, kann aber vom Anwender beliebig angepasst werden. Besonderheiten und Ausnahmen können über das Vertragsmodell abgefangen werden.

Kapitel 7: Anpassungen im Arbeitszusammenhang

In diesem Kapitel möchte ich eine Vorstellung davon vermitteln, wie Orientierung und Gestaltbarkeit auf Basis von Konzepten aus Anwendersicht umgesetzt werden könnte. Hierbei konzentriere ich mich im Wesentlichen auf einen Anwendungsfall.

Die im Laufe des Kapitels vorgestellten Werkzeuge und verwendeten Konzepte wurden nicht implementiert, es handelt sich lediglich um Oberflächenprototypen. Hinzu kommt, das JaZe als Web-Anwendung realisiert ist. Obwohl sie das Rahmenwerk JWAM verwendet, werden nicht alle WAM-Entwurfsmetaphern softwaretechnisch umgesetzt. *Werkzeuge* werden in JaZe z.B. nicht vergegenständlicht. Aus Sicht des Anwenders lässt sich JaZe aber sinnvoll mit den WAM-Konzepten beschreiben und eine Implementierung auf Basis der WAM-Konzepte wäre ohne größere Probleme umsetzbar. Für dieses Kapitel lege ich diese mögliche Implementierung zugrunde.

Der in diesem Kapitel beschriebene Anwendungsfall hat einen realen Hintergrund. Beim Einsatz von JaZe wurde von den Anwendern eine Änderung gefordert und schließlich umgesetzt. Zunächst beschreibe ich deshalb die fachliche Ausgangssituation und werde dann das Benutzungsmodell auf Basis der WAM-Konzepte vorstellen. Anschließend skizziere ich eine mögliche Anpassung, die ein Anwender mit den im vorausgegangenen Kapitel vorgestellten Konzepten hätte umsetzen können. Danach erläutere ich mit WAM-Konzepten als Beschreibungsmittel die vom Entwickler am Programmcode tatsächlich vorgenommenen Änderungen. Abschließend werde ich den vorgestellten Anwendungsfall und die Lösung diskutieren und auswerten.

7.1 Die Ausgangssituation

Nachdem JaZe einige Monate im Einsatz war, zeigte sich, dass mit dem Erstellen neuer Projekte die Liste der bestehenden Projekte zusehends unübersichtlicher wurde. Es wurde beschlossen, die Liste in zwei Listen aufzuteilen. Eine Liste mit den aktiven Projekten und eine Liste mit den abgeschlossenen Projekten.

Die Ausgangssituation stellt sich folgendermaßen dar: Um auf die Projekte zuzugreifen, aktiviert der Anwender den Arbeitszusammenhang „Projekte verwalten". Dazu klickt er in der Navigationsleiste auf den Knopf „Projekte". Ihm präsentiert sich dann das in Abbildung 21 dargestellte Bild. Der Arbeitszusammenhang „Projekte verwalten" besteht im Wesentlichen aus zwei Werkzeugen. Zuerst ist der Projekt-Erzeuger zu nennen (1). Er dient dem Erzeugen neuer Projekte. Dazu müssen lediglich der Projektname und der Kundenname eingegeben werden. Darunter befindet sich der Projektauflister (2). Die Navigationsleisten oben und links gehören zum übergeordneten Arbeitszusammenhang.

Um einen mehr konzeptionell orientierten Blick auf das Benutzungsmodell zu bekommen, öffnet der Anwender den *Navigator* (siehe Abbildung 22). Er bietet eine navigierbare Sicht über die Arbeitszusammenhänge. Ein Doppelklick mit der Maus auf einen Arbeitszusammenhang aktiviert ihn bringt ihn mit all seinen Werkzeugen an die Oberfläche. Der Navigator ist vergleichbar mit der *Taskleiste* unter Microsoft Windows XP. Die Taskleiste enthält alle gestarteten Programme, der Navigator alle aktiven und aktivierbaren Arbeitszusammenhänge. Im

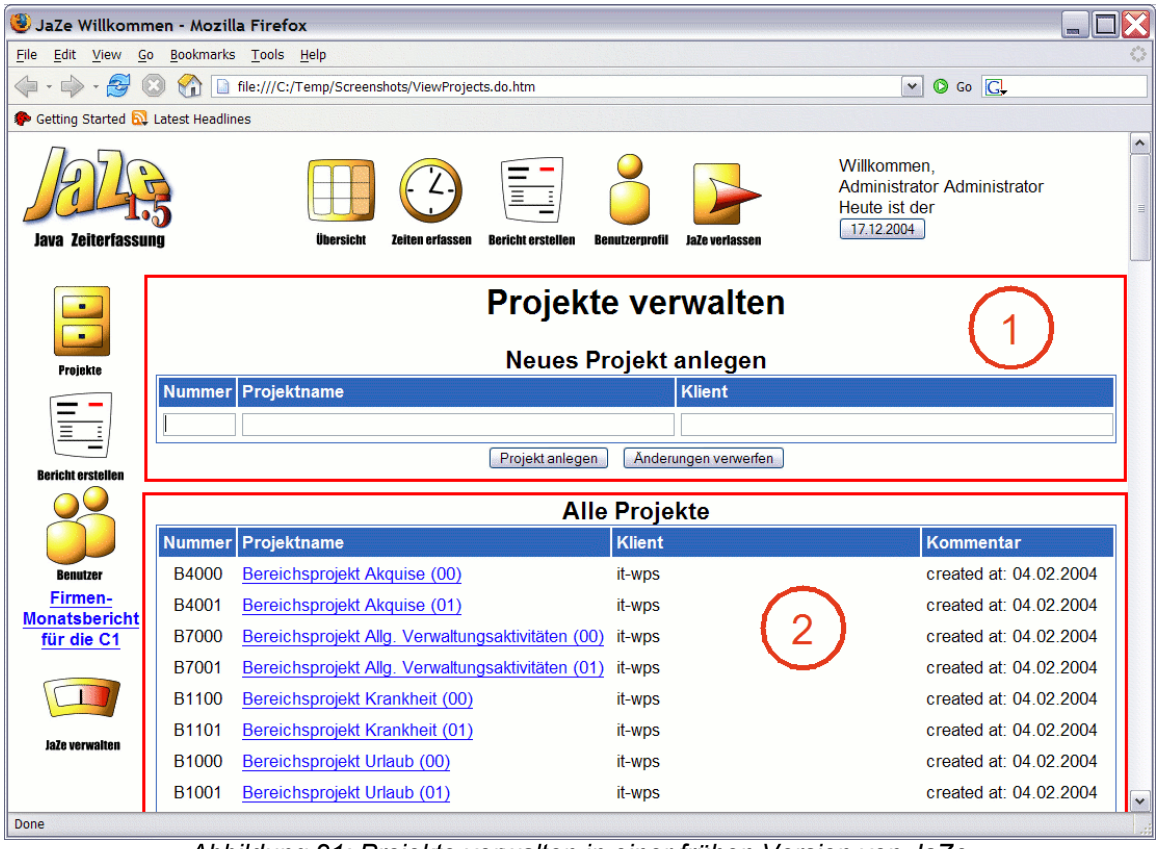

Abbildung 21: Projekte verwalten in einer frühen Version von JaZe

Gegensatz zur Taskleiste bildet der Navigator jedoch eine hierarchische Struktur ab, was den tatsächlichen Strukturen von Anwendungen näher kommt (siehe auch das OAI-Modell aus Abschnitt 4.4.4). Die blauen Zahnrad-Piktogramme repräsentieren dabei das Konzept *Arbeitszusammenhang*. Die Abbildung 22 zeigt die erste Ebene der Arbeitszusammenhangs-Hierarchie in JaZe vollständig an. In der zweiten Ebene sind nur die Arbeitszusammenhänge aus Abbildung 20 dargestellt.

Der Anwender möchte jedoch nicht navigieren, sondern das Benutzungsmodell ändern. Dazu aktiviert er den *Erkunder* im Menü. Dieser öffnet sich direkt neben dem Navigator (siehe Abbildung 23). Der Erkunder ermöglicht dem Anwender expliziten Zugriff auf das Benutzungsmodell. Mit seiner Hilfe kann er die Gegenstände des Arbeitszusammenhangs erkunden und manipulieren.

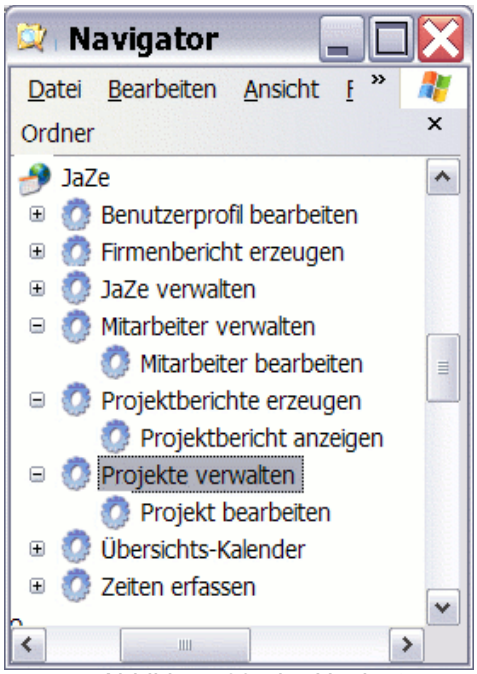

Abbildung 22: der Navigator

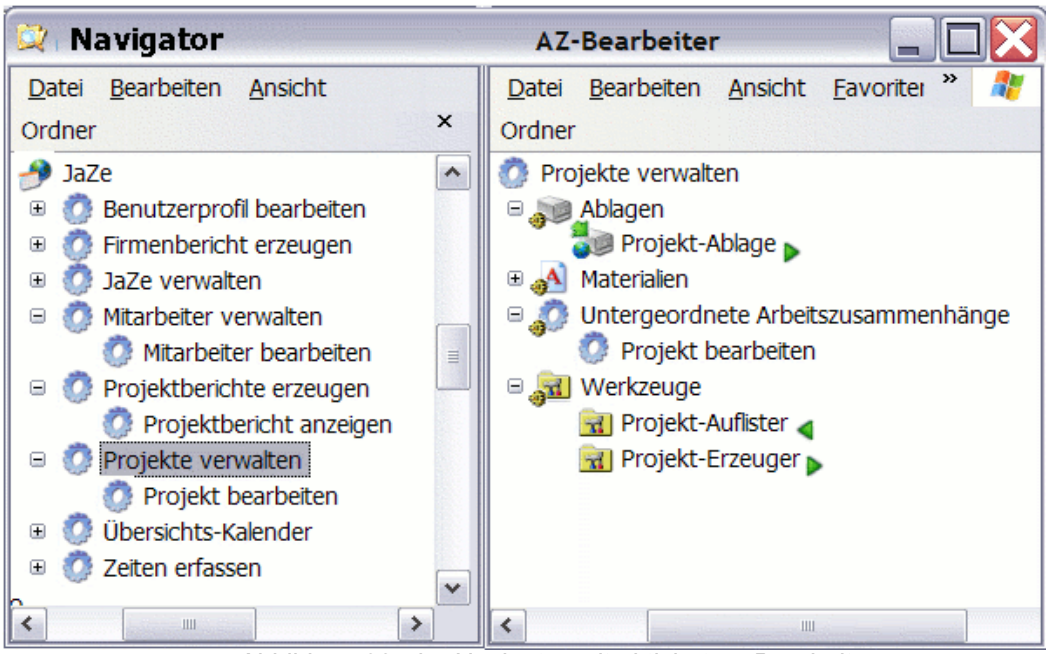

Abbildung 23: der Navigator mit aktiviertem Bearbeiter

Diese Ansicht ist durchaus mit einem Arbeitsverzeichnis benötigter Dateien vergleichbar, wie es der Explorer unter Microsoft Windows XP darstellt. Allerdings werden keine Verzeichnisse und Dateien dargestellt, sondern das für diesen Arbeitszusammenhang relevante Zeug. Die erste Ebene dieser Hierarchie bilden Verzeichnisse verschiedener Konzepte: *Ablagen*, *Materialien*, u*ntergeordnete Arbeitszusammenhänge* und *Werkzeuge*. Unter „Ablagen" finden sich z.B. alle Ablagen, die für den Arbeitszusammenhang „Projekte verwalten" relevant sind.

Das kleine Zahnrad vor den Bezeichnungen weist darauf hin, dass es sich hierbei um Filter handelt, d.h. automatisch erstellte Verzeichnisse. Der Anwender muss diese Verzeichnisse nicht manuell pflegen. Das Verzeichnis „Ablagen" enthält alle Zeug-Exemplare im Arbeitszusammenhang, die ein Subkonzept von Ablage sind. Bei den dargestellten Verzeichnissen handelt es sich um eine Voreinstellung, die der Anwender ändern kann. Für die Koordination sind diese vier Konzepte jedoch von wesentlicher Bedeutung, sodass diese Voreinstellung sinnvoll erscheint.

Die kleineren Piktogramme liefern weitere Informationen. Nehmen wir als Beispiel die „Projekt-Ablage". Die kleine Weltkugel links unten zeigt an, dass es sich hierbei um eine entfernte Ablage handelt. Sie befindet sich an einem anderen Ort. Aus dem Kontext-Menü kann der Anwender erfahren, dass es sich bei dem Ort, um den JaZe-Server handelt. Der kleine grüne Pfeil links oben weist darauf hin, dass die Ablage aus einem übergeordneten Arbeitskontext importiert wurde. Die Projekt-Ablage wird in diesem Arbeitszusammenhang also nicht verwahrt, sondern er enthält lediglich einen *Verweis*. Alle Ablagen der JaZe-Anwendung werden im Wurzelzusammenhang aufbewahrt, da sie an fast allen Stellen der Anwendung verwendet werden.

Der kleine grüne Pfeil auf der rechten Seite bedeutet, dass die Projekt-Ablage die Quelle für einen Materialfluss ist. Der Erkunder kann so eingestellt werden, dass nur die Materialfluss-Piktogramme nur bei dem Zeug angezeigt wird, mit dem ein ausgewähltes Zeug in einer Materialfluss-Beziehung steht. So kann der Anwender erkennen, in welcher Abhängigkeitsbeziehung das Zeug zueinander steht. Diese Ansicht gibt jedoch nur einen Überblick.

Der Anwender möchte jedoch die Koordination des Projekt-Auflisters ändern. Er wählt ihn im Erkunder aus und wählt im Kontext-Menü die Aktion „Materialfluss bearbeiten". Es öffnet sich ein weiteres Werkzeug, das in der Abbildung 24 schematisch dargestellt ist, der *Materialfluß-Bearbeiter*. Dort wird automatisch all jenes Zeug angezeigt, das mit dem Projekt-Auflister in einer Materialfluss-Beziehung steht. Zeug wird dabei als Kasten angezeigt, in dem die Umgangsformen einblendbar sind.

Der Anwender kann über Drag and Drop weiteres Zeug aus dem Arbeitszusammenhang in den Materialfluss einbeziehen. Außerdem stehen Leisten mit Vorlagen für einen Materialfluss bereit. Die Projekt-Ablage ist farbig hinterlegt, was kenntlich machen soll, dass sie sich an einem anderen Ort befindet.

Der ursprüngliche Materialfluss ist recht einfach. Er verbindet die `alleProjekte()` -Funktion mit der `setzeListe()`-Operation des Projekt-Auflisters. Dies betrifft den Hinfluß. Wird der Materialfluss aktualisiert oder der Auflister fordert eine Aktualisierung an, so wird die gesetzte Quellfunktion gerufen und die Rückgabe der Funktion (die Liste mit allen Projekten) am Projekt-Auflister als Material gesetzt. Dieser zeigt daraufhin die Liste an.

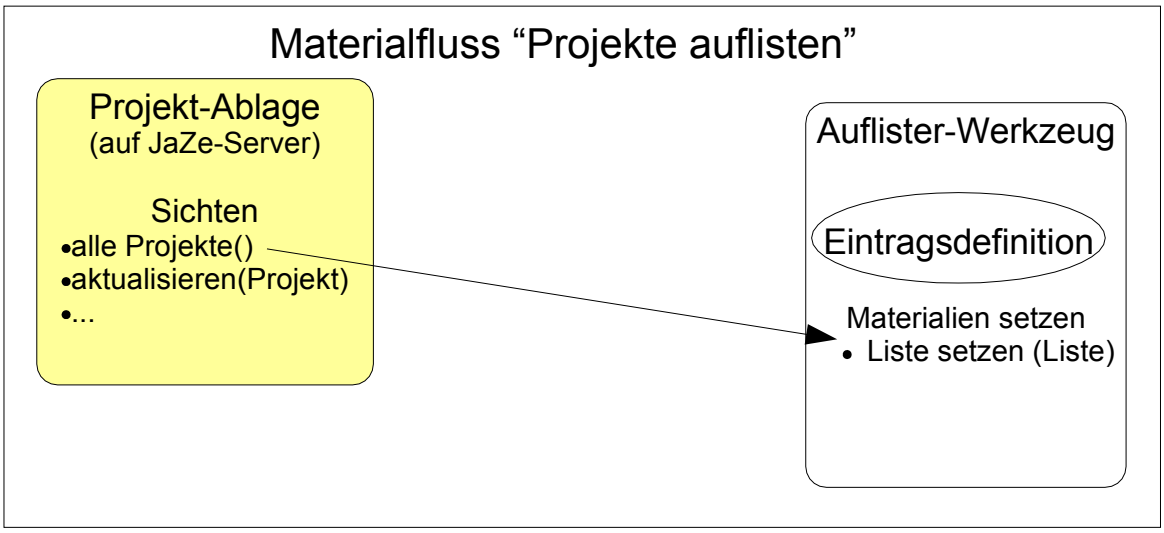

Abbildung 24: der ursprüngliche Materialfluss im Arbeitszusammenhang „Projekte verwalten”

7.2 Eine mögliche Anpassung durch Anwender

Der Anwender möchte nun die ursprünglich angezeigte große Liste in zwei kleinere Listen aufteilen. Die Grundidee ist folgende: Im Arbeitszusammenhang wird eine lokale Liste angelegt „abgeschlossene Projekte”. Diese Liste wird in einem zweiten Auflister angezeigt. Der Materialfluss zum ersten Auflister wird so verändert, dass er nur noch die Differenz-Menge zwischen allen existierenden Projekten und der Liste der abgeschlossenen Projekte transportiert.

Der Anwender beginnt damit, dass er im Erkunder über das Kontext-Menü eine neue *Liste* erzeugt und sie „abgeschlossene Projekte nennt” (siehe Abbildung 25). Danach öffnet er den bestehenden Materialfluss zum Projekt-Auflister im Materialfluss-Bearbeiter. Dort erzeugt er mit Hilfe von Vorlagen eine Differenzliste, fügt die Liste „abgeschlossene Projekte” dem Materialfluss hinzu und verbindet die einzelnen Elemente so, wie in Abbildung 26 gezeigt.

Die Differenzliste existiert nur innerhalb des Materialflusses, deshalb wird sie nicht im Erkunder angezeigt. Beim Verbinden der Liste „abgeschlossene Projekte” mit dem Auflister zeigt der Materialfluss-Bearbeiter eine Fehlermeldung an. Der Projekt-Auflister ist darauf eingestellt Projekte darzustellen, die Liste kann aber möglicherweise Zeug eines anderen Konzepts beinhalten. Der Materialfluss-Bearbeiter bietet daraufhin an, die Liste auf Zeug des Konzepts „Projekt” einzuschränken.

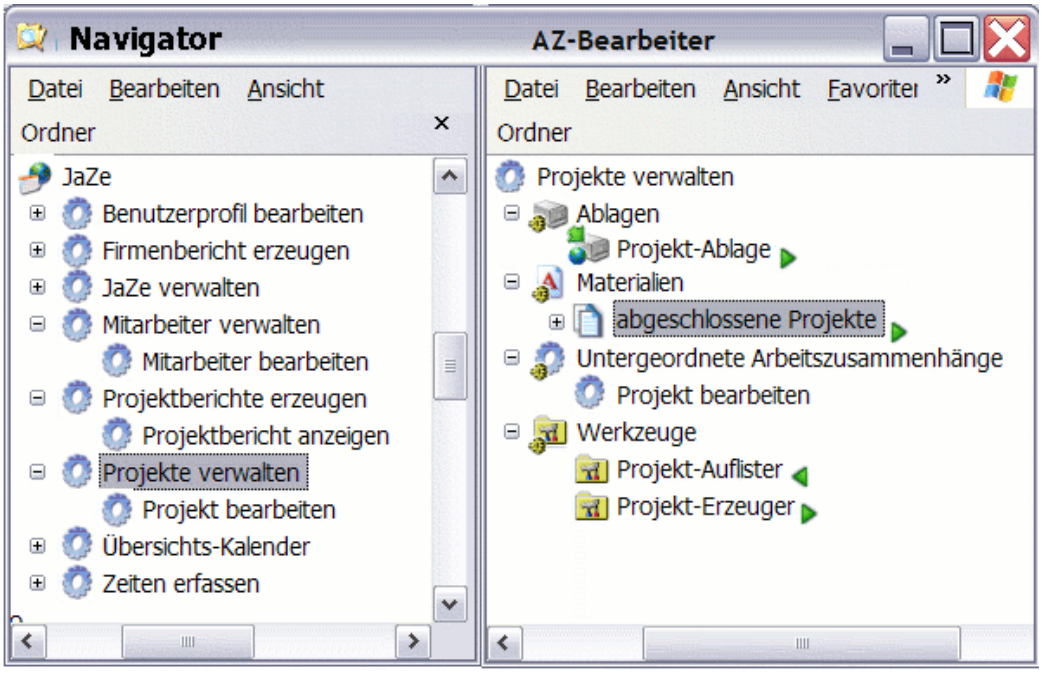

Abbildung 25: ein neuer Platzhalter "abgeschlossene Projekte" im Arbeitszusammenhang "Projekte verwalten"

Wird jetzt der Materialfluss aktualisiert, so wird von der Liste aller Projekte zuerst die Liste der abgeschlossenen Projekte abgezogen und erst dann im Projekt-Auflister „Aktive Projekte" angezeigt. Der Projekt-Auflister „Abgeschlossene Projekte" zeigt die Liste „abgeschlossene Projekte" an.

Der Anwender arrangiert die beiden Auflister an der Oberfläche so, dass sie schließlich untereinander stehen. Dabei bemerkt er, dass der Auflister „Abgeschlossene Projekte" keine Projekte anzeigt. Die Liste „abgeschlossene Projekte" ist ja auch noch leer. Um dies zu ändern, legt der Anwender einen Rückfluss vom Projekt-Auflister „Abgeschlossene Projekte" zur Liste „abgeschlossene Projekte" (siehe Abbildung 26). Dies hat zur Folge, dass Änderungen durch den Auflister an die Liste weiterpropagiert werden. Der Anwender kann jetzt per Drag and Drop Projekte vom Projekt-Auflister „Aktive Projekte" auf den Projekt-Auflister „Abgeschlossene Projekte" fallen lassen, um die Liste zu füllen.

Nachdem der Anwender den Arbeitszusammenhang gespeichert hat, ist die Anpassung erfolgreich durchgeführt worden. Im Arbeitszusammenhang befinden sich nun zwei Auflister, die jeweils die Liste der abgeschlossenen und die Liste der aktuellen (der nicht abgeschlossenen) Projekte anzeigen. Projekte können dadurch abgeschlossen werden, dass sie per Drag and Drop von dem ersten Auflister auf den zweiten Auflister gezogen werden und damit in die Liste der abgeschlossenen Projekte kopiert.

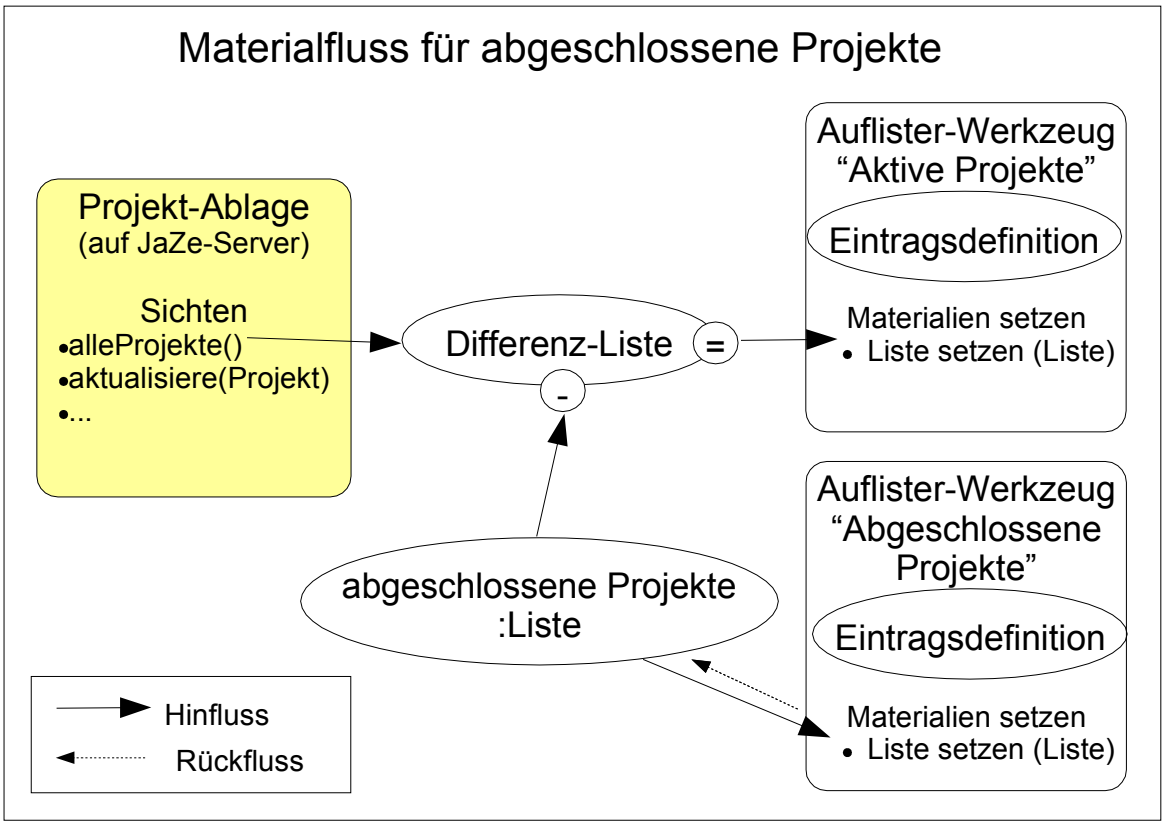

Abbildung 26: der durch den Anwender angepasste Materialfluss mit der lokalen Liste „abgeschlossene Projekte”

Um diese Anpassung für andere Anwender zugänglich zu machen, wäre es denkbar, die Liste „abgeschlossene Projekte” und die Differenz-Liste auf den Ort „JaZe-Server” zu verschieben und den Arbeitszusammenhang zum herunterladen auf den „Jaze-Server” hochzuladen.

7.3 Anpassung durch den Entwickler

Die von mir als Entwickler tatsächlich durchgeführten Änderungen unterscheiden sich von der eben vorgestellten im Wesentlichen dadurch, dass die Projekt-Ablage angepasst wurde. Nach der Änderung bietet die Projekt-Ablage zwei neue Funktionen, um auf den Inhalt zuzugreifen: `aktiveProjekte()` und `abgeschlosseneProjekte()` an. Der Arbeitszusammenhang wurde genau wie eben um ein zweites Werkzeug erweitert und die zugrundel iegenden Sichten entsprechend angepasst (siehe Abbildung 26).

Ein weiterer, nur am Rande relevanter Unterschied besteht darin, dass Projekte so abgeändert wurden, dass sie sich sperren lassen. Die Projekt-Ablage entscheidet anhand eines Projekts, in welche Liste es aufgenommen wird.

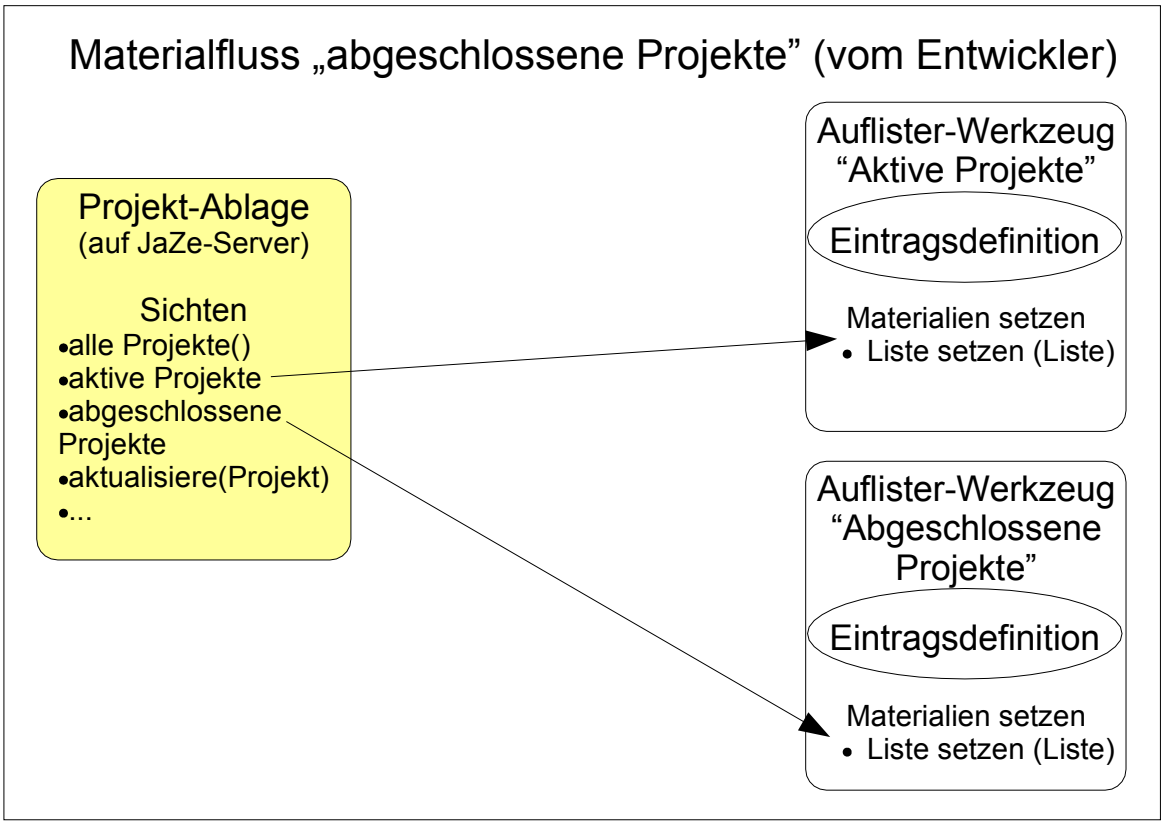

Abbildung 27: die Anpassung durch den Entwickler

7.4 Auswertung

Dieser Anwendungsfall zeigt ausgewählte Konzepte, die für eine Anpassung genutzt wurden. Bemerkenswert ist, dass sich die Anwendung nicht notwendigerweise anders präsentiert als ohne die Verwendung von vergegenständlichten Konzepten. Auf einen neuen Anwender, der nicht um vergegenständlichte Konzepte weiß, macht sie deshalb einen normalen Eindruck und lässt sich nicht schwieriger verwenden als andere Anwendungen ohne vergegenständlichte Konzepte.

Als Einstieg für eine konzeptionelle Betrachtung der Anwendung dient das Konzept des *Arbeitszusammenhangs.* Es wurden zwei Werkzeuge vorgestellt, die auf dem Arbeitszusammenhang arbeiten: den *Navigator* und den *Erkunder*. Beide Werkzeuge ermöglichen den Zugang zu einer Meta-Ebene. Diese Präsentation der Anwendung wird um eine konzeptionelle Sicht ergänzt. Sie stellt die grundlegenden Gegenstände der Arbeitssituation dar, die ohne den Erkunder nicht oder nur schwierig wahrzunehmen wären.

Beispielsweise setzt sich die Oberfläche einer Anwendung i.d.R. aus einzelnen Werkzeugen zusammen. Häufig sind die einzelnen Werkzeuge jedoch nur schwer zu unterscheiden. Der Erkunder bietet eine Darstellung, die die einzelnen Werkzeuge klar voneinander trennt und einzeln zugreifbar macht. Wählt der Anwender ein Werkzeug im Erkunder aus, so wird dieses in der Anwendung mit einem Rahmen markiert. Der Anwender kann auf diese Weise eine Verbindung zwischen dem sichtbaren Teil des Werkzeugs und seiner konzeptionellen Repräsentation im Erkunder ziehen. Möchte man ein bestimmtes Werkzeug

anpassen, so ermöglicht der Erkunder auf diese Weise ein Identifizieren und Auswählen. Dies hat deutliche Vorteile gegenüber dem üblichen Weg, den Anwendungen zum Anpassen anbieten: Über ein Menüpunkt „Einstellungen" bzw. „Optionen" gelangt der Anwender zu einem Dialog, der Anpassungen für jeden Aspekt der Anwendung ermöglicht. Dieser Dialog ist häufig nach willkürlich festgelegten Kriterien geordnet und kann sehr umfangreich sein. Den richtigen Punkt für die gewünschte Anpassung im Dialog zu finden, kann deshalb schwierig werden.

Darüber hinaus bietet der Erkunder Manipulationsmöglichkeiten, die ansonsten nur schwer zu realisieren wären. Im beschriebenen Anwendungsfall konnte der Anwender anhand der konzeptionellen Darstellung und mithilfe des Konzepts *Kopieren* ein neues Werkzeug erzeugen. Mit dem *Materialfluss-Bearbeiter* konnte dieses neue Werkzeug dann in den Arbeitszusammenhang integriert werden. Solche Anpassungen sind mit denen in Abschnitt 3.3.1 beschriebenen Gegenständen der Anpassung nicht ohne Weiteres möglich.

Abschließend möchte ich auf die Anpassung durch den Entwickler eingehen. Betrachtet man seine Anpassung haben den Materialfluss bzw. das Benutzungsmodell beinahe wieder so einfach gemacht, wie es in der Ausgangssituation beschrieben wurde. Hier ist ein stetiger Prozess denkbar, in dem die Anwender benötigte Anpassungen kurzfristig selbst vornehmen. Langfristig wird dieses dann vom Entwickler in das konkrete Zeug der Anwendung eingearbeitet. Dies sollte relativ problemlos möglich sein, da die Anpassungen der Anwender – über den Umweg der WAM-Konzepte - mit der WAM-Modellarchitektur konform sind. So erscheint es möglich, dass eine von den Anwendern regelmäßig angepasste Anwendung trotzdem langfristig softwaretechnisch beherrschbar bleiben kann.

Kapitel 8: Zusammenfassung und Ausblick

8.1 Problemstellung

Ausgangspunkt der vorliegenden Arbeit ist der Einsatz von Individual-Software zur Unterstützung von Büroarbeit. Die besonderen Eigenschaften von Büroarbeit und die Rahmenbedingungen für die Entwicklung von Individual-Software verschärfen ein grundsätzliches Problem von Computer-unterstützer Arbeit: *Kontrollverlust*. Eine menschliche Kernkompetenz, das flexible Anpassen der Arbeitsmittel an eine spezifische Aufgabenstellung, wird bei einer Unterstützung durch Software stark eingeschränkt. Dies ist sowohl aus Sicht des Arbeitgebers – der Arbeitnehmer kann seine Arbeit nicht mehr optimal erledigen – als auch mit Blick auf eine menschengerechte Arbeitsgestaltung bedenklich.

Einem Kontrollverlust kann mit dem Anbieten von Spielräumen in der Anwendung entgegengewirkt werden. Die Stufe von Kontrolle, die wieder das Anpassen der Arbeitsmittel an die Arbeitssituation ermöglicht, habe ich als *Gestaltbarkeit* bezeichnet.

8.2 Zielsetzung

Das Ziel dieser Arbeit war zu untersuchen, was unter gestaltbarer Software zu verstehen ist und einen Vorschlag zu erarbeiten, wie ihre Konstruktion unterstützt werden kann.

8.3 Ergebnisse

Wünschenswert wäre es gewesen, den erarbeiteten Lösungsvorschlag anhand einer Beispielanwendung softwaretechnisch umzusetzen und dann anschließend, anhand der Umsetzung, die Vor- und Nachteile zu diskutieren und zu bewerten. Dies ist im Rahmen der Bearbeitungszeit leider nicht möglich gewesen. Allerdings habe ich die notwendigen Grundlagen hierfür gelegt:

- **Wesentliche Begriffe sind geklärt:** *Büroarbeit*, *Kontrolle*, *Gestaltbarkeit*, *Anpassbarkeit*, *Orientierung, mentales Modell, Gedächtnisschema* und *Benutzungsmodell.*
- **Instrumente zum Unterstützen von Orientierung sind vorgestellt:** *konzeptionelle Modelle*, *Metaphern*, *Konsistenz* und *aktives Lernen*.
- **Eine Lösung wurde skizziert:** Der Begriff *Konzept* wurde von dem Begriff des Gedächtnisschemas abgeleitet. Vergegenständlicht in einem Anwendungs-Rahmenwerk unterstützen Konzepte konstruktiv das Einhalten von Regelhaftigkeit und damit *Konsistenz*. Gleichzeit ermöglichen sie es Anwendern, grundlegende Anpassungen durchzuführen. Konzepte von WAM-Anwendungen wurden identifiziert oder auf Basis von Konzeptions- und Entwurfswurfsmustern abgeleitet. Hervorzuheben ist dabei das Konzept des *Arbeitszusammenhangs*. Aufgrund seines einfachen hierarchischen Aufbaus verdeutlicht er auf einfache Weise die grundlegende Struktur einer WAM-Anwendung. Gleichzeitig ermöglicht er dem Anwender den Zugriff auf die benötigten

Arbeitsmittel. Schließlich habe ich im Rahmen eines Anwendungsfalls Oberflächenprototypen der Werkzeuge zum Bearbeiten des Arbeitszusammenhangs vorgestellt und gezeigt, dass auf Basis der genannte Konzepte weitergehende Anpassungen möglich sind, als die in Abschnitt 3.3.1 beschriebenen Anpassungen.

8.4 Ausblick

Für nachfolgende Arbeiten bietet sich ein weites Feld möglicher umsetzbarer Aspekte.

Werkzeug-Anpassungen

Auf der Hand liegt eine Umsetzung, de von mir in Abschnitt 6.5.2 beschriebenen anpassbaren Aktionen und Tastenkombinationen für Werkzeuge. Ausgewählte Gegenstände der Anpassung aus Abschnitt 3.3.1 könnten daraufhin untersucht werden, wie sich ihre Anpassbarkeit in das skizzierte Konzept einfügt.

Materialfluss

Das „Pipes and Filters”-Konzept von Unix ist ein beliebtes und mächtiges Konzept. Das Konzept *Materialfluss* ermöglicht ein ähnliches Prinzip für interaktive Anwendungen. Es kann auf weitere Anwendungsmöglichkeiten hin untersucht werden. Das Erstellen von Berichten beispielsweise lässt sich häufig mit Umsortieren und aufeinander Abbilden von Hierarchien, Listen und Tupeln beschreiben.

Subkonzepte von Werkzeug

Im Sprachgebrauch von WAM-Entwicklern fallen Begriffe auf, die sich als Subkonzepte von *Werkzeug* interpretieren lassen: Auflister, Bearbeiter, Anzeiger, Auswähler, usw. Es wäre interessant, bestehende WAM-Anwendungen daraufhin zu untersuchen, inwieweit sich die implementierten Werkzeuge in so ein Schema einordnen lassen.

Literaturverzeichnis

[Arbeitshandbuch 1997] Holger Luczak, Walter Volpert (Hrsg.), unter Mitarbeit von Thomas Müller: *Handbuch Arbeitswissenschaft*, Schäffel-Poeschel Verlag Stuttgart, 1997

[Beck und Andres 2004] Kent Beck und Dirk Andres: *Extreme Programming Explained*, Addision-Wesley Professional, 2004

[Bartlett 1932] F. C. Bartlett: *Remembering. A study in experimental and social psychology*, Cambridge: Cambridge University Press, 1932

[C1 WPS 2005] http://www.c1-wps.de/, zuletzt geprüft am 20.03.2005

[Carroll 1989] John M. Carroll: *An Overview of Minimalist Instruction*, IBM Research Division, Almaden, T.J. Waison, Tokyo, Zürich 1989

[Dix et al. 1998] Alan Dix, Janet Finlay, Gregory Abowd und Russell Beale: *Human-Computer Interaction*, 2. Ausgabe, London: Prentice Hall Europe, 1998

[Duden 2000] *Duden – Die deutsche Rechtschreibung*, Band 1, 22., völlig neu bearbeitete und erweiterte Auflage, Mannheim: Dudenverlag, 2000

[Duden Informatik 2001] *Duden Informatik – Ein Fachlexikon für Studium und Praxis*, 3. Auflage, Mannheim: Dudenverlag, 2001

[Dutke 1994] Stephan Dutke: *Mentale Modelle: Konstrukte des Wissens und Verstehens – Kognitionspsychologische Grundlagen der Software-Ergonomie*, Göttingen: Verlag für Angewandte Psychologie, 1994

[Eberleh, Oberquelle und Oppermann 1994] Edmund Eberleh, Horst Oberquelle und Reinhard Oppermann (Hrsg.): *Einführung in die Software-Ergonomie*, Berlin: Walter de Gruyter, 1994

[Floyd 1987] Christiane Floyd: *STEPS – eine Orientierung der Softwaretechnik auf sozialverträgliche Technikgestaltung*, Informatik Forum, Vol. 2, Nr 2

[Floyd 1997] Christiane Floyd: *Einführung in die Softwaretechnik – Scriptum zur gleichnamigen Vorlesung*, Universität Hamburg, 1997

[Floyd und Züllighoven 1999] Christiane Floyd und Heinz Züllighoven: *Softwaretechnik*, aus: P. Rechenberger und G. Pomberer (Hrsg.): *Informatik-Handbuch*, 2. Auflage, München: Carl Hanser Verlag, 1999

[Frese 1987] Michael Frese: *A Theory Of Control And Complexity: Implications For Software Design And Integration Of Computer Systems Into The Work Place*, aus: Michael Frese, Eberhard Ulich, Wolfgang Dzida (Hrsg.): *Psychological Issues Of Human-Computer Interaction In The Work Place*, Amsterdam: NHPC, 1987

[Friedrich 1995] Jürgen Friedrich: *Büro und Verwaltung*, 1995, aus: [Informatik und Gesellschaft 1995]

[Gamma et al. 1995] Erich Gamma, Richard Helm, Ralph Johnson und John Vlissides: *Design Patterns – Elements of Reusable Object-Oriented Software*, Boston: Addison-Wesley, 1995

[Gryczan 1995] Guido Gryczan: *Prozessmuster zur Unterstützung kooperativer Tätigkeit*, Wiesbaden: DUV, Deutscher Universitäts-Verlag, Dissertation an der Universität Hamburg, 1996

[Haaks 1991] Detlef Haaks: *Anpaßbare Informationssysteme – Auf dem Weg zu aufgaben- und benutzerorientierter Systemgestaltung und Funktionalität*, Dissertation am Fachbereich Informatik der Universität Hamburg, 1991

[Hacker 1994] Winfried Hacker: Arbeits- und organisationspsychologische Grundlagen der Software-Ergonomie, aus: [Eberleh, Oberquelle und Oppermann 1994]

[Heider 1958] F. Heider: *The psychology of interpersonal relations*, New York: Wiley, 1958

[Informatik und Gesellschaft 1995] Jürgen Friedrich, Thomas Herrmann, Max Peschek, Arno Rolf (Hrsg.): *Informatik und Gesellschaft*, Spektrum Akademischer Verlag: Heidelberg, 1995

[ISO 9241-10] ISO 9241-10: *Ergonomic requirements for office work with visual display terminals (VDTs) -- Part 10: Dialogue principles*, 1996

[JWAM 2005] http://www.jwam.de, zuletzt geprüft am 20.03.2005

[Kellog 1987] Wendy A. Kellogg: *Conceptual Consistency in the User Interface – Effects on User Performance*, aus: Hans-Jörg Bullinger und Brian Shackel, (Hrsg.): Proc. Human-Computer Interaction, INTERACT '87, Amsterdam: Elsevier Science, 1987

[Koch, Reiterer und Tjoa] Manfred Koch, Harald Reiterer und A Min Tjoa: S*oftware-Ergonomie - Gestalten von EDV-Systemen – Kriterien, Methoden und Werkzeuge*, Wien: Springer, 1991

[Luczak et al. 1989] H. Luczak, W. Volpert, A. Raeithel, W. Schwier unter Mitarbeit von Th. Müller und M. Rötting: *Arbeitswissenschaft, Kerndefinition – Gegenstandskatalog – Forschungsgebiete*, 3. Auflage, Eschborn: RKW, Köln: TÜV Rheinland, 1989

[Lehmann 1980] Meir M. Lehmann: *Programs, Life Cycles, and Laws of Software Evolution*, Proceedings of the IEEE, Vol. 68, No. 9, September 1980

[Maaß 1994] Susanne Maaß: *Transparenz – Eine zentrale Software-ergonomische Forderung*, Bericht FBI-HH-B-170/94 des Fachbereichs Informatik, Universität Hamburg, 1994

[Norman 1986] Donald A. Norman, Cognitive Engineering, aus: [Norman und Draper 1986]

[Norman und Draper 1986] Donald A. Norman, Stephen W. Draper (Editors): *User centered system design – New Perspectives on Human-Computer Interaction*, Lawrence Erlbaum Associates, Inc., Publishers, Hillsdale, New Jersey, London, 1986.

[Oberquelle 1986] Horst Oberquelle: *Adaption mit und durch den Benutzer an Stelle von automatischer Adaption*, Problembemangemessenheit von Benutzerschnittstellen und Anwendungssystemen, 6. Arbeitstagung Mensch-Maschine-Kommunikation, Burg Feuerstein bei Ebermannstadt, 1986

[Oberquelle 1994a] Horst Oberquelle: *Situationsbedingte und benutzerorientierte Anpaßbarkeit von Groupware*, aus: Anja Hartmann, Thomas Herrman, Markus Rhode und Volker Wulf (Hrsg.): *Menschengerechte Groupware – Software-Ergonomische Gestaltung und partizipative Umsetzung*, Berichte des German Chapters of the ACM, 1994

[Oberquelle 1994b] Horst Oberquelle: Formen der Mensch-Computer-Interaktion, aus: [Eberleh, Oberquelle und Oppermann 1994]

[Oberquelle 1997] Horst Oberquelle: *Informatik*, aus: [Arbeitshandbuch 1997]

[Oberquelle 1998] Horst Oberquelle: *Skript zur Vorlesung Software-Ergonomie II*, Universität Hamburg, 1998

[Oppermann 1994] Reinhard Oppermann: *Individualisierung von Benutzungsschnittstellen*, aus: [Eberleh, Oberquelle und Oppermann 1994]

[Pawson und Matthews 2002] Richard Pawson und Robert Matthews: *Naked Objects*, John Wiley and Sons Ltd., 2002, http://www.nakedobjects.org/, zuletzt überprüft am 20.03.2005

[Preece et al. 1994] Jenny Preece, Yvonne Rogers, Helen Sharp, David Benyon, Simon Holland und Tom Carey: *Human-Computer Interaction*, Reading, Massachusetts: Addison-Wesley, 1994

[Schmolitzky 2000] Axel Schmolitzky: *Ein Modell zur Trennung von Vererbung und Typabstraktion in objektorientierten Sprachen*, Dissertation, Aachen: Shaker Verlag, 2000

[Seligman 1974] M. E. P. Seligman: *Depression and learned helplessness*, in R. J. Friedmann, M. M. Katz (ed.): *The psychology of depression: Contemporary therory and research*, New York: Wiley, 1974

[Shneiderman 1998] Ben Shneiderman: *Designing the User Interface*, 3. Ausgabe, Reading, Massechusets: Addision-Wesley, 1998

[Sotograph 2005] http://www.software-tomography.com/html/sotograph.htm, zuletzt geprüft am 20.03.2005

[Stachowiak 1973] Herbert Stachowiak: *Allgemeine Modelltheorie*, Wien: Springer-Verlag, 1973

[Troy 1981] Norbert Troy: *Zur Bedeutung von Stresskontrolle – Experimentelle Untersuchungen über Arbeit unter Zeitdruck*, Dissertation, ETH Zürich, 1981

[Ulich 2001] Eberhard Ulich: *Arbeitspsychologie*, 5. vollständig überarbeitete Auflage, Hochschulverlag an der ETH Zürich: Zürich, Schäfer-Poesche: Stuttgart, 2001

[Weiß 1997] U. Weiß: *Konzeptionelle und technische Weiterentwicklung eines objektorientierten Frameworks nach der Werkzeug-Material-Metapher*. Diplomarbeit, Universität Hamburg, Fachbereich Informatik, Arbeitsbereich Softwaretechnik, 1997

[Züllighoven et al. 1998] Züllighoven, Heinz et al.: *Das objektorientierte Konstruktionshandbuch,* Heidelberg : dpunkt-Verlag, 1998