Diplomarbeit

Konzeption eines generischen Vorgehensmodells zur Entwicklung kollaborativer Applikationen und prototypische Implementierung am Beispiel einer J2EE-Plattform

Prof. Dr. L. Nastansky

Wintersemester 2004/05

Betreuer:
Dipl.-Wirt.-Inf. Ingo Erdmann

vorgelegt von:
Roland Zänger

Wirtschaftsinformatik
Matrikel Nr.: 6067732

Winfriedstraße 78
33098 Paderborn

Eidesstattliche Erklärung

Hiermit erkläre ich an Eides Statt, dass ich die vorliegende Arbeit selbständig und ohne unerlaubte Hilfe angefertigt, andere als die angegebenen Quellen und Hilfsmittel nicht benutzt und wörtlich oder inhaltlich entnommene Gedanken als solche kenntlich gemacht habe.

Die Arbeit wurde bisher keiner anderen Prüfungsbehörde vorgelegt und auch noch nicht veröffentlicht.

Paderborn, 28.03.2005 ……

Inhaltsverzeichnis

Abkürzungsverzeichnis

API	Application Programming Interface
CBSD	Component-Based Software Development
DBMS	Datenbank Management System
EIS	Enterprise Information System
EJB	Enterprise JavaBean
ERM	Entity-Relationship Modell
ERP	Enterprise Ressource Planning
GCC	Groupware Competence Center der Universität Paderborn
GUI	Graphical User Interface
HTML	Hypertext Markup Language
IBM	International Business Machines
IDE	Integrated Development Environment
J2EE	Java 2 Enterprise Edition
J2ME	Java 2 Micro Edition
J2SE	Java 2 Standard Edition
JDBC	Java Database Connectivity
JCP	Java Community Process
JNDI	Java Naming and Directory Interface
JRE	Java Runtime Environment
JSF	JavaServer Faces
JSP	JavaServer Pages
JSR	Java Specification Request
JVM	Java Virtual Machine
LDAP	Lightweight Directory Access Protocol
ML	Markup Language
MVC	Model-View-Controller
OMT	Object Modeling Technique
OOP	Objektorientierte Programmierung
PDA	Personal Digital Assistant

RAD	Rapid Application Development
RADD	Rapid Application Development and Deployment
RSA	Verschlüsselungsverfahren nach Rivest, Shamir und Adleman
RUP	Rational Unified Process
SDO	Service Data Object
SE	Software Engineering
UI	User Interface
UML	Unified Modeling Language
UP	Unified Process
WSAD	WebSphere Studio Application Developer
XML	eXtensible Markup Language
XP	eXtreme Programming

Abbildungsverzeichnis

Tabellenverzeichnis

1 Einleitung

1.1 Szenario

„Hypothesen sind Gerüste, die man vor dem Gebäude aufführt, und die man abträgt, wenn das Gebäude fertig ist. Sie sind dem Arbeiter unentbehrlich; er muß nur das Gerüst nicht für das Gebäude ansehen.“
Johann Wolfgang von Goethe

Auch zu Beginn des dritten Jahrtausends setzt sich die Durchdringung der gesamten Gesellschaft durch die Informationstechnologie weiter fort. Die neuen Techniken und die zahlreichen Möglichkeiten, die sich aus ihrer Verbreitung ergeben, haben in das tägliche Leben nicht nur Einzug gehalten, sondern sind, beispielsweise in der Form des Versands elektronischer Mail und der Beschaffung von Informationen aller Art mithilfe des Internets, zu einem festen Bestandteil des Tagesablaufs breiter Bevölkerungsschichten geworden. Voraussichtlich wird dieser Anteil in den kommenden Jahren weiter anwachsen und sich auch auf zusätzliche Bereiche des Alltags ausweiten.

Im Gegensatz dazu setzt die Wirtschafts- und Geschäftswelt bereits heute die Informations- und Kommunikationstechnologie in nahezu allen Bereichen ein. Einem ansteigenden Wettbewerbsdruck, der durch die Prozesse der Globalisierung und der zunehmenden Transparenz der Märke entsteht, muss durch geeignete Maßnahmen begegnet werden. Neben der Optimierung bestehender Geschäftprozesse und der Integration und Abstimmung des Qualitätsmanagements, müssen die Unternehmen die Mitarbeiter im Sinne einer zielgerichteten Kommunikation, Kooperation und Koordination weiter unterstützen, um so entscheidende Wettbewerbsvorteile für sich realisieren zu können. Auch die Zusammenarbeit von Mitarbeitern, die verteilt über verschiedene Standorte und möglicherweise innerhalb verschiedener Zeitzonen gemeinsam an einem Projekt arbeiten, veranlasst die Firmen, Applikationen zu entwickeln und einzusetzen, die diese orts- und zeitungebundene Zusammenarbeit in vielfältiger und effizienter Weise unterstützen. Je höher die Qualität dieser kollaborativen Applikationen und je

umfassender und zutreffender deren Unterstützung für die verschiedenen Mitarbeiter ausfällt, umso größer ist der Vorteil, der sich aus dem Einsatz dieser Applikationen für die einzelnen Unternehmen ergibt.

Die steigende Komplexität der kollaborativen Anwendungen, die wachsenden Ansprüche an das Qualitätsmanagement, die Einsparung von Unternehmensressourcen und die Planungssicherheit der durchzuführenden Projekte sind immer häufiger die Gründe für die Verwendung so genannter Vorgehensmodelle. Diese Modelle bieten eine Anleitung für die Realisierung von Softwareprojekten an. Vielfältige Vorgehens- und Tätigkeitsbeschreibungen helfen dabei den leitenden sowie den ausführenden Gruppen, sämtliche Phasen eines Entwicklungsprozesses erfolgreich zu durchlaufen und auf diesem Wege ein optimales Softwareprodukt zu verwirklichen.

1.2 Zielsetzung

Das Ziel der vorliegenden Ausarbeitung ist die Konzeption eines generischen Vorgehensmodells zur Entwicklung kollaborativer Applikationen sowie dessen prototypische Implementierung auf der Basis einer J2EE-Plattform. Das erarbeitete Modell soll die Vorgehensweise bei der Durchführung dieser speziellen Softwareprojekte, von den ersten Vorüberlegungen bis zur abschließenden Übergabe des kompletten Softwaresystems an den Auftraggeber, umfassend und zielgerichtet unterstützen.

Eine Vereinigung der positiven Aspekte bereits existierender Vorgehensmodelle soll unter der Berücksichtigung des besonderen Charakters kollaborativer Applikationen vollzogen werden. Darüber hinaus sollen die Einschränkungen und die Bereiche der Softwareerstellung, die von den bekannten Vorgehensmodellen nicht oder nicht hinreichend unterstützt werden, mithilfe entsprechender Vorgangsbeschreibungen des neuen Modells ausgeglichen werden.

Anhand des neuen Vorgehensmodells soll ein Prototyp einer klassischen kollaborativen Applikation auf der Grundlage der J2EE-Technologie erstellt werden. Dieser Prototyp soll sich hierbei leicht in eine schon existierende Systemumgebung integrieren lassen. Die Arbeitsschritte, die bei der prototypischen Implementierung durchzuführen sind, sollen innerhalb dieser Arbeit beispielhaft realisiert und dokumentiert werden.

1.3 Aufbau der Arbeit

Im nachfolgenden Kapitel wird das Thema der Ausarbeitung abgegrenzt und die Gesamtheit der grundlegenden Begriffe und Definitionen, die dem weiteren Verlauf dieser Arbeit als Basis dienen, erläutert und beschrieben. Zunächst wird das Verständnis der Gruppe der kollaborativen Applikationen, das hier zugrunde gelegt wird, charakterisiert. Von dieser Schilderung ausgehend werden insbesondere die Plattform der klassischen Groupware-Umgebung und die Plattform der J2EE-Technologie im Detail dargestellt. An dieser Stelle werden auch die grundlegenden technologischen Aspekte des praktischen Teils dieser Arbeit ausführlich beschrieben. Die anschließende Definition des Begriffs Softwareentwicklung bildet eine Überleitung zu einer Betrachtung bereits existierender Vorgehensmodelle für die Unterstützung von Arbeitsprozessen der Softwareentwicklung. Hierbei werden zunächst die Gruppen von Vorgehensmodellen vorgestellt, um darauf aufbauend einzelne Vorgehensmodelle ausführlicher zu betrachten.

Der danach folgende Abschnitt befasst sich mit der Konzeption eines neuen Vorgehensmodells, das insbesondere zur verbesserten Unterstützung der Entwicklung kollaborativer Applikationen dienen soll. Im ersten Schritt werden die speziellen Anforderungen herausgearbeitet, die diese Art von Anwendungen an ein Vorgehensmodell stellen. Anhand dieser vielfältigen Anforderungen werden die Eigenschaften existierender Vorgehensmodelle ausführlich diskutiert und gegeneinander abgewogen, um auf diese Weise in einem anschließenden Schritt eine möglichst optimale Lösung für die Entwicklung kollaborativer Applikationen in der Form eines neuen Vorgehensmodells zu definieren.

Das vierte Kapitel belegt den erfolgreichen Einsatz des zuvor erarbeiteten Vorgehensmodells. Einleitend werden die grundlegenden Funktionen und Merkmale der angestrebten Anwendung beschrieben. Als Vorlage für diese Applikation dient der Knowledge-Pool (K-Pool), der am Groupware Competence Center (GCC) der Universität Paderborn entwickelt und eingesetzt wird. Daraufhin wird das Durchlaufen sämtlicher Entwicklungsphasen des neuen Vorgehensmodells während der Realisierung dieses konkreten Beispiels einer kollaborativen Applikation exemplarisch dokumentiert. Abschließend wird die Vorgehensweise, die durch das neue Vorgehensmodell charakterisiert wird, von der typischen Arbeitsweise während

der Softwareentwicklung auf der Basis einer klassischen Groupware-Umgebung abgegrenzt.

Im fünften Kapitel werden die Möglichkeiten der Integration zusätzlicher Erweiterungen und der Durchführung weiterer Schritte zur Optimierung des erarbeiteten Konzepts dargestellt.

Eine Zusammenfassung der Ausarbeitung und der darin erzielten Ergebnisse liefert Kapitel sechs. Dieser Abschnitt bildet zusammen mit dem Verzeichnis der hier verwendeten Literatur den Abschluss dieser Arbeit.

Der Anhang gibt einen Überblick über die Hersteller, deren Produkte bei der Erstellung dieser Arbeit erwähnt beziehungsweise eingesetzt wurden. Darüber hinaus werden der Inhalt und die Verwendung der begleitenden CD-ROM erläutert.

2 Grundlagen und thematische Abgrenzung

Nach einer Beschreibung der besonderen Eigenschaften von kollaborativen Anwendungen werden in diesem Kapitel zwei besonders interessante Plattformen für Applikationen dieser Gruppe herausgegriffen und in ihren Eigenschaften erläutert. Darauf folgend wird eine ausführliche Betrachtung verschiedener Vorgehensmodelle zur Realisierung von Softwareprodukten durchgeführt.

2.1 Kollaborative Applikationen

Die Begriffsdefinition bezüglich kollaborativer Anwendungen ist in der Literatur nicht einheitlich.[1] Eine sehr allgemeine Definition dieses Ausdrucks beschreibt sie als Anwendungen, „die von verschiedenen Personen an verschiedenen Orten gleichzeitig benutzt werden können“.[2] Coleman schreibt, dass es sich bei kollaborativen Anwendungen im Allgemeinen um so genannte *Groupware* handelt. Als typische Anwendungsbereiche werden E-Mail, Gruppen-Kalender, Dokumenten-Management sowie Sprach- und Videokonferenzen genannt. In Colemans Definition von Kollaboration hebt dieser vor allem die Möglichkeit hervor, dass mehrere Benutzer mit einer ebenfalls unbestimmten Anzahl weiterer Benutzer interagieren können. Dies unterscheide sich zum Beispiel von der Kommunikation via E-Mail, da hierbei immer nur ein Anwender Initiator einer Aktivität sein könne.[3]

Diese Sichtweise lässt sich teilweise auch bei Nastansky wieder finden. Allerdings wird hier zunächst das *Share-Prinzip* als grundlegendes Paradigma für kollaborative Anwendungen beziehungsweise Groupware genannt. Dabei ist der Zugriff der Mitglieder einer Arbeitsgruppe auf einen gemeinsamen Datenbestand von zentraler Bedeutung. Das Share-Prinzip findet in Groupware-Applikationen Anwendung, denen das *Pull-Modell* zugrunde liegt. Sie ermöglichen es den Benutzern, individuell Informationen zu teilen, zu pflegen sowie deren Strukturierung zu verbessern. Darüber hinaus können diese Informationen in verschiedene Kontexte eingebettet und in wechselseitige Beziehungen gesetzt werden.[4] Nastansky weist im Gegensatz zu Coleman Programmen, die ausschließlich die Verwendung von E-Mail durch den Benutzer unterstützen, eine gesonderte Bedeutung zu. Sie folgen dem *Send-Prinzip*,

[1] Vgl. Dewan 2004.
[2] Vgl. Steinmetz 2001.
[3] Vgl. Coleman 1997 S. 39 sowie S. 52.
[4] Vgl. Nastansky 2000 S. 241.

welches dem *Push-Modell* als Grundlage dient. Bei dieser durch den Benutzer initiierten Art der Kommunikation werden die gesendeten Informationen in dem so genannten „store-and-forward"-Versand dem Empfänger zugestellt.[5] Folgt man diesen Definitionen von kollaborativen Anwendungen als Groupware, so kann bei Winkelmann dazu folgende Beschreibung gefunden werden: „Sie unterstützt die Gruppe als Ganzes durch Bereitstellung von zur Arbeit erforderlicher Information in aufbereiteter Form, ohne den einzelnen direkt zu steuern."[6] Hier lässt sich der Verweis auf den wahlfreien Zugriff auf arbeitsrelevante Informationen durch Gruppenmitglieder erkennen. Nach Winkelmann besteht Groupware vornehmlich aus

- Einem Mailsystem
- Einer Vielzahl intelligent organisierter Referenzdokumente mit betrieblichen Informationen, wie Handbücher, betriebliche Standards, Abkürzungsverzeichnisse und Nomenklaturen, Rundschreiben etc.
- Protokollanwendungen zu Ergebnissen von täglichen Aktivitäten wie Projektarbeit, Kundenarbeit etc., um den Bearbeitungsstand für alle betreffenden Mitarbeiter sichtbar zu machen und z.B. bei neuen Kundenkontakten das gesamte bisherige Wissen zu dem Vorgang schnell bereitzustellen. Derartige Anwendungen tragen zum „Wissensmanagement" (Knowledge Management) im Unternehmen bei, d.h. das Detailwissen einzelner Mitarbeiter allen verfügbar zu machen.[6]

Winkelmann bestätigt und konkretisiert somit die bereits definierten Bestandteile und Aufgaben von kollaborativen Anwendungen beziehungsweise Groupware. Als neuen Betrachtungspunkt führt Winkelmann deren Funktion des Wissensmanagements und des Wissenstransfers zwischen Mitarbeitern eines Unternehmens ein. Wagner beschreibt Wissensmanagement bezogen auf den Einsatz von Groupware als einen Transformationsprozess „des *individuellen* Wissens der Mitarbeiter zu einem *gemeinsamen* Wissen aller Mitarbeiter".[7] Als Ziel des Wissensmanagements definiert Nastansky „die Akkumulation von Wissen und die daraus entstehenden Wettbewerbsvorteile zu fördern".[8]

[5] Vgl. Nastansky 2000 S. 240 f.
[6] Vgl. Winkelmann 2001a S. 117.
[7] Vgl. Wagner 1995 S. 3.
[8] Vgl. Nastansky 2000 S. 258.

Hingegen sieht Lotus in kollaborative Anwendungen nur einen Bestandteil der Komponente *Kollaboration*, die gemeinsam mit den Komponenten *Kommunikation* und *Koordination* den Begriff der Groupware als Ganzes beschreibt.[9] Dabei definiert auch Lotus einen gemeinsamen Datenbestand, der flexibel genutzt werden kann, als die Grundlage dieser Applikationen. Das Spektrum möglicher Anwendungen reicht laut Lotus von einfachen Diskussionsplattformen, die nur über einen eingeschränkten Funktionsumfang verfügen, bis zu komplexen Datenbankanwendungen.

2.2 Plattformen kollaborativer Anwendungen

In diesem Abschnitt werden beispielhaft zwei der wichtigsten Plattformen von kollaborativen Anwendungen vorgestellt. Dabei wird insbesondere auf die unterschiedlichen Architekturen eingegangen, die der jeweiligen Plattform zugrunde liegen. Außerdem werden die unterschiedlichen Entwicklungs- und Ausführungsumgebungen beschrieben und voneinander abgegrenzt.

2.2.1 Klassische Groupware-Umgebung

Eine klassische Groupware-Umgebung basiert auf einer modernen Client-Server-Architektur und unterstützt die verteilte Datenhaltung (siehe Abschnitt 2.2.1.2).[10] Groupware-Umgebungen gehören der so genannten *Middleware* an. Diese befindet sich zwischen den Schichten der *Systemplattform*, die aus Systemhardware und Betriebssystem bestehen, und der Schicht der *Anwendungsprogramme*. Da Groupware-Umgebungen ihre Middleware-Dienste in verteilten und heterogenen Systemlandschaften anbieten, müssen sie sowohl unabhängig von den verwendeten Betriebssystemen, als auch von den eingesetzten Anwendungen arbeiten. Middleware ermöglicht so den Austausch von Informationen unterschiedlicher Komponenten eines Systems.[11] Die wesentlichen Bestandteile einer Groupware-Umgebung, insbesondere deren integrierte Entwicklungsumgebung für Datenbanken, werden nachfolgend genauer beschrieben und dienen als Grundlage für anschließende Kapitel.

[9] Vgl. Lotus 1995 S. 14 f.
[10] Vgl. Nastansky 2000 S 250.
[11] Vgl. Nastansky 2000 S. 257.

2.2.1.1 Verbunddokumente und Ansichten

Nach Nastansky bestehen die semi-strukturierten *Verbunddokumente* (engl. compound documents) einer Groupware-Umgebung aus verschiedenen so genannten *Container-Objekten*. Diese können sowohl strukturierte als auch unstrukturierte Informationen in sich tragen. Auf diese Weise lassen sich sowohl Datenfelder relational organisierter Umgebungen als auch Felder mit „weniger formalisierten, unstrukturierten oder multimedialen Datentypen sowie Prozessagenten“ realisieren. Dabei stellt ein *Item* die kleinste Einheit innerhalb eines Verbunddokuments dar. Es repräsentiert den Wert, der seinem Bezeichner zugewiesen ist.[12]

Verschiedene Items bilden zusammen ein *note-object*. Um die vielfältigen Informationen der note-objects dem Benutzer zur Verfügung zu stellen, werden sie mit einer *Maske* (engl. form) verknüpft. Die Felder einer Maske dienen der Präsentation der Inhalte korrespondierender items. Ein note-object, das auf diesem Weg mit einer Maske verbunden ist, wird als *Dokument* charakterisiert. Masken können sowohl vielfältige Darstellungsvorschriften und Felder als auch Programmlogik enthalten. Diese Darstellungsvorschriften können dabei in Abhängigkeit von dem Endgerät, das der Benutzer im Moment des Zugriffs einsetzt, gestaltet werden.[13] Um die Details eines Dokuments zu betrachten, öffnet der Anwender dieses üblicherweise aus einer so genannten *Ansicht* (engl. view) heraus. Diese Ansichten sind leistungsfähige Sichten auf eine beliebige Anzahl von Dokumenten unterschiedlicher Beschaffenheit. Sie können dabei die Auswahl der jeweils dargestellten Dokumente filtern oder in bestimmten Kategorien zusammenfassen. So können hinterlegte Informationen für den Benutzer übersichtlich und in aufbereiteter Form zur Verfügung gestellt werden.[14]

2.2.1.2 Verteilte Datenbanken und Replikation

Ein grundlegendes Konzept der Groupware-Umgebungen ist die Speicherung der Dokumente in Datenbanken, die ihrerseits auf Servern vorgehalten werden.[15] Da die Zugriffsmöglichkeiten der Benutzer auf eine Datenbank konfigurierbar sind, können auch verschiedene Anwender unabhängig voneinander eine Datenbank nutzen. Diese

[12] Vgl. Nastansky 2000 S. 252 ff., Lotus Development 2004a sowie Papows 1997 S. 348 ff.
[13] Vgl. Nastansky 2000 S. 252 ff., Lotus Development 2004.
[14] Vgl. Papows 1997 S. 348 f.
[15] Vgl. Papows 1997 S. 347.

Datenbanken grenzt Nastansky von relationalen Datenbanken ab: „Im Gegensatz zu relationalen Datenbanken, die in der Regel auf normalisierten Datenmodellen basieren und überwiegend zentralisierte Informationsverarbeitung mit hohen Anforderungen an die Konsistenz und Datenhaltung und –verarbeitung unterstützen, liegt Groupware-Datenbanken ein grundsätzlich andersartiges logisches und physikalisches Datenmodell zugrunde.“[16]

Nastansky schreibt weiter, dass diese Groupware-Datenbanken aus so genannten *Message-Objekten* bestehen. Diese zunächst voneinander unabhängigen Informationsträger können mittels Indizes strukturiert und administriert werden. Auf diese Weise können Message-Objekte untereinander logisch verknüpft werden. Die beschriebene Art der Datenhaltung ermöglicht eine wichtige Technologie des Datenabgleichs innerhalb verteilter Umgebungen, die so genannte *Replikation*.[17] Nach Papows dient die Replikation dazu, die Informationen untereinander so abzugleichen, dass alle Kopien oder Repliken (engl. replicas) einer Datenbank den identischen Inhalt aufweisen. Diese Datenkonsistenz soll ebenfalls hergestellt werden können, wenn beispielsweise auf Notebooks gespeicherte Datenbanken für einen Zeitraum nicht mit den Servern in Verbindung standen. Hierfür sind somit „intelligente Replikationsmechanismen und organisatorische Regelungen für Informationsallokation und –strukturierung (..) zentrale Lösungsanforderungen“[18], die es den Anwendern ermöglichen, Informationen orts- und zeitunabhängig aufzufinden, auszutauschen, auszuwerten und weiterzuverarbeiten, um flexibel „Entscheidungen zu treffen, kundenbezogene Transaktionen abzuwickeln oder Projektvorgänge zu initiieren“.[19]

2.2.1.3 Sicherheit

Arbeiten unterschiedliche Anwender mit gemeinsamen Datenbeständen, erhalten Sicherheits- und Zugriffskonzepte besonderes Gewicht, da in solchen Büroinformations- und Kommunikationssystemen auch wichtige interne Daten übertragen und gespeichert werden. Groupware-Umgebungen bieten zu diesem Zweck ein differenzierendes und vielschichtiges Zugriffskonzept, das es erlaubt,

[16] Vgl. Nastansky 2000 S. 250.
[17] Vgl. Nastansky 2000 S. 250, Nastansky 1998 S. 184 ff. und Papows 1997 S. 353.
[18] Vgl. Nastansky 2000 S. 250.
[19] Vgl. Lotus 2003, Nastansky 2000 S. 251 sowie Nastansky 1998 S. 195.

allen beteiligten Mitgliedern einer Arbeitsgruppe den abgestuften Zugriff auf die für sie relevanten Teilmengen der hinterlegten Informationen und Daten zu gewähren. Hierbei lassen sich die Zugriffe fein graduiert, von der Ebene vollständiger Datenbanken bis auf die Ebene einzelner Designelemente, definieren. Es besteht die Möglichkeit, „Personen abstrakten Organisations- und Struktureinheiten, so genannten Rollen oder Gruppen zuzuweisen, die wiederum unter- und übergeordnete Rollen und Gruppen beinhalten“ können.[20] Diese abstrakte Rollen- und Gruppenhierarchie gestattet eine realitätsnahe Abbildung vorhandener Organisationsstrukturen. Dadurch können Zugriffsregelungen für Benutzer realisiert werden, die sowohl abhängig von dem individuellen Benutzernamen als auch von der Zugehörigkeit des Benutzers zu definierten Gruppen beziehungsweise Rollen sind.[21] Diese Sicherheits- und Zugriffsmechanismen sind technisch in vielfältiger Weise realisiert. Das so genannte *Ein-Schlüssel-Verfahren* und das *Zwei-Schlüssel-Verfahren*, welches als *RSA-Verschlüsselung*[22] implementiert ist, bilden zusammen mit dem Einsatz digitaler Zertifikate die Grundlage der oben erläuterten Konzepte.[23]

2.2.1.4 RAD-Entwicklungsumgebung

Nach Nastansky sind die vielfältigen Anforderungen, die an Groupware-Systeme bezüglich ihres Nutzerkreises gestellt werden, ein wesentliches Abgrenzungsmerkmal gegenüber anderen Plattformen. So böten Groupware-Systeme „innovative Architekturkonzepte in Hinblick auf die Differenzierbarkeit von Datenmodellen und die Entwicklung von Be- und Verarbeitungsfunktionalitäten für spezifische Nutzergruppen“.[24] Um diese Anforderung zu erfüllen, wird eine leistungsfähige und flexible Entwicklungsumgebung benötigt.

Während der Programmiertätigkeiten werden Änderungen an der sichtbaren Benutzeroberfläche (engl. graphical user interface, GUI) wie zum Beispiel Masken, Ansichten etc. unmittelbar dem Entwickler dargestellt, ohne dass alle

[20] Vgl. Nastansky 2000 S. 255, Papows 1997 S. 354 sowie Lotus Development 2004b.
[21] Vgl. Nastansky 2000 S. 255.
[22] RSA-Verschlüsselung: Ein bekanntes asymmetrisches Verschlüsselungsverfahren, das 1978 am Massachussetts Institute of Technology von Ron Rivest, Adi Shamir und Leonard Adleman entwickelt wurde. Die RSA-Verschlüsselung ist ein Public-Key-Verfahren und wird auch zur digitalen Signatur eingesetzt. Sie beruht auf der Tatsache, dass es extrem aufwendig ist, zwei große Primzahlen p und q aus der alleinigen Kenntnis des Produkts $n = p \cdot q$ zurückzugewinnen. Zur Verschlüsselung genügt im Wesentlichen die Bekanntgabe von n, zur Entschlüsselung müssen beide Primzahlen p und q bekannt sein (…); zitiert nach Brockhaus 2003 S. 777.
[23] Vgl. Nastansky 2000 S. 255, Papows 1997 S. 354.
[24] Vgl. Nastansky 2000 S. 256.

Ereignisprozeduren vollständig definiert sein müssen.[25] Diese Arbeitsweise wird *Rapid Application Development* (RAD) genannt.[26] Folglich handelt es sich bei den integrierten Entwicklungsumgebungen klassischer Groupware-Plattformen um RAD-Entwicklungsumgebungen.[27] Papows erweitert diese Aussage und fasst die gesamte Groupware-Plattform zu einer *Rapid Application Development and Deployment Environment* (RADD) zusammen.[28] Die Gründe für diese umfassende Definition sollen im weiteren Text erläutert werden, sofern sie der genaueren Beschreibung der RAD-Entwicklungsumgebung dienen.

Zunächst führt Papows an, dass die grundlegenden Funktionen der Client-Server-Infrastruktur bereits vorhanden seien. Die Entwickler bräuchten sich nicht mit deren komplexer Implementierung zu beschäftigen. Auf diese Weise könnten die zugrunde liegenden Mechanismen der Speicherung von Informationen in Message-Objekten, der Replikation und der Nachrichtenübermittlung mit vergleichsweise geringem Aufwand in neue Applikationen integriert werden. Außerdem ständen den Anwendungsentwicklern die gesamte Sicherheitsarchitektur der Plattform sowie dessen Verzeichnisdienste unmittelbar zur Verfügung. Darüber hinaus böte die RAD-Entwicklungsumgebung neben dem Einsatz aktueller Programmiersprachen auch Möglichkeiten auf ein umfangreiches *Application Programming Interface* (API) zuzugreifen. Weiterhin könne mithilfe von RAD-Entwicklungsumgebungen das Programmierverfahren des *Rapid Prototypings* Anwendung finden.[29] Rapid Prototyping stellt dabei eine Lösung für die Tatsache dar, dass künftige Anwender ihre Wünsche und Anforderungen an eine Applikation erst artikulieren können, wenn das Programm bereits vorliegt. Aus diesem Grund wird ein kostengünstiger und kurzfristig modifizierbarer Prototyp in Rücksprache mit den späteren Anwendern entworfen, ohne dass das Design oder die Programmfunktionen vollständig ausprogrammiert werden. Dieser Prototyp kann den Bedürfnissen und Vorstellungen der Endanwender in kurzen Zyklen immer weiter angepasst werden. Das gewünschte Softwaresystem wird später nach der Vorlage des Prototyps realisiert.[30]

---

[25] Vgl. Lotus Development 1999 S. 13.
[26] Vgl. Papows 1997 S. 354, Winkelmann 2001b S. 275 und Riemann 2001 S. 342.
[27] Vgl. IBM 2004.
[28] Vgl. Papows 1997 S. 354.
[29] Vgl. Nastansky 2000 S. 256 f.
[30] Vgl. Raasch 1992 S. 17 ff., Riemann 2001 S. 341.

Papows führt darüber hinaus die zahlreichen Datenbank-Vorlagen (engl. templates) an, die gemeinsam mit klassischen Groupware-Umgebungen ausgeliefert werden.[31] Durch deren Modifikation und Erweiterung können die Entwickler effizient und zeitnah auf die individuellen Bedürfnisse bei der Informationsverarbeitung innerhalb heterogener Büro-Umgebungen eingehen. Nach Nastansky können auf diesem Weg Programmoberflächen und vollständige Applikationen realisiert werden, die sich vor allem an den Bedürfnissen der Endbenutzer orientieren und auf diesem Wege team-spezifische Arbeitsprozesse unterstützen.

2.2.2 Java 2 Enterprise Edition

In den folgenden Abschnitten wird die Plattform Java 2 Enterprise Edition (J2EE) dargestellt und erläutert. Insbesondere die Architektur und die speziellen Funktionen, die Java 2 Enterprise Edition auszeichnen, werden dabei genauer beschrieben.

Zunächst wird die objektorientierte Programmiersprache Java in ihren Eigenschaften vorgestellt. Daraufhin werden die spezifischen Merkmale von J2EE herausgearbeitet, die sich aus deren Einsatz im Umfeld eines Unternehmens ergeben. Das Model-View-Controller Konzept wird danach als ein grundlegendes Paradigma der Strukturierung vorgestellt. Nach einer Darstellung der J2EE-Mehrschichtarchitektur wird abschließend mit der Erläuterung der JavaServer Faces auf eine spezielle Technologie der J2EE-Plattform eingegangen, die insbesondere für den praktischen Anteil dieser Ausarbeitung von besonderer Bedeutung sein wird.

2.2.2.1 Java

Die objektorientierte Programmiersprache *Java* wurde im Jahr 1991 von der Firma Sun Microsystems im Rahmen des Green-Projekts entwickelt.[32] Einer der wesentlichen Vorteile von Java gegenüber anderen Programmiersprachen ist die Möglichkeit, Anwendungen plattformunabhängig ausführen zu können. Der Einsatz einer so genannten *Java Virtual Machine* (JVM), die speziell an das jeweilige Betriebssystem angepasst ist, ermöglicht diese hohe Flexibilität.

Entgegen herkömmlichen Programmübersetzungen, die den vorliegenden Programmcode jeweils für das entsprechende Betriebssystem übersetzen, ist der

[31] Vgl. Papows 1997 S. 354 ff. und Lotus Development 2004a.
[32] Vgl. Lemay 1999 S. 18.

Prozess des Kompilierens bei Java-Programmen in zwei separate Schritte aufgeteilt. Zunächst wird in einem ersten Schritt aus dem Quelltext der *Bytecode* erzeugt. Soll der Bytecode nun ausgeführt werden, dient die Java Virtual Machine des ausführenden Computers als lokaler Interpreter, der den allgemeingültigen Bytecode in plattformspezifische Befehle transformiert.[33]

Lemay sieht in der Tätigkeit der objektorientierten Softwareentwicklung eine Methode des Programmierens, die Computerprogramme als eine Menge interagierender Objekte charakterisiert. Darüber hinaus definiert Lemay die *objektorientierte Programmierung* (OOP) vor allem als eine Organisationsmethode von Programmen.[34] Schäffer folgt diesen Ansätzen und führt den Grundsatz der Objektorientierung weiter aus. Somit empfiehlt die OOP nach Schäffer:

- Ähnliche Strukturen zusammenzufassen,
- Daten mit den zugehörigen Grundoperationen zu Objekten zu kapseln,
- umfassende bzw. vollständige Schnittstellen zu bieten und
- stark aufeinander aufbauende Strukturen mittels Vererbung oder Aggregation auszudrücken.[35]

Da auch verschiedene Partner von Sun Microsystems maßgeblich an der Arbeit an den aktuellen Java-Spezifikationen beteiligt sind, wurde der *Java Community Process* (JCP)[36] etabliert. Der JCP dient dazu, die ständige Weiterentwicklung der Java-Plattform effizient und transparent zu gestalten. Dabei werden Vorschläge für neue Schnittstellen und Funktionen eingebracht, ausgearbeitet und gegebenenfalls in folgende Versionen der Java-Plattform aufgenommen. Auf diese Weise wird sichergestellt, dass der Spezifikationsprozess für die Mitarbeit möglichst vieler Anbieter offen zugänglich ist.[37]

Laut Schäffer hat Sun Microsystems mit der Benennung der Version 1.2 der Java-Spezifikation als Java 2 versucht, den erreichten Fortschritt und die Stabilität der Java-Plattform zu betonen. Zugleich wurde die Lizenzierung entsprechend der vielfältigen Einsatzmöglichkeiten angepasst.

---

[33] Vgl. Lemay 1999 S. 22 ff., Schäffer 2002 S. 23 ff.
[34] Vgl. Lemay 1999 S. 25.
[35] Vgl. Schäffer 2002 S. 24.
[36] Weitere Informationen zum Java Community Process können der Webseite „http://www.jcp.org/en/home/index“ entnommen werden.
[37] Schäffer 2002 S. 37 f.

Java 2 Micro Edition (J2ME) bildet die Entwicklungsplattform für so genannte Java enabled devices, wie zum Beispiel Personal Digital Assistants (PDA) oder moderne Mobiltelefone. *Java 2 Standard Edition* (J2SE) dient als Basis zur Entwicklung von Applets und Applikationen. J2SE verfügt über umfangreiche Bibliotheken zur Eingabe beziehungsweise Ausgabe von Daten sowie für die Modellierung grafischer Benutzeroberflächen. *Java 2 Enterprise Edition* umfasst den Bereich des unternehmensweiten Einsatzes von Anwendungen.[38] Im folgenden Abschnitt soll dieser umfangreiche Einsatzbereich weiter erläutert werden.

2.2.2.2 Einsatz im Unternehmensumfeld

Java 2 Enterprise Edition bezieht sich auf den Einsatz so genannter *Enterprise-Applikationen.* Es handelt sich hierbei um Java-Programme, die innerhalb von Unternehmen eingesetzt werden. Dieses Umfeld stellt vielfältige Anforderungen an die verwendete Software. Da in einer solchen Umgebung viele Mitarbeiter ihre Aufgaben parallel wahrnehmen, muss die gleichzeitige Nutzung der Applikation durch verschiedene Anwender möglich sein. Eine Enterprise-Applikation muss sich darüber hinaus auch in bestehende Strukturen einbetten lassen, um den Zugriff auf bereits vorhandene Systeme und Daten zu gewährleisten. Die ganzheitliche Abbildung von Unternehmensprozessen und Geschäftsabläufen ist eine weitere umfassende Anforderung. Nach Schäffer benötigen Enterprise-Applikationen, die insbesondere von Unternehmen des E-Business' eingesetzt werden, zusätzliche Eigenschaften. Diese ergeben sich daraus, dass alle Beteiligten eines Geschäftsprozesses mithilfe des Internets an dessen Bearbeitung direkt mitwirken. Dadurch wird laut Schäffer ein hoher Integrationsgrad aller beteiligten Unternehmen erreicht. Schäffer definiert Java 2 Enterprise Edition als eine Plattform, die sowohl „neue als auch bewährte Konzepte für die effiziente Implementierung von Unternehmensanwendungen mit aktuellen Technologien" realisiert.[39]

Haas liefert hierzu eine eher technisch geprägte Definition: „J2EE definiert ein *Programmier- und Architekturmodell* für wiederverwendbare, komponentenbasierte und verteilte Softwaresysteme mit in der Regel Web-basierter Benutzeroberfläche,

[38] Vgl. Turau 2001 S. 16, Schäffer 2002 S. 38. Weitere, die unterschiedlichen Bereiche der Java-Plattform betreffende Informationen können der Webseite „http://wwws.sun.com/software/learnabout/java/" entnommen werden.
[39] Vgl. Schäffer 2002 S. 19.

die einer Vielzahl von unterschiedlichen IT-Infrastrukturen mit minimalen Änderungen bereitgestellt und eingesetzt werden können.“[40]

Für Haas wird dieses Ziel durch fünf Kernkonzepte der J2EE realisiert. Einerseits böten *Enterprise-APIs*, wie zum Beispiel *Java Database Connectivity* (JDBC) für den Zugriff auf Datenbanken oder *Java Naming and Directory Interface* (JNDI) für die Nutzung von Namens- und Verzeichnisdiensten, einheitliche Client-Schnittstellen für die Nutzung heterogener Dienste und Ressourcen. So könnten nach Haas die implementierungsabhängigen Unterschiede abstrahiert werden. Als zweiter Punkt sei die Portierbarkeit zu nennen. *Portierbare Web-Komponenten* werden durch Servlet[41]- und JavaServer Pages[42] (JSP) -Technologie unterstützt. Das dritte Konzept sei nach Haas die *Trennung von Geschäftslogik und Middleware-Diensten*. Dies bedeute die exakte Unterscheidung des Entwurfs der Geschäftslogik und der benötigten Middleware-Dienste, wie dem Lastausgleich oder der Transaktionssteuerung durch das serverseitige Komponentenmodell der Enterprise JavaBeans[43] (EJB). Ein weiteres grundlegendes Konzept sei die *Spezialisierung und Rollendefinition*, die eine klare Aufteilung der Rollen bei dem Entwurf und der Bereitstellung von J2EE-basierten Softwaresystemen erlaube. Insgesamt liege J2EE durch den Einsatz verschiedener Deskriptoren ein leistungsfähiges Einsatz- und Bereitstellungsmodell zugrunde. Diese Deskriptoren basieren auf der Auszeichnungssprache (engl. Markup Language, ML) *eXtensible Markup Language* (XML) und erlauben es dem Benutzer, J2EE-Anwendungen zu konfigurieren.[44]

---

[40] Vgl. Haas 2002 S. 441.

[41] Bei Servlets handelt es sich um Java-Programme, die auf einem Server ausgeführt werden. Sie werden zur Erweiterung der Funktionalität eines Web-Servers eingesetzt, wobei sie dynamisch HTML-Code generieren können. Serverseitig kann auf die gesamte Funktionsvielfalt von J2EE zugegriffen werden (Vgl. Haas 2002, S. 241).

[42] JavaServer Pages (JSP) bestehen aus HTML-Code, Java-Programmcode und Tags. Im Web-Container des Servers werden sie in äquivalente Servlets umgewandelt und dann ausgeführt. JSPs erweitern die Servlet-Technologie und erlauben eine weitgehende Trennung von Seitenlayout und Programm-Logik (Vgl. Haas 2002, S. 5).

[43] Enterprise JavaBeans (EJB) sind nicht-visuelle Komponenten, die ihre Dienste über Schnittstellen anbieten und deren Eigenschaften (Sicherheit, Persistenz, Transaktionsverhalten) mittels XML konfiguriert werden. EJBs werden in Containern ausgeführt, die den Zugriff auf diese Dienste koordinieren. Diese Container sind in die Laufzeitumgebung von EJB-Servern eingebettet und organisieren das Management des Lebenszyklus’ von Objekten, die Lastverteilung und die Koordination von Systemressourcen (Vgl. Haas 2002, S. 5).

[44] Vgl. Haas 2002 S. 441. Weitere Informationen zu Java 2 Enterprise Edition können der Webseite „http://java.sun.com/j2ee/“ entnommen werden.

2.2.2.3 Model-View-Controller Konzept

Nach Haas empfiehlt J2EE die Verwendung des so genannten *Model-View-Controller* (MVC)-Paradigmas für die Entwicklung. Durch den Einsatz des Model-View-Controller Konzeptes wird die Abbildung komplexer Informationsflüsse innerhalb von Softwaresystemen durch eine klare Aufgaben- und Zuständigkeitsverteilung erleichtert.[45]

Das Modell (engl. *Model*) steht stellvertretend für die zentralen Entitäten beziehungsweise Objekte des vorliegenden Systems. Deren Zustände sind durch die aktuellen Attribute der Entitäten charakterisiert. Darüber hinaus repräsentiert das Modell alle zugrunde liegenden Daten und deren Anbindung an eine Datenquelle sowie sämtliche Verarbeitungsintelligenz der vorliegenden Geschäftsprozesse.

Die Sicht (engl. *View*) erfüllt die Aufgabe der visuellen Darstellung des Modells für den Benutzer (engl. User Interface, UI). Hierbei ist eine Vielzahl von Sichten auf ein Modell denkbar. Sichten können auch Elemente der Interaktion enthalten, die es dem Benutzer erlauben, auf die Abläufe der Applikation initiativ einzuwirken.

Der Koordinator (engl. *Controller*) bildet das Verbindungsglied zwischen dem Modell und der Sicht. Er steuert die Verarbeitung der Benutzereingaben, die von einer Sicht übermittelt werden, und initiiert resultierende Zustandsänderungen im Modell oder den Aufruf von Funktionen der im Modell implementierten Geschäftslogik. Eine Applikation kann über verschiedene Koordinatoren verfügen, die jeweils einem bestimmten Bereich zugeordnet sind.

Der Vorteil, der sich aus der Verwendung des MVC-Konzeptes ergibt, liegt vor allem in der klaren Strukturierung der unterschiedlichen Komponenten.[46] Diese liegen in sich geschlossen und mit explizit definierten Schnittstellen zu ihrer Umgebung vor. Die Module, die auf diese Weise gekapselt sind und somit voneinander vollkommen unabhängig agieren können, lassen sich komfortabel austauschen und wieder verwenden. Darüber hinaus ist der Zeitaufwand sowohl für die Installation als auch für die Wartung einer Anwendung mit diesem hohen Strukturierungsgrad wesentlich geringer, da die einzelnen Komponenten beliebig verändert oder ausgetauscht werden können, ohne die umgebenden Bausteine zu

[45] Vgl. Haas 2002 S. 445.
[46] Eine umfassende Definition von (Software-)Komponenten lässt sich bei Schryen 2001 auf den Seiten 36 bis 39 und bei Whitehead 2002 auf den Seiten 18 bis 35 nachlesen.

beeinflussen. Durch die Möglichkeit der parallelen Entwicklung der einzelnen Komponenten lässt sich zudem die Entwicklungszeit einer Applikation verkürzen.[47]

2.2.2.4 J2EE Mehrschichtarchitektur

Eine Mehrschichtarchitektur erweitert die verbreitete Zweischichtarchitektur. Zwischen dem Client und dem Datenbank-Server wird dabei ein Applikations-Server installiert, der in der Lage ist, verschieden Zugriffe parallel zu bearbeiten (engl. multithreaded). Auf diesem Server werden Enterprise JavaBeans gespeichert und die Programmintelligenz ausgeführt.[48]

Die erste Schicht, die als *Client Tier* bezeichnet wird, sieht drei unterschiedliche Zugriffsweisen auf die mittlere Schicht (*Middle Tier*) vor. Mithilfe einer Ausführungsumgebung für Java-Programme (engl. Java Runtime Environment, JRE), die auch als *Java Client* bezeichnet wird, können Anwender, durch eine Firewall nach außen abgesichert, lokale Java-Anwendungen ausführen. Diese Applikationen nutzen Enterprise JavaBeans in der zweiten Schicht und verschiedene Komponenten des Web-Containers, der seinerseits zum Beispiel Servlets oder JavaServer Pages enthalten kann. Als zweite Zugriffsmethode können Java-Anwendungen innerhalb des Intranets im *Web Client* genutzt werden. Dabei handelt es sich um die Programmausführung in einem Web-Browser, der ebenfalls über eine Ausführungsumgebung für Java-Programme verfügt. In diesem Fall sind die Java-Anwendungen meist in Dokumente der Auszeichnungssprache *Hypertext Markup Language* (HTML) als so genannte Applets (zusammengesetzt aus engl. „*Appl*ication snipp*et***"**) eingebettet. Die dritte Möglichkeit, Java-Programme auszuführen, unterscheidet sich von der zweiten Zugriffsmethode ausschließlich im Standpunkt des Anwenders. Während sich der Benutzer bei den Methoden eins und zwei innerhalb des Unternehmens befand, charakterisiert die Zugriffsart drei die Ausführung von Java-Programmen über das Internet. Damit kann der Benutzer ortsungebunden auf die Anwendungen des Unternehmens zugreifen.[49]

Die mittlere Schicht enthält die Geschäftsobjekte, die ihrerseits die Geschäftprozesse abbilden.[50] Eckel hebt insbesondere die eindeutige Strukturierung der Komponenten

[47] Vgl. IBM 2004c S. 2-10 ff., Crawford 2003 S. 38, Schäffer 2002 S. 63 f.
[48] Vgl. Austin 2000 S. 13.
[49] Vgl. IBM 2004c S. 2-17, Schäffer 2002 S. 28 f.
[50] Vgl. Schäffer 2002 S. 28.

hervor. So können diese Objekte aufgrund ihrer Kapselung in unterschiedlichen Applikationen eingesetzt werden.[51] Darüber hinaus stellt die mittlere Schicht diesen Objekten viele generische Dienste zur Verfügung, die es dem Entwickler erlauben, sich auf die anwendungsspezifischen Aufgaben zu konzentrieren. Zu diesen Diensten gehören beispielsweise „die Verwaltung des Lebenszyklus und die Sicherung der Persistenz (Speicherung) der Geschäftsobjekte, eine Umgebung für Transaktionen und die Kommunikation mit dem Client, dem Back-End und anderen Servern der Mittelschicht".[52] Nach Fields sind auch die automatisierte Lastverteilung, die Wahrung der Sicherheit und das gemeinsame Nutzen von Ressourcen im Allgemeinen als weitere Dienstleitungen zu nennen.[53]

Die Back-End- oder Server-Schicht wird auch als *Enterprise Information Systems* (EIS) *Tier* benannt. Diese dritte Schicht kann ältere Applikationen, die nicht dem Konzept der Mehrschichtarchitektur folgen, in die neue Umgebung integrieren. Daneben kann diese Schicht klassische *Datenbank Management Systeme* (DBMS) enthalten, die auf einem relationalen Datenbankmodell basieren. Auch die Anbindung komplexer Systeme zur unternehmensweiten Planung von unterschiedlichsten Ressourcen (engl. *Enterprise Ressource Planning*, ERP) kann in der dritten Schicht erfolgen.[54]

2.2.2.5 JavaServer Faces

Bei der Technologie der *JavaServer Faces* (JSF) handelt es sich um ein Framework zur Erzeugung von Benutzeroberflächen von Applikationen, die beispielsweise in einem Web-Browser ausgeführt werden können. Diese Technologie wird im Rahmen des *Java Community Process'* (JCP) unter dem *Java Specification Request* (JSR) 127 spezifiziert.[55] Nach Haiges beschreibt die Spezifikation APIs zur Darstellung von gekapselten Komponenten, die ihrerseits Elemente einer Benutzeroberfläche repräsentieren, sowie das Management deren interne Zustände. Als weitere Bestandteile dieser APIs definiert Haiges die programmatische Ereignisbehandlung, Mechanismen zur Validierung von Eingaben, die durch den Anwender

[51] Vgl. Eckel 1998 S. 672.
[52] Vgl. Schäffer 2002 S. 28.
[53] Vgl. Fields 2001 S. 276.
[54] Vgl. IBM 2004c S. 2-17.
[55] Vgl. Haiges 2003 S.1. Weiter Informationen zu dem Java Specification Request 127 kann der Webseite „http://www.jcp.org/en/jsr/detail?id=127" entnommen werden.

vorgenommen werden, und definierbare Navigationsregeln zwischen den unterschiedlichen Bildschirmseiten, die ein Anwender anfordern kann. Darüber hinaus unterstützen die APIs die Internationalisierung von Applikationen. Die Entwickler sollen so während der Programmierung mehrsprachiger Anwendungen entlastet werden. Als weiteren Bestandteil der Spezifikation führt Haiges die *JavaServer Pages Custom Tag Library* an, die es den Entwicklern erlaubt, JSF-Oberflächen in der Form von JSP-Seiten darstellen zu lassen. Haiges sieht den Schwerpunkt der JSF-Technologie auf der Gestaltung der Benutzeroberfläche.[56] Ähnlich hierzu beschreibt Armstrong das JSF-Framework und führt weiter aus, dass es sich dabei um Web-Applikationen handelt, die auf einem Java Server ausgeführt werden und deren Oberfläche dynamisch wiedergegeben (engl. *rendering*) wird. Nach Armstrong ist es möglich, wieder verwendbare und erweiterbare UI-Komponenten einer dargestellten Seite mit deren korrespondierenden Daten des Servers zu verknüpfen.[57]

Wahli erweitert diese Definitionen, indem er JSF als ein Rapid Application Development Framework bezeichnet. Durch die Verwendung von standardisierten und wieder verwendbaren UI-Komponenten sei es möglich, Rapid Prototyping als Vorgehensweise bei der Anwendungsentwicklung zu wählen. Diese erlaube es, das Oberflächendesign in kleinen Schritten zu erweitern und ständigen Tests zu unterziehen.[58] Des Weiteren vertritt Wahli die Meinung, dass die JSF-Spezifikation in ihrer Architektur das Model-View-Controller Konzept (siehe Abschnitt 2.2.2.3) zur Trennung der Darstellungs-, Datenhaltungs- und Koordinationskomponenten umsetzt. Abbildung 1 zeigt die Übertragung des MVC-Konzepts auf die JSF-Technologie.

Die Model-Komponente wird dabei durch so genannte *Backing-Beans* charakterisiert, die auch als *Managed Beans* bezeichnet werden. Es handelt sich zunächst um Java-Klassen, die dem bekannten Konzept der JavaBeans folgend ihre Eigenschaftswerte zur Laufzeit speichern und verändern können.[59] Diese JavaBeans werden nach der JSF-Spezifikation mit den UI-Komponenten verknüpft. Sie

---

[56] Vgl. Haiges 2003 S. 1.
[57] Vgl. Armstrong 2004 S. 647.
[58] Vgl. Wahli 2004 S. 6 f.
[59] Weitere Informationen zu JavaBeans können bei Duane 2001 S. 144 ff. und Amstrong 2004 S. 505 ff. nachgelesen werden.

speichern und verarbeiten deren Werte beziehungsweise die Werte der Instanzen, die von vorliegenden UI-Komponenten abgeleitet wurden. Backing-Beans können außerdem Methoden beinhalten, die die Realisierung von Eingabe-Validierungen, Ereignisverarbeitungen oder Navigationsregeln ermöglichen.[60] Die View-Komponente des MVC-Konzepts wird bei der Entwicklung von JSF-Applikationen mithilfe von JavaServer Pages realisiert. Diese werden mit den Daten der Model-Komponente kombiniert und gemeinsam mit den JSF-Bausteinen der Benutzeroberfläche zur Anzeige gebracht. Als Koordinator tritt innerhalb der JSF-Architektur ein *FacesServlet* auf. Dieses Servlet überwacht die Ereignisverarbeitung und übernimmt das Management der Objekte sowie die Weiterleitung nach den definierten Navigationsregeln (siehe Abbildung 1).[61]

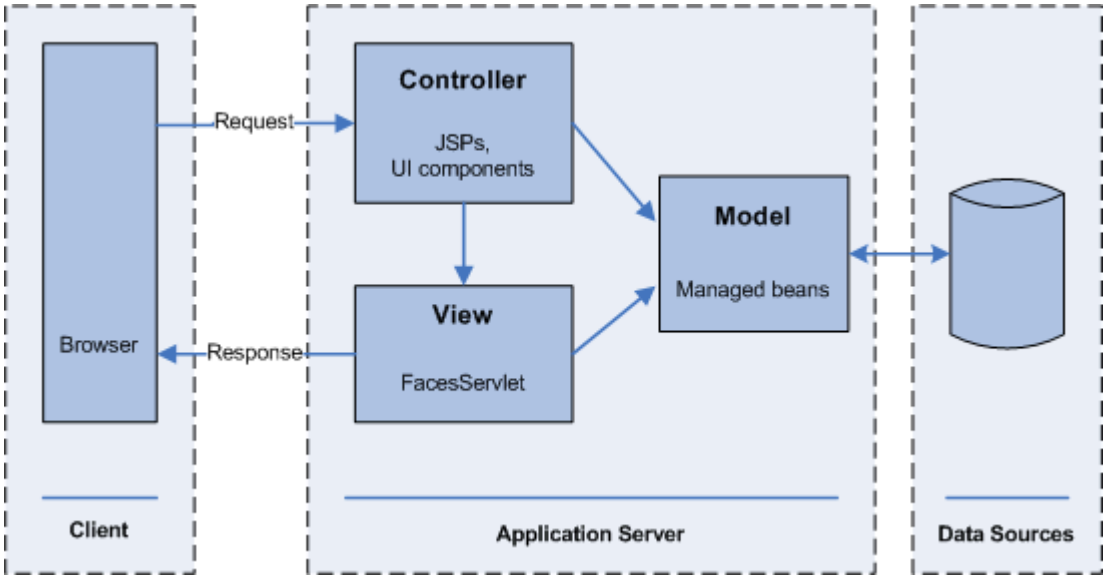


Abbildung 1 - MVC-Architektur der JSF-Technologie[62]

Einen großen Vorteil der JSF-Technologie sieht Haiges in deren unkomplizierten Einsatz. Dies liege darin begründet, dass die Vielzahl namhafter Hersteller, die an dem Spezifizierungsprozess beteiligt sind, untereinander im Wettbewerb stehe. Daraus resultiere eine Entwicklung hin zu immer intuitiveren und komfortableren Entwicklungsumgebungen für JSF-Anwendungen.[63] Diese Vielfalt sieht Haiges als sehr positiv im Bezug auf die Vermeidung so genannter *Vendor Locks*. Sie

[60] Vgl. Armstrong 2004 S. 674ff.
[61] Vgl. Wahli 2004 S. 5.
[62] Vgl. Erdmann 2003a S. 9, Wahli 2004 S. 5.
[63] Vgl. Haiges 2003 S. 2.

beschreiben die feste Bindung einer Technik an nur einen Hersteller und die negativen Effekte, die sich aus dieser Situation ergeben können.[64]

Die Flexibilität der JSF-Technologie bezüglich der eingesetzten Endgeräte stellt einen weiteren wichtigen Vorteil dar. UI-Komponenten können somit abhängig vom Endgerät unterschiedlich wiedergeben werden. Hierbei werden verschiedene so genannte *render kits* eingesetzt. Diese Renderer können unterschiedliche Darstellungsformen einer UI-Komponente in Abhängigkeit zu dem benutzten Endgerätes erzeugen.[65]

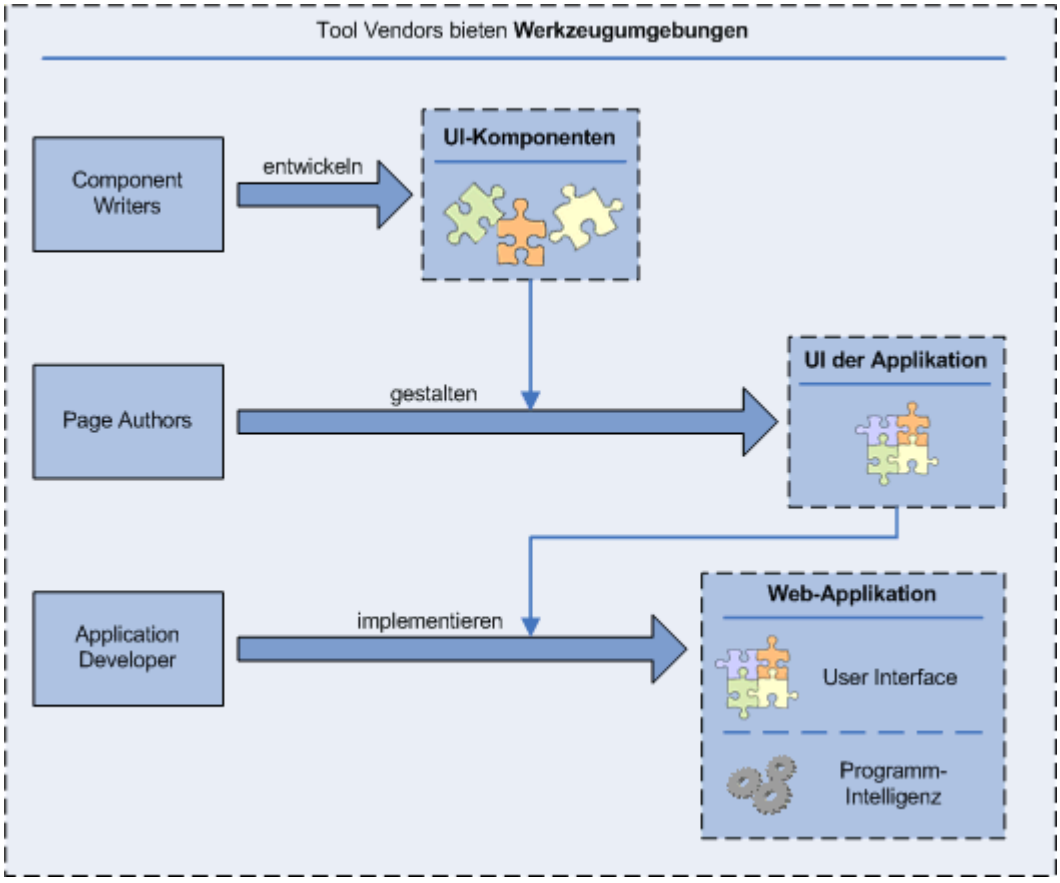


Abbildung 2 - Rollenkonzept des JSF-Frameworks

Eines der Hauptargumente für den Einsatz der JavaServer Faces ist die klare Rollenaufteilung, die durch die Trennung der Präsentations-Ebene von der Ebene der Programm-Logik ermöglicht wird. Auf diese Weise können die Mitarbeiter eines Teams sich auf ihren speziellen Bereich fokussieren.[66] Amstrong definiert vier unterschiedliche Rollen, deren Unterscheidung durch das JSF-Framework begünstigt

[64] Vgl. Haiges 2003 S. 5.
[65] Vgl. Haiges 2003 S. 3, Amstrong 2004 S. 665.
[66] Vgl. Armstrong 2004 S. 649.

wird. Die verschiedenen Rollen und Tätigkeitsfelder der *Page Authors, Application Developers, Component Writers* und *Tool Vendors* werden in der Abbildung 2 ausführlich dargestellt.[67]

Die Wiederverwendbarkeit der UI-Komponenten in beliebigen Applikationen und die Möglichkeit, diese um weitere Eigenschaften zu erweitern, stellen laut Kitain weitere Vorteile des JSF-Frameworks dar. So können die Entwicklung und die Wartung vorliegender Anwendungen vereinfacht und die Produktivität der Programmierer gesteigert werden.[68]

2.3 Softwareentwicklung

Sneed beschreibt *Softwareentwicklung* als einen nicht-linearen Prozess, der weder über einen definierten Anfang noch ein definiertes Ende verfügt. Vielmehr handle es sich dabei um einen endlosen Zyklus, der erst dann beendet ist, wenn die Software keine Verwendung mehr findet. Dies liege darin begründet, dass die Schnittstellen zwischen Softwaresystemen und ihren Benutzern viel zu komplex seien, als das diese eine unmittelbare Implementierung in nur einem Schritt zuließen. Diesen Zustand erklärt Sneed mit dem stetigen Wandel, dem die Arbeitsprozesse unterliegen, die mit Hilfe von Software automatisiert und somit erleichtert werden sollen. Darüber hinaus schafft die Automatisierung beziehungsweise die Programmierung eigene Probleme, die ebenfalls programmatisch gelöst werden müssen. Außerdem entsteht mit der Automatisierung auch vielfach erst ein differenzierendes Problembewusstsein und damit neue Anforderungen an eine Software. Da die zugrunde liegende Hardware einem hohen Entwicklungstempo unterliegt und Software-Systeme vielfach auch Schnittstellen zu anderen umgebenden Software-Systemen unterhalten, muss eine Software fortwährend den Veränderungen in ihrer Umgebung angepasst werden. Vor diesem Hintergrund könne die Entwicklung eines solchen Systems zwar temporären Unterbrechungen unterliegen, aber niemals als abgeschlossen gelten. Sneed vertritt somit die These, dass es irreführend sei, von der Wartung einer Software zu sprechen, da es sich dabei vielmehr um eine permanente Weiterentwicklung handle. Er definiert ferner ein quantitatives und ein qualitatives Ziel für die Softwareentwicklung. Einerseits sollen möglichst viele Funktionen in einem kurzen

[67] Vgl. Armstrong 2004 S. 651 f., Haiges 2003 S. 3.
[68] Vgl. Kitain o. J., Liu o. J.

Zeitraum mit einem geringen Aufwand integriert werden. Andererseits dürfe dabei die Qualität des Softwareprodukts nicht vernachlässigt werden.[69]

2.4 Vorgehensmodelle der Softwareentwicklung

Während nach Horn *Vorgehensmodelle* (engl. Software Process Model) die Arbeitsschritte der einzelnen Phasen der Softwareentwicklung mit deren Voraussetzungen (Vorbedingungen) und Ergebnissen (Nachbedingungen) detailliert darstellen, beschreiben diese Modelle nach Hesse „in idealisierender und von Details abstrahierender Weise den Software-Entwicklungsprozess, die dabei auszuführenden Tätigkeiten und die zu erbringenden Ergebnisse".[70] In den folgenden Abschnitten werden zunächst die Qualitätskriterien zur Beurteilung von Vorgehensmodellen erläutert. Daraufhin werden unterschiedliche Vorgehensweisen und Arbeitsmethoden zur Erstellung von Softwareprodukten, nach ihren gemeinsamen Eigenschaften gruppiert, vorgestellt.

2.4.1 Qualitäts- und Vergleichskriterien

Nach Bunse existieren sechs Merkmale, um Vorgehensmodelle voneinander bezüglich ihrer Qualität zu unterscheiden. Das Merkmal der *Vollständigkeit* beurteilt, ob ein Vorgehensmodell alle Phasen der Software-Entwicklung abbildet und die gesamten Tätigkeiten bezüglich des Projektmanagements, der Qualitätssicherung und des Konfigurationsmanagements erfasst. Das Kriterium der *Systematik* beschreibt die Verfügbarkeit einer einheitlichen Begriffswelt, die eine einheitliche und eindeutige Kommunikation zulässt. Die *Modularität* hingegen bezieht sich auf die Möglichkeit, kompakte Einheiten bilden zu können, die mittels präziser Vorgaben und eindeutig definierter Ergebnisse komfortabel plan- und überprüfbar sind. Das Qualitätskriterium der *Allgemeingültigkeit* bezieht sich auf die Skalierbarkeit des Vorgehensmodells und beurteilt somit dessen Eignung zu der Entwicklung von Systemen unterschiedlicher Größe. Die *Anpassbarkeit* charakterisiert die Flexibilität des Modells gegenüber organisationsspezifischen und technischen Besonderheiten. Inwieweit der Einsatz eines Vorgehensmodells bei der Softwareentwicklung durch

[69] Vgl. Sneed 1986 S. 24 ff.
[70] Vgl. Hesse 1992 S. 104 und Horn 1993 S. 21.

ein existierendes Werkzeug berücksichtigt beziehungsweise gefördert wird, beschreibt das Kriterium der *Werkzeugunterstützung*.[71]

Kneuper bemängelt, dass der überwiegende Teil der Vorgehensmodelle nur bezüglich des zeitlichen Aspekts strukturiert seien. Der Entwicklungsprozess sei nur in Phasen und Tätigkeiten unterteilt, die sequenziell oder teilweise parallel zu durchlaufen sind. Als Beispiel hierfür verweist Kneuper auf das klassische Wasserfallmodell (siehe hierzu Abschnitt 2.4.2). Für eine differenzierte Betrachtung seien weitere Dimensionen neben der zeitlichen notwendig. Als zusätzliche Dimension führt Kneuper neben der *Zeit* auch eine Komponente *Raum* ein. Die Definition der *Softwarearchitektur* als dritter Baustein ist damit begründet, dass die Vorgehensmodelle während bestimmter Phasen an Elemente der zugrunde liegenden Architektur, wie zum Beispiel vorhandene Komponenten, Subsysteme und Module, gebunden sein können. Als vierte Dimension definiert Kneuper die umgebende *Organisation*. An der Softwareentwicklung können unterschiedlichste, teilweise heterogene Gruppen beteiligt sein, die auch in dem jeweiligen Vorgehensmodell Berücksichtigung finden sollen.[72]

2.4.2 Phasen-, Wasserfall- und Schleifenmodelle

Während das Phasenmodell erstmals 1956 veröffentlicht wurde, publizierte W. W. Royce das Wasserfall- beziehungsweise das Schleifenmodell erstmalig im Jahre 1970.[73] Als Gemeinsamkeit der drei Modelle führt Bunse die Tatsache an, dass sämtliche Phasen der Modelle sequenziell abgearbeitet werden. Der Eintritt in die nachfolgende Phase setzt den Abschluss der vorhergehenden Phase voraus. Nach Bunse liefert jede Phase ein definiertes Ergebnis, zum Beispiel in Form von Anforderungs- oder Entwurfsdokumenten, die in nachfolgenden Arbeitsschritten weiterverarbeitet und vom Projektmanagement als Meilensteine verwendet werden. Auf diese Weise ließe sich nach Bunse der Projektfortschritt überprüfen. Die verschiedenen Ausprägungen von Vorgehensmodellen dieser Gruppe lassen sich fast ausnahmslos in sechs Phasen aufteilen. Die Anzahl der Arbeitsschritte und die jeweiligen Tätigkeiten können dabei um wenige Arbeitsschritte differieren

[71] Vgl. Bunse 2002 S. 103 f.
[72] Vgl. Kneuper 1998 S. 113 ff.
[73] Siehe hierzu Royce, W. W.: Managing the Development of Large Software Systems: Concepts and Techniques. Proc. IEEE WESCON, 1970.

beziehungsweise verschoben sein.[74] Bei den Phasen handelt es sich um den Abschnitt der *Analyse*, des *Entwurfs,* der *Implementierung,* der *Integration* und *Installation* und des *Einsatzes*.[75]

Als einen Vorteil dieser Vorgehensmodelle definiert Bunse die Tatsache, dass diese aufgrund ihrer klaren Struktur kontrollierte Iterationen zulassen. Die definierten Abschlussdokumente der einzelnen Phasen könnten so Maßnahmen zur Qualitätssicherung unterzogen werden. Gegebenenfalls könnten Phasen erneut durchlaufen werden, um einen Fehler unverzüglich zu beheben. Die konkreten Ausprägungen der Schleifen- und Wasserfallmodelle lassen teilweise auch weitere Rückschritte über den direkten Vorgänger hinaus zu. Ein weiterer Vorteil gegenüber anderen Vorgehensmodellen sei, dass es keine Notwendigkeit für ein aufwendiges Versionsmanagement gäbe. Jeder Durchlauf einer Phase des Modells erzeuge lediglich eine Version des jeweiligen Ergebnisses. Dazu könnten mögliche Rückschritte weitere Versionsnummern erzeugen, die aber in ihrer Anzahl insgesamt als vergleichsweise gering zu definieren sind.

Die Verwendung von Modellen dieser Gruppe setzt nach Bunse Erfahrungen mit den eingesetzten Entwicklungstechniken und mit der Abschätzung von Projektkosten voraus, da die streng sequenzielle Bearbeitung der Projektphasen das Nutzen von Erfahrungen innerhalb desselben Projektes stark erschwert. Modelle aus der Familie der Phasen-, Wasserfall- und Schleifenmodelle sollten nicht eingesetzt werden, wenn ein fester Auslieferungstermin für das Softwaresystem verpflichtend vorgegeben ist.[76] Um diese Nachteile zu vermeiden, wurden Modifikationen und Änderungen vorgeschlagen und erprobt. Die Alternativen, die sich teilweise daraus ergeben haben, sollen in den folgenden Abschnitten beschrieben werden.

2.4.3 Inkrementelle, iterative und evolutionäre Vorgehensmodelle

Nach Bunse besteht die Gemeinsamkeit aller Vorgehensmodelle dieser Gruppe in der Entwicklung von Software-Systemen in Inkrementen. Der gesamte

[74] Eine genaue Gegenüberstellung verschiedener Phasenmodelle kann bei Koslowski 1988 auf der Seite 129 nachgelesen werden.
[75] Vgl. Bunse 2002 S. 3 bis 7, Suhl 2002 S. 37 bis 39, Horn 1993 S. 14 f., Riemann 2001 S. 330 bis 339 und Pomberger 1996 S. 17 bis 20.
[76] Vgl. Bunse 2002 S. 5 f. Eine ausführliche Betrachtung der Vor- und Nachteile kann bei Pomberger 1993 auf der Seite 22 sowie bei Raasch 1992 auf der Seite 413 nachgelesen werden.

Anforderungskatalog wird dabei in kleinere Teilmengen zerlegt, für die alle Ergebnisse in sequenzieller Ordnung erarbeitet werden. Die Arbeitspakete zur Erzielung dieser Ergebnisse gleichen den Phasen der Phasen-, Wasserfall-, und Schleifenmodellen (siehe Abschnitt 2.4.2). Diese Pakete werden allerdings nicht Phasen genannt, da sie zu unterschiedlichen Zeitpunkten wiederholt durchlaufen werden. Ein Inkrement beinhaltet dabei alle erarbeiteten Ergebnisse einer Teilmenge. Liegen alle Ergebnisse eines Inkrements vor, wird die Teilmenge der Anforderungen erweitert und auf diese Weise ein weiteres Inkrement geschaffen. Folgeinkremente sind somit immer Erweiterungen bereits vorliegender Inkremente. Dieser wiederkehrende Prozess endet mit der Auslieferung des Softwaresystems an den Kunden. Bei Vorgehensmodellen dieser Gruppe sind Änderungen und Weiterentwicklungen des Softwaresystems fester Bestandteil des Entwicklungsprozesses. Wie bei den Phasen-, Wasserfall-, und Schleifenmodellen sind dabei Rückschritte in vorherige Arbeitspakete zulässig.[77]

Evolutionäre Vorgehensmodelle sind inhaltlich an die Modelle der inkrementellen und iterativen Vorgehensweisen angelehnt. Allerdings führen sie eine Risikoanalyse während der Definition der Anforderungen eines folgenden Inkrements durch. Das Spiralmodell von Boehm erfasst die gesamte Softwareentwicklung als einen risikogetriebenen Prozess. Auf diese Weise kann auf neue Erfahrungen und Erkenntnisse schon während der Projektdurchführung dynamisch reagiert werden. Der gesamte Vorgang der Softwareentwicklung wird dabei auf eine Spirale abgebildet. Jeder Umlauf der Spiral umfasst vier Quadranten. Die Ziele des jeweiligen Umlaufs und deren mögliche Alternativen werden innerhalb des ersten Quadranten identifiziert, um daraufhin im zweiten Quadranten bewertet zu werden. Die Bewertung umfasst auch das Aufdecken von Risikoquellen und gegebenenfalls die Definition von Gegenmaßnahmen. Der dritte Quadrant der Spirale charakterisiert die Implementierung und die Abnahme des jeweiligen Entwicklungsschritts. Der letzte Quadrant sieht die Planung des nächsten Zyklus vor.[78]

Nach Bunse haben inkrementelle, iterative und evolutionäre Vorgehensmodelle den Nachteil, dass sie durch die Vielzahl verschiedener Ergebnisse der unterschiedlichen

[77] Vgl. Bunse 2002 S. 11 bis S. 13.
[78] Weitere Informationen zu dem Vorgehensmodell von B. Boehm kann bei Boehm 1988 auf den Seiten 61 bis 72, bei Hesse 1992 auf den Seiten 72 bis 75 und bei Pomberger 1996 auf den Seiten 26 bis 28 nachgelesen werden.

Inkremente ein umfassendes Versionsmanagement erfordern. Die Anzahl der Versionen steigt zudem weiter, wenn innerhalb eines Inkrements Rückschritte durchgeführt werden müssen oder ein Ergebnis von unterschiedlichen Entwicklern bearbeitet wird. Außerdem kann der Arbeitsaufwand unvorhersehbar schnell ansteigen, wenn beispielsweise im Verlauf eines vorliegenden Inkrements Änderungen an den Anforderungen eines bereits abgeschlossenen Inkrements notwendig werden. In diesem Falle müssten schon abgeschlossene Inkremente erneut überarbeitet werden.

Der Einsatz von Vorgehensmodellen dieser Familie bietet sich nach Bunse an, wenn das zu entwickelnde System noch nicht detailliert beschrieben ist oder die Anforderungen nicht eindeutig formuliert sind. Während erste Inkremente erarbeitet werden, könnten weitere Teile der Anforderungen an folgende Inkremente konkretisiert werden. Als weiteren Vorteil erlaubt das inkrementelle Vorgehen einen langsamen und nachvollziehbaren Umstieg für die Anwender, die sich bereits früh in vorliegende Teile der Systemversion einarbeiten können. Darüber hinaus sieht Bunse bei diesen Vorgehensweisen ein geringeres Risiko einer Fehlentwicklung, da einzelne Inkremente bereits frühzeitig mit dem Auftraggeber besprochen werden können. Die Aufgliederung in Inkrementen erlaubt es außerdem, die Projektplanung basierend auf den bereits erzielten Inkrementen stetig anzupassen und zu verbessern.[79]

Nach Bunse existieren zwei wesentliche Faktoren, die den Einsatz dieser Vorgehensmodelle besonders beeinflussen. Einerseits müssen sich die Anforderungen leicht aufteilen lassen. Es dürfen also innerhalb des Anforderungskatalogs nur wenige Abhängigkeiten bestehen. Andererseits muss die Systemarchitektur, die verwendet wird, möglichst einfach zu erweitern sein, um die im Vorgehensmodell beschriebe Vorgehensweise befolgen zu können.

Bunse empfiehlt den Einsatz von inkrementellen, iterativen und evolutionären Vorgehensmodellen, wenn die Erfahrungen mit den eingesetzten Entwicklungstechniken als gering einzuschätzen sind. Mit steigender Anzahl vollendeter Inkremente steigt die Erfahrung der Entwickler. Diese Erfahrung kann in folgenden Inkrementen direkt genutzt und erweitert werden. Auch schlechte

[79] Vgl. Bunse 2001 S. 13 bis S. 15.

Erfahrungen zum Beispiel mit einer speziellen Technik können bereits im nächsten Inkrement vermieden werden. Des Weiteren rät Bunse zu Vorgehensweisen dieser Familie, wenn der Termin für die Auslieferung des Softwaresystems auf jeden Fall eingehalten werden muss. So könnten im Falle eines zeitlichen Verzugs fertige Inkremente ausgeliefert werden, während beispielsweise Inkremente weniger wichtiger Anforderungen entfallen oder nachgeliefert werden können.[80]

2.4.4 Prototypische Vorgehensmodelle

Die Vorgehensweise prototypischer Vorgehensmodelle ist an die der Phasen-, Wasserfall- und Schleifenmodelle (siehe Abschnitt 2.4.2) angelehnt. Sowohl die Reihenfolge der Phasen, als auch die Arbeitspakete, die während einzelner Phasen abgearbeitet werden, entsprechen den Arbeitsanweisungen der Familie der Phasen-, Wasserfall- und Schleifenmodelle. Zusätzlich werden bei prototypischen Vorgehensmodellen zu unterschiedlichen Zeitpunkten der Entwurf und die Entwicklung von Prototypen vorgeschrieben.[81]

Einen *Software-Prototyp* beschreibt Pomberger als ein Modell des geplanten Softwareprodukts. Dieses Modell wird mit geringem Aufwand erstellt und verfügt somit auch nicht über die gesamte Funktionsvielfalt, die für das Produkt vorgesehen ist. Prototypen seien aber soweit entwickelt, dass der Anwender wesentliche Eigenschaften des späteren Systems an dem Modell erproben und bewerten kann. Riemann sieht die Darstellung der geplanten Benutzeroberfläche und deren Handhabung als eine der Hauptaufgaben von Software-Prototypen. Allgemein ist die Aufgabe von Software-Prototypen „eine Überprüfung unklarer oder schwieriger Teile der Anforderungen durch den Kunden bereits sehr früh im Entwicklungsprozess.“[82] Durch die gewonnenen Ergebnisse können vorliegende Anforderungen angepasst und weiter konkretisiert werden.

Alle Arbeiten, die zur Erstellung eines Prototypens notwendig sind, werden nach Pomberger als *Prototyping* bezeichnet.[83] Dieser Herangehensweise liegt nach Budde

---

[80] Vgl. Bunse 2001 S. 14 f. Weitere Informationen zu evolutionären und inkrementellen Vorgehensmodellen können bei Balzert 2001 auf den Seiten 55 ff. sowie Raasch 1992 auf den Seiten 414 bis Seite 416 nachgelesen werden.
[81] Vgl. Bunse 2001 S. 7.
[82] Vgl. Bunse 2001 S. 8.
[83] Vgl. Pomberger 1996 S. 4 und Riemann 2001 S. 342. Weitere Informationen über die unterschiedlichen Formen des Prototyping können bei Koslowski 1988 auf den Seiten 166 bis 172 und bei Pomberger 1996 auf den Seiten 4 f. nachgelesen werden.

die Annahme zugrunde, dass es sich bei der Entwicklung von Softwareprodukten um einen Prozess der stetigen Weiterentwicklung handle. Hierbei diene das Experimentieren mit frühen Prototypen als Kommunikationsbasis für alle am Projekt beteiligten Interessensgruppen.[84]

Sowohl Riemann als auch Bunse sehen den Einsatz von Software-Prototypen vor allem für die Phasen der Analyse und des Entwurfs vor. Die Erkenntnisse und Erfahrungen, die hier gesammelt werden, könnten zu einer erneuten Ausführung bereits abgearbeiteten Phasen führen.

Durch die Verwendung von Analyse- und Entwurfsprototypen besitzen prototypische Vorgehensmodelle den Vorteil, dass die Prototypen als Diskussionsgrundlage für die beteiligten Projektgruppen eine konkretere und ausführlichere Anforderungsbeschreibung ermöglichen. Dadurch wird zudem das Risiko einer Fehlentwicklung deutlich minimiert. Prototypen bieten außerdem die Möglichkeit, die einzelnen Anforderungen zu priorisieren und auf diesem Wege Systemanforderungen zu identifizieren, die im Falle eines Zeitverzuges des Projekts nicht realisiert werden. Durch ihre Ähnlichkeit zu den Modellen der Phasen-, Wasserfall- und Schleifenmodelle erfordern prototypische Vorgehensmodelle ebenfalls kein komplexes Versionsmanagement. Neben den Ergebnissen der einzelnen Phasen müssen lediglich die Ergebnisse eventueller Rückschritte verwaltet werden. Nach Bunse könnten weitere Versionen entstehen, wenn mehrere Entwickler ein Ergebnis bearbeiten.[85]

Bunse empfiehlt den Einsatz von prototypischen Vorgehensmodellen insbesondere in den Fällen, in denen die Anforderungen beziehungsweise Teile der Anforderungen an das zu erstellende Softwaresystem nicht eindeutig formuliert sind. Die Visualisierung der zu erwartenden Funktionalität mithilfe von Prototypen fördert die klare Identifizierung der Anforderungen und der kritischen Bereiche einer Software. Auf diese Weise lassen sich Fehlentwicklungen früher erkennen und für den weiteren Projektverlauf vermeiden.

Auch die prototypischen Vorgehensmodelle setzen Erfahrungen der Entwickler mit der einzusetzenden Technologie voraus, da die sequenzielle Arbeitsweise die effiziente Nutzung gesammelter Erfahrungen früherer Phasen des Projektes nur bei

[84] Vgl. Budde 1992 S. 6 f.
[85] Vgl. Bunse 2001 S. 9.

Rückschritten ermöglicht. Darüber hinaus müssen die Entwickler erfahren im Umgang mit Technologien sein, die zur Erstellung von Prototypen verwendet werden können.[86]

Bunse rät von dem Einsatz prototypischer Vorgehensmodelle ab, wenn ein bindender Auslieferungstermin für das Software-System einzuhalten ist, da erst nach der vollständigen Abarbeitung aller Entwicklungsschritte und Projektphasen ein Endergebnis entsteht.

2.4.5 Vorgehensmodelle der objektorientierten Softwareentwicklung

Die bisher vorgestellten Gruppen von Vorgehensmodellen beschreiben Vorgehensweisen und Arbeitspakete bei der Entwicklung von Softwaresystemen unabhängig von möglichen Entwicklungsparadigmen. Der immer stärkere Einsatz von *objektorientierten Programmiertechniken* hat aber auch indirekten Einfluss auf den Aufbau und den Ablauf heutiger Softwareprojekte. In den folgenden Abschnitten werden die wichtigsten Vorgehensmodelle vorgestellt, die das objektorientierte Entwicklungsparadigma explizit berücksichtigen. Eine bedeutende Entwicklung innerhalb der objektorientierten Softwareentwicklung ist die *komponentenorientierte Softwareentwicklung* (engl. component-based software development, CBSD). Vorgehensmodelle, die diese spezielle Methodik unterstützen, werden abschließend erläutert.

2.4.5.1 Object Modeling Technique

Die *Object Modeling Technique*, die im folgenden Text verkürzend mit OMT bezeichnet wird, wurde von James Rumbaugh und dessen Mitarbeitern Anfang der 1990er Jahre entworfen und ist seit dem eins der am weitesten verbreiteten Vorgehensmodelle der objektorientierten Softwareentwicklung.[87] Nach Bunse ist das Ziel der OMT die Technik, die bisher zur Modellierung von Softwaresystemen benutzt wurde, auch im Umfeld der objektorientierten Entwicklung einzusetzen. OMT bezieht sich dabei vor allem auf drei unterschiedliche Diagrammtypen, die auch als Modelle bezeichnet werden. Die Modelle werden in der so genannten

[86] Vgl. Bunse 2001 S. 10.
[87] Vgl. Bunse 2001 S. 19.

Unified Modeling Language (UML)[88] beschrieben. Während das *Objektmodell* den statischen strukturellen Aufbau des späteren Systems in der Form von Klassen und Attributen beschreibt, verdeutlicht das *dynamische Modell* das zeitabhängige Verhalten der Software bezüglich konkreter Objekte als Zustandsautomat. Das *funktionale Modell* charakterisiert sämtliche Funktionen zur Datenverarbeitung.

Jedes dieser Modelle wird in drei Phasen unterteilt. Diese Phasen der *Analyse*, des *Entwurfs* und der *Implementierung* sind jeweils sequenziell zu durchlaufen. Die Phasen der drei Modelle sind untereinander durch einfache Konsistenzregeln verknüpft. Nach Bunse wird die Leistungsfähigkeit dieser Vorgehensweise vor allem während der Analyse deutlich. Durch die Verwendung verschiedener Diagramme wird diese Arbeit methodisch unterstützt. So ist beispielsweise zur Ermittlung der Anforderungen an ein Softwaresystem der Einsatz von Use-Case-Diagrammen[89] vorgesehen. Diese Diagramme erlauben es, typische Interaktionen zwischen zukünftigen Anwendern und dem geplanten System darzustellen. Bunse gibt zu bedenken, dass die beschriebene Unterstützung durch das Modell in den späteren Phasen eines Projektes nachlässt. So sehe die Entwurfsphase lediglich die Aufteilung des Systems in konzeptionelle Bausteine vor. Bei dem Entwurf moderner Umgebungen, die beispielsweise der Mehr-Schicht-Architektur (siehe hierzu Abschnitt 2.2.2.4) in ihrem Aufbau folgen, sind die Projektmitarbeiter auf ihre persönlichen Erfahrungen angewiesen. Bunse bemängelt ebenfalls, dass die Phase der Implementierung nur durch einige einfache Grundregeln unterstützt würde.[90]

2.4.5.2 Vorgehensmodell nach Booch

Das Vorgehensmodell wurde erstmal von Grady Booch 1991 veröffentlicht. Eine überarbeitete Version von 1994 ergänzte die objektorientierte Analyse und die Unterscheidung eines *Mikro-* und eines *Makroprozesses* zur Softwareentwicklung. Der Schwerpunkt dieses Vorgehensmodells liegt auf der Identifizierung von Objekten und Klassen des späteren Systems. Die Vorgehensweise des Modells vereint die Arbeitsweise der inkrementellen, iterativen und evolutionären

[88] Weitere Informationen zur Unified Modeling Language können bei Oestereich 2001auf den Seiten 191 bis 314 nachgelesen werden.
[89] Weitere Informationen zu Use-Case-Diagrammen können bei Oestereich 2001 auf den Seiten 196 bis 208 nachgelesen werden.
[90] Vgl. Bunse 2001 S. 19 bis S. 24. Weitere Informationen zu der Object Modeling Technique können bei Rumbaugh 1993 auf den Seiten 23 bis 25 nachgelesen werden.

Vorgehensmodelle (siehe Abschnitt 2.4.3) in Form des Mikroprozesses mit denen der Phasen-, Wasserfall- und Schleifenmodelle (siehe Abschnitt 2.4.2) als Makroprozess. Der Mikroprozess soll nach Bunse die Kreativität und ständige Innovation innerhalb des Entwicklungsprozesses sichern, indem die gesammelten Erfahrungen eines Durchlaufes in folgenden Iterationen eingebracht werden können. Der Mikroprozess bildet somit die tägliche Entwicklungsarbeit der Programmierer ab. Booch schreibt innerhalb dieses Mikroprozesses die Verwendung verschiedener Diagramme vor, die der stetigen Verbesserung und Erweiterung der Klassen- und Objektdefinitionen dienen. Der Makroprozess umfasst eine Vielzahl übergeordneter Arbeiten. Er dient vor allem als ein Kontrollgerüst für den Mikroprozess. Hier werden Maßnahmen zur Qualitätssicherung, Codeüberprüfung und Dokumentation beschrieben. Darüber hinaus werden die Risikoabschätzungen und die Festlegung von Meilensteinen des Projekts unterstützt. Neben den Phasen der *Analyse*, des *Entwurfs* und der *Evolution*, die jeweils über einen separaten Mikroprozess verfügen, besteht der Makroprozess zusätzlich aus einer vorgelagerten *Konzeptualisierung* und einer abschließenden Phase der *Wartung*.

Bunse bemängelt an diesem Vorgehensmodell, dass es nur wenig Anleitung zur konkreten Durchführung der Aktivitäten gibt. Des Weiteren sei die unüberschaubare Anzahl unterschiedlicher Diagramme, die während des Mikroprozesses verwendet werden sollen, problematisch, da nicht ausreichend erläutert wird, wie die Diagramme untereinander verknüpft sind. Auch Vorschriften bezüglich der Konsistenz zwischen den Diagrammen fehlten dem Vorgehensmodell. Darüber hinaus wird nach Bunse eine systematische Erstellung der Anforderungen sowie deren Implementierung kaum unterstützt. Vielmehr lägen die Vorteile des Booch-Modells in der objektorientierten Dokumentation der Anforderungen und dem objektorientierten Entwurf. Bunse hebt zusammenfassend die starke Betonung der Systemarchitektur in Form von Objektklassen hervor.[91]

2.4.5.3 Objectory-Vorgehensmodell

Das *Objectory-Vorgehensmodell* wurde im Jahre 1992 von Ivar Jacobson erarbeitet und zählt zusammen mit dem Vorgehensmodell nach Booch und dem OMT-Vorgehensmodell zu der ersten Generation objektorientierter Vorgehensmodelle. In

[91] Vgl. Bunse 2001 S. 26 bis S. 34. Weitere Informationen zur Booch-Methode können bei White 1994 nachgelesen werden.

diesem Modell nehmen Use-Case-Diagramme in Form eines so genannten Use-Case Modells für die Phasen der *Analyse*, der *Konstruktion* und des *Testens* eine besonders wichtige Rolle ein. Die Vorgehensweise des Objectory-Modells beschreibt einen inkrementellen und iterativen Prozess, der an das Spiralmodell von Boehm (siehe Abschnitt 2.4.3) angelehnt ist. So werden die drei Phasen in jedem Iterationsschritt unter stetig weiterentwickelten Anforderungen durchlaufen. Während der Phase der Analyse wird basierend auf einem bereits entworfenen Anforderungsdokument definiert, welche Einsatzmöglichkeiten und Funktionen das spätere System haben wird. Außerdem wird zu diesem Zeitpunkt die Struktur des Softwaresystems so bestimmt, dass dessen Bestandteile für den weiteren Projektverlauf eindeutig zu charakterisieren sind. Die Phase der Konstruktion umfasst die Entwicklung und die konkrete Umsetzung eines Entwurfmodells. Parallel zu der Phase der Konstruktion existiert die Phase der *Komponenten-Entwicklung*, in der neue Komponenten kreiert und bereits bestehende Bausteine modifiziert werden. Das Objectory-Modell definiert dabei Komponenten als bereits implementierte Teile des Systems, die aber in mehreren Systemen Anwendung finden. Die Testphase schließt sich an die Konstruktion an. Sie überprüft mithilfe dokumentierter Testfälle die korrekte Implementierung funktionaler und nichtfunktionaler Anforderungen sowohl einzelner Objektfunktionen als auch des gesamten Softwaresystems.

Das Objectory-Vorgehensmodell legt besonderen Wert auf die inhaltliche Verknüpfung der Analyse- und der Entwurfsmodelle, um die beiden Modellen direkt in Beziehung zueinander setzen zu können. Da der Ansatz des Objectory-Modells vor allem die Verwendung von Use-Case-Diagrammen vorsieht, unterstützt es besonders die Ermittlung funktionaler Anforderungen an das zu entwickelnde System. Für die Phasen des Entwurfs und der Konstruktion ist die methodische Unterstützung durch das Vorgehensmodell hingegen nicht ausreichend.[92]

2.4.5.4 Extreme Programming

Extreme Programming (XP) wurde Ende der 1990er Jahre von Kent Beck erstmals vorgestellt und gehört zu den populärsten Vorgehensmodellen der objektorientierten Programmierung. Die Vorgehensweise dieses Modells gliedert sich nach Bunse in zwei Phasen. Zunächst wird in dem ersten Schritt die *Planung* durchgeführt. Hierbei

[92] Vgl. Bunse 2001 S. 37 f. Weitere Informationen zu dem Objectory-Vorgehensmodell können bei Bunse 2001 auf den Seiten 39 bis 46 nachgelesen werden.

wird neben der Festlegung der Budget- und Zeitplanung auch eine detaillierte Analyse der Kunden- beziehungsweise der Benutzeranforderungen durchgeführt. Außerdem sieht das Modell die Entwicklung einer ersten prototypischen Systemarchitektur vor, die als Grundlage für die anschließenden Planungen der Implementierung dient.

So genannte *User-Stories* werden gemeinsam mit dem Kunden erstellt und beschreiben in kurzer Form die Erwartungen an die zu erstellende Applikation. Sie können mit der bekannten Technik der Use-Cases erweitert werden und dienen als Basis für die Testszenarien, die in der zweiten Phase des Vorgehensmodells ebenfalls von Bedeutung sind. Die zweite Phase wird nach Bunse als *Iterative Phase* bezeichnet und beschreibt die Realisierung des Systems in wiederkehrenden Zyklen. Zentraler Bestandteil der zweiten Phase ist die konsequente Erarbeitung und Durchführung von Testfällen. Bei der Entwicklung oder der Modifikation von Klassen entwirft das Programmierteam zunächst sämtliche Testfälle in Form von automatisierbaren *Unit Tests*.[93] Die Implementierung einer Programmieraufgabe gilt erst als vollständig abgeschlossen, wenn der erzeugte Programmcode alle entsprechenden Testfälle erfolgreich bewältigt.

Während der gesamten Programmierung überprüfen die Entwickler ständig, ob es eine Möglichkeit gibt, den Programmcode zu vereinfachen und gleichzeitig alle vorgegebenen Tests korrekt zu durchlaufen. Dieser Vorgang wird *Refactoring* genannt.[94] Die Programmrealisierung wird dabei von Programmierteams wahrgenommen, die jeweils aus zwei Entwicklern bestehen. Die Teams arbeiten gemeinsam an einem Computer. Diese Arbeitsweise wird als *Pair Programming* bezeichnet.[95] Neben den Unit Tests, die die Korrektheit in der Funktion und der Interaktion überprüfen, wird am Ende jeder Iteration das gesamte vorliegende Softwaresystem einem Akzeptanztest unterzogen. Hierzu werden die Tests, die in der ersten Phase erarbeitet wurden, durchlaufen. Die konkrete Ausprägung dieser Testfälle orientiert sich somit an den definierten User Stories. Wird ein Akzeptanztest nicht bestanden, so müssen betroffene Programmierteams das System

[93] Weitere Informationen zu Unit Tests können bei Wille 2001 auf den Seiten 304 bis 327 und bei Jacobson 1992 auf den Seiten 323 bis 330 nachgelesen werden.
[94] Vgl. Beck 2000 S. 58.
[95] Weitere Informationen zum Programmieren in Paaren können bei Beck 2000 auf der Seite 58 f. nachgelesen werden.

entsprechend überarbeiten. Als ein weiteres grundlegendes Prinzip von XP beschreibt Bunse die Möglichkeit, der Programmierteams sämtliche Teile des Softwaresystems modifizieren zu können. Ziel dieses Prinzips ist die Etablierung einer kollektiven Verantwortung für die Qualität des Quellcodes. Zusammenfassend definiert Bunse den Teamgedanken als das zentrale Moment des XP-Vorgehensmodells. Neben den technischen und methodischen Arbeitsanweisungen diskutiert dieses Vorgehensmodell verschiedene soziale Faktoren. Dazu gehören teambildende Maßnahmen, die bereits geschilderte gemeinsame Verantwortung für die Ergebnisse der Implementierung und die frühzeitige Beteiligung der Kunden an der Durchführung des Projekts. Das Pair Programming, die Festschreibung von mindestens zwei Wochen Urlaub pro Jahr und die Beschränkung der Wochenarbeitszeit auf 40 Stunden sind ebenfalls Faktoren, die die soziale Komponente des XP-Vorgehensmodells betonen.

Die Vorteile dieses Modells liegen nach Bunse in der guten Planbarkeit und Vorhersagbarkeit von XP-Projekten. Diese Eigenschaften liegen in der Kürze der einzelnen Entwicklungszyklen begründet. Darüber hinaus zeichnen sich mittels XP entwickelte Softwaresysteme durch eine einfache und effiziente Implementierung aus, da der Quellcode und die Architektur des Systems ständig optimiert werden.

Als Nachteil führt Bunse an, dass die ideale Teamgröße von Projekten, die XP als Vorgehensweise wählen, bei 12-14 Personen liegt. Diese Anzahl begrenze somit auch gleichzeitig die Größe des gesamten Projektes und damit auch den Umfang des zu realisierenden Systems. Außerdem erfordert XP von allen Beteiligten eines Projektes ein großes Maß an Disziplin, damit alle Prinzipien des Vorgehensmodells auch über den gesamten Projektverlauf Anwendung finden.

Nach Bunse wird Extreme Programming vor allem für Projekte als Vorgehensweise gewählt, die einen extrem kurzen Zeitraum für die Implementierung vorsehen und deren Anforderungen an das System möglicherweise Veränderungen unterliegen. Als Beispiel für den Einsatz des XP-Vorgehensmodells nennt Bunse die Entwicklung von Web-Applikationen.[96]

[96] Vgl. Bunse 2001 S. 55 bis S. 64 und Beck 2000 S. 53 bis S. 70 Weitere Informationen zu Extreme Programming können bei Beck 2000 ausführlich nachgelesen werden.

2.4.5.5 Catalysis

Die Vorteile der komponentenbasierten Softwareentwicklung führten nach Bunse zu einem Bedarf an Vorgehensmodellen, die unter anderem die Erstellung und Wiederverwendung von Softwarekomponenten unterstützen. Das *Catalysis-Vorgehensmodell* wurde 1998 von D'Souza und Wills vorgestellt. Zunächst war das Vorgehensmodell als eine Weiterentwicklung des OMT-Vorgehensmodells (siehe Abschnitt 2.4.5.1) geplant. Mit der zunehmenden Bedeutung der neuen Technologien entwickelte Catalysis sich zu „einem der ersten UML-basierten Vorgehensmodelle für die objektorientierte und komponentenbasierte Software-Entwicklung".[97]

Das Vorgehensmodell basiert auf drei grundlegenden Prinzipien. Während das Prinzip der *Abstraktion* vor allem die Fokussierung auf die bedeutenden Aspekte und das Erzeugen von eindeutigen Beschreibungen vorsieht, verdeutlicht das Prinzip der *Exaktheit* die Notwendigkeit, mögliche Inkonsistenzen frühzeitig zu vermeiden und die verwendeten abstrakten Modelle möglichst genau zu definieren. Das Prinzip der *Bausteine* beschreibt, dass alle Tätigkeiten der Wiederverwendung in den Bereichen der Architektur, des Designs und der Modelldefinition immer einen Vorteil darstellen und deshalb positiv zu bewerten seien.[98] Nach Bunse ist das Vorgehensmodell in vier Phasen aufgeteilt, deren Abfolge vorgegeben ist, während für die Tätigkeiten innerhalb dieser Phasen zunächst keine Reihenfolge verbindlich vorgeschrieben wird. Die Abfolge der Tätigkeiten wird abhängig von dem vorliegenden Projekt in so genannten *Catalysis Prozessmustern* definiert.

Im folgenden Text wird stellvertretend das Prozessmuster für Geschäftsinformationssysteme vorgestellt. Es umfasst die Realisierung einer Benutzerschnittstelle und einer Datenanbindung. Somit kann dieses Prozessmuster als ein typischer Vertreter von Standardapplikationen betrachtet werden. Alle Ergebnisse der vier Phasen werden durch Konsistenzregeln und eindeutige Abhängigkeiten zwischen den Phasen definiert. Auf diese Weise wird eine qualitative Überprüfung der Ergebnisse unterstützt. Die erste Phase der *Anforderungsanalyse* beschreibt die Ermittlung der Anforderungen an das zu erstellende Softwaresystem und liefert eine Beschreibung bezüglich der Funktionalität, der Produktqualität und der Systemumgebung. Die

[97] Vgl. Bunse 2001 S. 65.
[98] Vgl. D'Souza 1999 S. 37 bis S. 39.

Systemspezifikation bildet die zweite Phase. Hier werden basierend auf den erarbeiteten Anforderungen der ersten Phase mithilfe unterschiedlicher Diagrammtypen die technischen Gegebenheiten des Systems dargestellt. Auch die Modellierung der Benutzeroberfläche wird in dieser Phase durchgeführt. Der *Architekturentwurf* dient als dritte Phase des Catalysis-Vorgehensmodells der Vorbereitung der Implementierung des Systems. Die Architektur wird dabei unter zwei unterschiedlichen Aspekten definiert. Die Anwendungsarchitektur beschreibt den logischen Aufbau, während die physikalische Architektur die technische Zusammensetzung des geplanten Systems charakterisiert. Der anschließende *Komponentenentwurf* umfasst als letzte Phase des Vorgehensmodells die detaillierte Beschreibung der identifizierten Komponenten, deren Implementierung und Test. Als grundlegendes Prinzip des Catalysis-Vorgehensmodells sieht Bunse die iterativ-inkrementelle Vorgehensweise, die es erlaubt, gewonnene Erkenntnisse aus früheren Inkrementen in vorliegende Prozesse einfließen zu lassen.[99]

Bunse bemängelt an dem Vorgehensmodell, dass aufgrund dessen Komplexität Probleme bei dem Erlernen und der Einführung auftreten könnten. Darüber hinaus sei die Verwendung von Prozessmustern zur Definition konkreter Arbeitsabläufe durchaus problematisch, da bisher nur wenige Muster existierten. Nach Bunse enthält Catalysis eine Vielzahl an Techniken und Beschreibungsformen, die viel Erfahrung bei den Projektmanagern und den Entwicklern voraussetze. Unter dieser Bedingung liefere das Catalysis-Vorgehensmodell allerdings qualitativ hochwertige Softwaresysteme.[100]

2.4.5.6 Unified Process

Im Jahr 1998 stellten Ivar Jacobson, James Rumbaugh und Grady Booch *Unified Process* erstmals vor. Nach Bunse ist die Grundlage des Vorgehensmodells der Rational Objectory Process, welcher seinerseits von dem Objectory-Vorgehensmodell (siehe Abschnitt 2.4.5.3) abgeleitet wurde. Dabei führe Rational Objectory Process den Use-Case-basierten Ansatz des Objectory-Vorgehensmodells mit dem Ansatz der Booch-Methode (siehe Abschnitt 2.4.5.2) zusammen, welcher

[99] Vgl. Bunse 2001 S. 65 bis S. 72. Weitere Informationen zu dem Catalysis-Vorgehensmodell lassen sich bei D'Souza 1999 nachlesen beziehungsweise der Webseite „http://www.catalysis.org" entnehmen.
[100] Vgl. Bunse 2001 S. 67.

vor allem die Entwicklung der Systemarchitektur unterstützt. Unified Process erweitert den Rational Objectory Process um Tätigkeiten wie Implementierung, Test, Anforderungs- und Konfigurationsmanagement. Unified Process ist somit nach Bunse „Use-Case-getrieben, architekturzentriert, iterativ und inkrementell“.[101] Bunse begründet dies damit, dass in Anlehnung an die Objectory-Vorgehensweise Use-Cases sowohl für die Ermittlung der funktionalen Anforderungen als auch als Grundlage zur Definition von Entwurfs- und Implementierungsmodellen dienen. Die besondere Rolle der Systemarchitektur habe Unified Process von der Booch-Methode übernommen. Inkrementell und iterativ sei das Vorgehen dieses Vorgehensmodells, da große Softwaresysteme in viele so genannte Mini-Projekte zerlegt und in kleinen Schritten erarbeitet werden.

Bei Unified Process handelt es sich um ein komponentenbasiertes Vorgehensmodell, da das Zielsystem aus einzelnen Softwarekomponenten erstellt wird, die über eindeutig definierte Schnittstellen miteinander verknüpft werden. Einen Zyklus innerhalb des Modells definiert Jacobson als eine Abfolge der vier Phasen *Beginn*, *Ausarbeitung*, *Konstruktion* und *Übergang*. Jede dieser vier Phasen enthält wiederum die Arbeitsschritte *Anforderungsermittlung*, *Analyse*, *Entwurf*, *Implementierung* und *Test*, deren jeweilige Bedeutung von dem gesamten Projektfortschritt und der gerade vorliegenden Phase abhängt. Diese Arbeitsschritte werden in Iterationen durchlaufen. Ist eine Phase vollständig abgearbeitet, werden anhand dieses Meilensteins die Planungen und Entscheidungen für die folgende Phase getroffen. Sind alle vier Phasen einmal durchlaufen, wird das so erarbeitete Produkt an den Kunden ausgeliefert und ein neuer Zyklus eingeleitet.[102] Für jedes Arbeitspaket sieht Unified Process die eindeutige Definition von Eingangs- und Ergebnisdokumenten vor. Darüber hinaus empfiehlt dieses Modell die Definition unterschiedlicher Entwicklerrollen, die die verschiedenen Tätigkeiten in den entsprechenden Phasen wahrnehmen. Bunse bemängelt an Unified Process vor allem die fehlende methodische Unterstützung der konkreten Entwicklungstätigkeit. Hier würden die speziellen Eigenschaften unterschiedlicher Systemarchitekturen zu wenig Berücksichtigung finden. Dies liegt nach Bunse darin begründet, dass es sich bei Unified Process lediglich um ein Rahmenwerk handelt. Diese Tatsache führt auch

[101] Vgl. Bunse 2001 S. 76.
[102] Vgl. Jacobson 1999 S.11 und Bunse 2001 S. 77.

dazu, dass vor Projektbeginn umfangreiche Anpassungen des Vorgehensmodells an die vorliegende Situation vorgenommen werden müssen. Nach Bunse existiert ebenfalls eine Variante von Unified Process, die in Form einer Datenbank konkrete Richtlinien, Vorlagen und Werkzeugunterstützung bietet. Diese Variante wird mit *Rational Unified Process (RUP)*[103] bezeichnet.[104]

2.4.5.7 KobrA-Methode

Nach Bunse wurde die KobrA-Methode erstmals 2001 als Vorgehensmodell durch ein Team der Fraunhofer-Gesellschaft vorgestellt. Laut Fraunhofer handelt es sich bei der Bezeichnung *KobrA* um ein Akronym für *Komponentenbasierte Anwendungsentwicklung*. So beschreibe KobrA ein Vorgehensmodell, das in allen Phasen dem Paradigma der komponentenorientierten Softwareentwicklung folge.[105] Bunse beschreibt, dass bei dieser Vorgehensweise das Zielsystem als eine hierarchische Anordnung der einzelnen Komponenten definiert wird. Diese Komponenten werden dabei alle auf dieselbe Art modelliert und implementiert. Laut Bunses zusammenfassender Beschreibung sei die KobrA-Methode „ein systematischer und rekursiver Ansatz zur Entwicklung von qualitativ hochwertigen, komponentenorientierten Systemen“.[106]

Das Vorgehensmodell charakterisiert drei Phasen, die in einem rekursiven Prozess durchlaufen werden. Diese drei Phasen sind die *Kontext-Realisierung*, das *Framework*- beziehungsweise *Application Engineering* und die *Implementierung*. Vorgegebene Konsistenzregeln zwischen den Ergebnissen erlauben deren Überprüfung im Rahmen von Inspektionen. Durch die rekursive Vorgehensweise entsteht bei der Entwicklung des Systems ein so genannter *Komponentenbaum*, der zugleich als zentrales Ergebnis eines mit KobrA entwickelten Systems gilt. Die Eigenschaften jeder Komponente dieses Baumes werden durch UML-Diagramme und weitere Textdokumente beschrieben. Als einen weiteren Vorteil dieser Methode erläutert Bunse die vorgesehenen Möglichkeiten zur Definition von Regelungen zur

[103] Weitere Informationen zu Rational Unified Prozess können der Webseite „http://www-306.ibm.com/software/awdtools/rup/“ entnommen werden.
[104] Vgl. Bunse 2001 S. 75 bis S. 89 Weitere Informationen zu Unified Process können Jacobson 1999 entnommen werden.
[105] Vgl. Fraunhofer 2002.
[106] Vgl. Bunse 2001 S. 89f.

Qualitätssicherung im Entwicklungsprozess, die speziell die vorliegende objektorientierte und komponentenbasierte Softwareentwicklung adressieren.[107]

2.4.5.8 Sonstige Vorgehensmodelle

Dieser Abschnitt beschreibt weitere Vorgehensmodelle, die während der Recherchetätigkeiten herausgearbeitet worden sind. Es handelt sich dabei um Modelle, die zum Beispiel aufgrund ihrer geringen Verbreitung oder ihrer engen Verwandtschaft zu bereits erläuterten Modellen in dieser Arbeit keine weitere Betrachtung finden.

Das *Fusion-Modell* wurde von Derek Coleman und seinen Arbeitskollegen innerhalb der Firma Hewlett-Packard (HP) in der Mitte der 1990er Jahren entwickelt. Es ist auf der Grundlage der Ideen und Konzepte der Vorgehensmodelle OMT (siehe Abschnitt 2.4.5.1), Booch (siehe Abschnitt 2.4.5.2) und Objectory (siehe Abschnitt 2.4.5.3) erarbeitet worden. Durch den hohen Strukturierungsgrad und die Vereinheitlichung der bekannten Konzepte der Objektorientierung wurde das Fusion-Vorgehensmodell schnell bekannt. Aufgrund der fehlenden Weiterentwicklung seitens HP erreichte das Modell allerdings keinen großen Verbreitungsgrad.[108]

Das *objektorientierte Life-Cycle-Modell* und das *objekt- und prototypenorientierte Life-Cycle-Modell* orientieren sich an den klassischen Vorgehensmodellen der Phasen-, Wasserfall- und Schleifenmodelle (siehe Abschnitt 2.4.2). Sie sehen nur geringe Änderungen an diesen bestehenden Vorgehensmodellen vor.[109]

Der *Transformations-Ansatz* wurde nach Hesse Mitte der 1970er Jahre entwickelt. Verschiedene Gruppen beschäftigten sich zu diesem Zeitpunkt mit diesem Ansatz. Das grundlegende Prinzip dieser Vorgehensweise ist es, das zu erstellende Softwaresystem basierend auf den Vorgaben und Spezifikationen durch schrittweise Transformationen zu entwickeln.[110]

Neben den Vorgehensmodellen, die in den vorhergehenden Abschnitten dargestellt sind, existieren weitere Vorgehensweisen und Modellen, die die Arbeitsschritte zur

---

[107] Vgl. Fraunhofer 2001 und Bunse 2001 S. 89 bis S. 91. Weitere Informationen zur KobrA-Methode können bei Bunse 2001 S. 91 bis S. 98 und Fraunhofer 2001 nachgelesen werden.
[108] Vgl. Bunse 2001 S. 46. Weitere Informationen zu dem Fusion-Vorgehensmodell können bei Bunse 2001 auf den Seiten 47 bis 54 und bei Coleman 1994 nachgelesen werden.
[109] Vgl. Pomberger 1996 S. 28 bis S. 32.
[110] Vgl. Hesse 1992 S. 71 f.

Erstellung eines Software-Systems beschreiben und methodisch unterstützen. Diese bieten aber kaum neue Ansätze und Techniken und lassen sich so überwiegend den Modellen und Modellfamilien des Kapitels 2.4 f. zuordnen. Sie werden deshalb auch in dieser Arbeit nicht weiter betrachtet.

3 Konzeption eines Vorgehensmodells für kollaborative Applikationen

Der folgende Abschnitt beschreibt die Erarbeitung eines Vorgehensmodells bezogen auf die Entwicklung von kollaborative Anwendungen. Zunächst werden basierend auf den Eigenschaften kollaborativer Anwendungen Anforderungen an ein Vorgehensmodell definiert, welches die Entwicklung dieser Applikationen unterstützt. Unter dem Gesichtspunkt dieser Anforderungen werden die Modelle diskutiert, die bereits im zweiten Kapitel dargestellt wurden. Aus dieser Auswahl von Vorgehensmodellen wird darauf folgend das neue Modell konzipiert.

3.1 Anforderungen an das Vorgehensmodell

3.1.1 Sicherheit

Kollaborative Applikationen speichern ihre Daten zentral und halten diese Informationen unterschiedlichen Benutzern vor. Diese Benutzer können dabei wahlfrei auf die hinterlegten Daten zugreifen. Damit sensible Daten geschützt werden können, und beispielsweise der Datenbestand einer Firma nur deren Mitarbeitern vorbehalten ist, sind verschiedene Sicherheitsmechanismen notwendig. Diese Sicherungsmaßnahmen werden zusammen mit den Anforderungen an ein Vorgehensmodell, die sich daraus ergeben, in den nachfolgenden Abschnitten erläutert.

3.1.1.1 Sicherung des Zugriffs

Werden Informationen so gespeichert, dass unterschiedliche Personen auf diese Daten zugreifen können, so muss eine Regelung dieser Zugriffe möglich sein. Nicht alle Benutzer benötigen den Zugriff auf alle hinterlegten Daten. Sensible Firmeninformationen, die nur für Mitglieder der Firmenleitung zugreifbar sein sollen, oder Ergebnisse der Forschungsabteilung, die nur einem ausgewählten Mitarbeiterkreis vorbehalten sind, wären hierfür denkbare Beispiele. Die Sichtbarkeit und damit die Verfügbarkeit der Daten muss somit abhängig von der Person sein, die auf dem Datenbestand arbeitet.

Ein differenzierendes Benutzermanagement ist somit für die Applikation unerlässlich. Nach Nastansky hat sich die Benutzerverwaltung mithilfe eines

abstrakten Rollen- und Gruppenkonzepts als zweckmäßig erwiesen. Für die einzelnen Tätigkeitsbereiche werden abstrakte Rollen definiert und mit den nötigen Zugriffsrechten versehen. Den Benutzern, die diese Tätigkeiten gerade ausführen, werden die entsprechenden Rollen zugewiesen. Sie erben somit die zugehörigen Zugriffsrechte der jeweiligen Rolle. Durch die Definition von Gruppen lassen sich unterschiedliche Benutzer und untergeordnete Gruppen zu einer organisatorischen Einheit zusammenfassen und mit den Rechten versehen, die der Aufgabe dieser neuen Gruppe entsprechen. Da sowohl Rollen als auch Gruppen wiederum unter- und übergeordnete Rollen und Gruppen beinhalten können, ist so eine realitätsnahe Abbildung der Organisationsstruktur möglich.[111]

Ein Vorgehensmodell muss also die Erarbeitung eines Konzepts zur Regelung des Datenzugriffs unterstützen. Die Entwickler sollen so angeleitet werden, ein Zugriffskonzept für das zu erstellende Softwaresystem zu erarbeiten, das dem Anwender den Zugriff auf genau die Teilmenge der gespeicherten Informationen erlaubt, die für ihn zugelassen und relevant ist.

3.1.1.2 Datensicherheit

Kollaborative Applikationen verarbeiten und speichern Informationen unterschiedlicher Arten. So werden neben Feldinhalten vorgegebener Formulare auch unstrukturierte oder multimediale Inhalte, so genannter *Rich Text*, hinterlegt. Diese Inhalte müssen bereits in kleinen Teilmengen mit Sicherungsmechanismen zu schützen sein, da nicht alle Informationen eines Datensatzes für alle Benutzer zugänglich sein sollen. Öffnet beispielsweise ein Mitarbeiter des Vertriebs einen Datensatz eines Kunden, so sollen auch nur die Informationen sichtbar sein, die in diesem Umfeld gerade notwendig sind. Darüber hinaus sollten Benutzer auch nur die Informationen verändern können, für die sie auch die entsprechende Berechtigung haben. Auch die Konzeption dieser Sicherheitsmechanismen muss von einem Vorgehensmodell für kollaborative Applikationen berücksichtigt und unterstützt werden.

[111] Vgl. Nastansky 2000 S. 255.

3.1.2 Client Architektur

Da kollaborative Applikationen von Benutzern an verschiedenen Orten zu verschiedenen Zeiten genutzt werden (siehe Abschnitt 2.1), ergibt sich insbesondere für die Architektur des Zielsystems die Notwendigkeit, die Ausgestaltung des Clients zu überdenken. Hierbei muss herausgearbeitet werden, an welchem Ort die unter Umständen sehr umfangreiche Programmlogik ausgeführt werden soll. So genannte *Fat-Client-Lösungen* führen die Programmlogik lokal auf dem Computer des jeweiligen Benutzers aus, während bei *Thin-Client-Lösungen* Programmvorgänge hauptsächlich auf dem verbundenen Server abgearbeitet werden und der Client zum Beispiel in Form eines Browsers vornehmlich zur Darstellung der errechneten Ergebnisse dient. Beide Ansätze haben sowohl Vor- als auch Nachteile, die bei dem Entwurf kollaborativer Applikationen im Einzelfall gegeneinander abgewägt werden sollen. Auch eine Kombination beider Wege sollte bei dieser Überlegung nicht als Lösung ausgeschlossen werden.[112] Deshalb sollte ein Vorgehensmodell die Diskussion dieser verschiedenen Lösungswege unterstützen.

3.1.3 Sessionmanagement und Skalierbarkeit

Ihrer Beschreibung nach werden kollaborative Applikationen von unterschiedlichen Personen zu beliebigen Zeitpunkten genutzt. Dies bedeutet, dass die Anwendungen auch für parallele Zugriffe unterschiedlicher Benutzer ausgelegt sein müssen. Es müssen Techniken vorgesehen werden, die beispielsweise eine interne Trennung der einzelnen Sitzungen (engl. sessions) der Anwender erlauben. Darüber hinaus muss untersucht werden, welche Anzahl von Benutzern das spätere System verwenden wird. Auch eine geplante oder vorhersehbare Erhöhung der Anzahl der Anwender nach der Einführung des Systems muss während der Planung der Architektur beachtet werden. Die Bestimmung des Grades der Skalierbarkeit eines geplanten Softwaresystems muss neben dem Entwurf des Sessionmanagements auch von einem Vorgehensmodell vorgesehen und beschrieben werden.

3.1.4 Benutzeroberfläche

Die Benutzeroberfläche ist als Schnittstelle zwischen der Anwendung und dem Benutzer bei nahezu jedem Programm von entscheidender Bedeutung. Neben den

[112] Weitere Informationen zu Vor- und Nachteilen von Fat- und Thin-Clients können bei Erdmann 2003b nachgelesen werden.

Vorschriften zur Gestaltung von Benutzeroberflächen, existieren einige Anforderungen, die sich direkt aus den Eigenschaften kollaborativer Applikationen ergeben.[113] Das Merkmal kollaborativer Applikationen, von einer beliebigen Anzahl von Benutzern zur gleichen Zeit verwendet zu werden, wirkt sich auch auf die Gestaltung der Programmoberfläche aus. Soll eine kollaborative Applikation zum Beispiel als Kommunikationsplattform eines Projektes genutzt werden, dessen Projektmitglieder örtlich voneinander getrennt in den Firmenniederlassungen unterschiedlicher Länder arbeiten, so kann die Mehrsprachigkeit einer Applikation ein grundlegender Bestandteil der Programmentwicklung sein. Die Möglichkeit der Internationalisierung von Applikationen muss allerdings zu einem frühen Zeitpunkt in der Systemarchitektur vorgesehen werden, damit aufwendige Modifikationen zu einem späteren Zeitpunkt vermieden werden können.

Darüber hinaus werden kollaborative Applikationen als Anwendungen beschrieben, die den Anwendern arbeitsrelevante Informationen in aufbereiteter Form präsentieren. Diese Selektion von Information, die in dem jeweiligen Arbeitsumfeld relevant ist, sowie deren Aufbereitung zu einer optimalen Präsentation, stellen weitere Anforderungen an die Benutzeroberfläche dar. Der Anwender muss die Möglichkeit haben, mithilfe von geeigneten Mechanismen wie zum Beispiel spezialisierten Ansichten oder Suchfunktionen, die für ihn wichtigen Informationen möglichst unkompliziert zu finden.

Insbesondere unter dem Gesichtspunkt des Wissensmanagements als Aufgabe kollaborativer Applikationen erhält die Präsentation der Daten eine besonders hohe Bedeutung. Um die Akkumulation von Wissen effektiv betreiben zu können, müssen neben den eigentlichen Daten auch die Zusammenhänge zwischen den einzelnen Informationen so dargestellt werden, dass sie durch den Anwender unmittelbar zu erfassen sind. Die Realisierung dieser Mechanismen muss innerhalb des Entwicklungsprozesses der Applikationen durch das Vorgehensmodell berücksichtigt werden.

[113] Die detaillierten software-ergonomischen Grundsätze können als Norm DIN EN ISO 9241-10 nachgelesen werden.

3.1.5 Mailsystem

Die Realisierung eines Mailsystems als typischen Bestandteil kollaborativer Applikationen soll hingegen nicht Teil des Vorgehensmodells sein, da Mailsysteme inzwischen zu der grundlegenden IT-Infrastruktur nahezu jeder Organisation gehören. Des Weiteren ist es nicht sinnvoll, bei jeder Entwicklung einer weiteren kollaborativen Anwendung ein zusätzliches Mailsystem zu planen und zu implementieren. Vielmehr stellt die Programmierung und Einführung eines Systems zur Übermittlung und Verwaltung von E-Mails innerhalb einer Organisation einen sehr seltenen Vorgang dar, der somit nicht in dem Vorgehensmodell für die Entwicklung kollaborativer Applikationen Berücksichtigung finden muss.

3.1.6 Datenstruktur

Die Struktur der gespeicherten Daten einer kollaborativen Applikation wird vielfach durch die Eigenschaften und die Anforderungen der Anwendung bestimmt. So müssen zum Beispiel innerhalb dieser Struktur die bereits in Abschnitt 3.1.4 beschriebenen Möglichkeiten der Verknüpfung von Informationen vorgesehen sein, damit deren Darstellung in ihrem jeweiligen Kontext verwirklicht werden kann. Vor allem unter dem Gesichtspunkt des Wissensmanagements, als einer weiteren wichtigen Eigenschaft kollaborativer Applikationen (siehe Abschnitt 2.1), ist die Möglichkeit zur Herstellung von Beziehungen zwischen den einzelnen Wissensbausteinen eine grundlegende Anforderung.

Darüber hinaus muss die spätere Implementierung von Mechanismen zur Filterung und Aufbereitung der hinterlegten Datensätze in der Datenstruktur beachtet werden. So muss beispielsweise bei dem geplanten Einsatz von relationalen Datenbanken[114] als Datenspeicher die Notwendigkeit der effizienten und redundanzfreien Verknüpfung der Datentabellen, die sämtliche Informationen einer relationalen Datenbank beinhalten, berücksichtigt werden. Aus diesem Grund sollte die Erstellung von Datenstrukturen durch ein Vorgehensmodell methodisch unterstützt werden.

---

[114] Relationale Datenbanken speichern Daten in einem *Relationalen Datenmodell*. Hierbei wird die komplexe Datenstruktur auf zweidimensionale Tabellen übertragen. Diese Tabellen werden als *Relationen* bezeichnet. Mithilfe dieser Tabellen werden alle Objekte und Beziehungen der abzubildenden realen Welt und ihre *Attribute* charakterisiert. Selbst komplexe Beziehungen zwischen den Tabellen sind durch die wechselseitige Übernahme von so genannten *Primärschlüsselattributen* möglich. Zur Erarbeitung redundanzarmer Datenstrukturen existiert ein etabliertes Regelwerk (siehe hierzu Wagner 2001 S. 209 bis S. 217). Vgl. Wagner 2001 S. 205.

3.1.7 Zusammenfassung

Abschließend lassen sich aus den speziellen Anforderungen, die die Entwicklung kollaborativer Applikationen an ein so spezialisiertes Vorgehensmodell stellt, drei grundlegende Forderungen zusammenfassen. Zum einen ist die möglichst genaue Analyse der gewünschten Systemeigenschaften von grundlegender Bedeutung. Sowohl die Diskussion von Fat- und Thin-Client-Lösungen (siehe Abschnitt 3.1.2) als auch die Definition der vorgesehenen Eigenschaften einer geplanten Benutzeroberfläche (siehe Abschnitt 3.1.4) bedürfen einer genauen vorherigen Untersuchung der Wünsche und Vorstellungen des Auftraggebers. Auch die Bestimmung der applikationsabhängigen Merkmale einer Datenstruktur (siehe Abschnitt 3.1.6) erfordert eine umfassende Berücksichtigung innerhalb eines Vorgehensmodells in Form einer durch das Modell unterstützten Analyse.

Als zweite wichtige Anforderung lässt sich die Unterstützung aller Tätigkeiten zur Erarbeitung der Systemarchitektur ablesen. In nahezu allen beschriebenen Anforderungen sind Elemente der Architektur des späteren Systems betroffen. Insbesondere die große Bedeutung fein graduierter Sicherheitsmechanismen zur Regelung und Sicherung eines differenzierten Datenzugriffs (siehe Abschnitt 3.1.1) sind besonders hervorzuheben. Auch die Bereiche der Skalierbarkeit und des Managements der Benutzer-Sitzungen (siehe Abschnitt 3.1.3) müssen neben den speziellen Eigenschaften der Benutzeroberfläche, wie zum Beispiel die Mehrsprachigkeit einer Software (siehe Abschnitt 3.1.4), bei der Planung der Systemarchitektur berücksichtigt und unterstützt werden.

Die bereits angeführten Aspekte der Skalierbarkeit sowie die Notwendigkeit von geeigneten Sicherheitsmaßnahmen stellen ebenfalls eine besonders hohe Anforderung an die Qualität des zu programmierenden Codes. So muss der Programmcode möglichst einfach geschrieben sein, damit spätere Erweiterungen oder Anpassungen mit möglichst geringem Aufwand realisierbar sind. Darüber hinaus ist eine stabile und fehlerfreie Codebasis für die beschriebenen Sicherungsmechanismen (siehe Abschnitt 3.1.1) einer kollaborativen Applikation unerlässlich. Somit sollte dieser Qualitätsanspruch durch geeignete Schritte innerhalb eines Vorgehensmodells für die Entwicklung dieser Anwendungen einbezogen werden.

3.2 Diskussion geeigneter Vorgehensmodelle

Die in Abschnitt 3.1 erläuterten speziellen Anforderungen kollaborativer Anwendungen sollen in einem neuen Vorgehensmodell Berücksichtigung finden. Dieses Vorgehensmodell wird in den folgenden Abschnitten aus einer Diskussion bereits bestehender Vorgehensbeschreibungen erarbeitet. Hierbei werden die verschiedenen Vor- und Nachteile der in Abschnitt 2.4 vorgestellten Vorgehensmodelle auch unter der Betrachtung der besonderen Anforderungen kollaborativer Applikationen gegeneinander abgewogen, um schließlich eine möglichst optimale Lösung zu definieren.

3.2.1 Vergleich der Gruppen von Vorgehensmodellen

Dieser Abschnitt grenzt die Vorgehensmodelle der Abschnitte 2.4.2 bis 2.4.4 gegeneinander ab. Hierbei werden die jeweiligen Gruppen von Vorgehensmodellen mithilfe bestimmter Kriterien miteinander verglichen, um die Familie von Vorgehensweisen zu erarbeiten, die als Grundlage für ein Vorgehensmodell für kollaborative Applikationen dienen soll. Die im Abschnitt 2.4.5 explizit erläuterten Modelle der objektorientierten Softwareentwicklung finden in diesem Abschnitt keine weitere Betrachtung, da sie vielfach Eigenschaften besitzen und Vorgehensweisen beschreiben, die eine Zuordnung zu den drei hier betrachteten Familien von Vorgehensmodellen erlauben.

3.2.1.1 Anforderungsermittlung und Anforderungsstabilität

Während die *Anforderungsermittlung* vor allem als Abgrenzungskriterium der Vorgehensmodelle bei der Identifizierung der Anforderungen an ein Softwaresystem dienen soll, beschreibt das Kriterium der *Anforderungsstabilität*, wie flexibel die Vorgehensmodelle hinsichtlich möglicherweise auftretender Änderungen der ermittelten Anforderungen in dem zeitlichen Verlauf des Projekts reagieren können.

Die Modelle der Phasen-, Wasserfall- und Schleifenmodelle sehen eine ganzheitliche Definition der Systemanforderungen innerhalb einer geschlossenen Phase vor. Hierbei werden alle Anforderungen und Schnittstellen in einem Schritt berücksichtig. Durch die Möglichkeit von Rückschritten kann auf die Änderung der Anforderungen eingegangen werden. Dieser Weg führt aber zu Problemen, wenn die Anforderungen ständigen Modifikationen unterworfen sind. In diesem Falle würde die Phase der Anforderungsdefinition nie verlassen werden. Durch die vorgeschriebene

sequenzielle Vorgehensweise würde somit zu keinem Zeitpunkt ein lauffähiges System entstehen, das an den Kunden geliefert werden kann. Entsprechend dieser Eigenschaften sind Modelle dieser Gruppe weniger für Projekte geeignet, deren Anforderungen an das Softwaresystem unklar sind beziehungsweise starken Veränderungen unterliegen. Aufgrund der Verwendung von Prototypen sind dagegen prototypische Vorgehensmodelle in der Lage, eine umfassende und vollständige Anforderungsermittlung vorzunehmen. Der Prototyp ist dabei besonders als Diskussionsbasis und bei der Veranschaulichung der Forderung an ein Softwaresystem hilfreich. Auf diesem Wege können unklare Anforderungen konkretisiert und schrittweise herausgearbeitet werden.

Wie bei den Phasen-, Wasserfall- und Schleifenmodellen existiert auch bei den prototypischen Vorgehensmodellen das Problem, dass sich ständig ändernde Vorgaben für das System zur Folge haben, dass die Phasen der Analyse und des Entwurfs durch die gegebene sequenzielle Vorgehensweise nicht verlassen werden. So wäre die Auslieferung eines Systems nie möglich. Die Verwendung von Inkrementen in der Vorgehensweise der Gruppe der inkrementellen, evolutionären und iterativen Vorgehensmodellen kann wie bei der prototypischen Vorgehensweise erfolgreich eingesetzt werden, wenn die Anforderungen noch nicht eindeutig formuliert vorliegen und in Schritten beziehungsweise in Inkrementen erarbeitet werden müssen. Im Gegensatz zu den beiden vorherigen Gruppen von Modellen kann bei Vorgehensmodellen mit inkrementeller, evolutionärer und iterativer Vorgehensweise ein Softwaresystem ausgeliefert werden, wenn sich die Anforderungen an das System über den zeitlichen Projektverlauf verändern. Als Bedingung für diese Flexibilität ist allerdings zu beachten, dass sich die Anforderungen an das zu erstellende System in Teilmengen aufteilen lassen müssen, die untereinander möglichst wenige Abhängigkeiten aufweisen.

3.2.1.2 Integration des Auftraggebers

Dieser Abschnitt vergleicht die Gruppen von Vorgehensmodellen unter dem Gesichtspunkt der Beteiligung des Kunden beziehungsweise des Auftraggebers. Vor allem für die genaue Definition der Anforderungen und bezüglich möglicher Akzeptanztests des Systems kann sich diese Beteiligung positiv auf das spätere Gesamtergebnis auswirken.

Während sich die Zusammenarbeit zwischen Kunden und Projektmitarbeitern bei den Phasen-, Wasserfall- und Schleifenmodellen auf die einmalige Beschreibung der Anforderungen während der Phase der Analyse beschränkt, integrieren die prototypischen Modelle den Kunden frühzeitig in den Projektverlauf. In enger Kooperation entsteht hier während der Phasen der Analyse und des Entwurfs eine eindeutige und gemeinsame Vorstellung über die Eigenschaften des Zielsystems. Der Grad der *Integration des Auftraggebers* ist bei der Gruppe der inkrementellen, evolutionären und iterativen Vorgehensmodellen ebenfalls als hoch zu bewerten. Allerdings wird hierbei neben der Beteiligung des Kunden an der Phase der Analyse auch nach jedem durchlaufenen Zyklus das Entwicklungsergebnis einer gemeinsamen Überprüfung mit dem Auftraggeber unterzogen.

3.2.1.3 Qualitätsmanagement und Risikobewertung

Während das *Qualitätsmanagement* bei der Durchführung von Softwareprojekten vor allem die Möglichkeiten von Tests, Fortschrittsüberprüfungen sowie der Fehlerbehebung umfasst, beschreibt die *Risikobewertung* in diesem Umfeld die Chance im Projektverlauf zu definierten Zeitpunkten das Risiko einer Fehlentwicklung realistisch abzuschätzen und diese Einschätzungen zu verfeinern.

Sowohl Phasen-, Wasserfall- und Schleifenmodelle als auch prototypische Modelle sehen nach dem Abschluss einer Phase Maßnahmen zur Qualitätssicherung vor. So werden zum Beispiel Rückschritte in vorherige Phasen möglich, wenn die Ergebnisse als unzureichend bewertet werden. Inkrementelle, evolutionäre und iterative Vorgehensmodelle bieten neben den Überprüfungen, die zwischen den Phasen durchgeführt werden, auch die Möglichkeit durch den Einsatz von Integrationstests die Qualität des gesamten Produktes des jeweiligen Zyklus' zu überprüfen und entsprechende Vorgaben für nachfolgende Inkremente zu treffen.

Als einzige Familie von Vorgehensmodellen wird bei der inkrementellen, evolutionären und iterativen Vorgehensweise auch die explizite Untersuchung von Risikoquellen vorgesehen. So wird schon früh während jedes Zyklus' eine umfassende Risikoabschätzung durchgeführt, die den weiteren Entwicklungsverlauf des Systems absichert.

3.2.1.4 Flexibilität und Termintreue

Unter der *Flexibilität* und der *Termintreue* soll in diesem Umfeld die Fähigkeit von Vorgehensmodellen verstanden werden, auf mögliche Verzögerungen und Abweichungen vom vorgesehenen Zeitplan während des Projektverlaufs angemessen reagieren zu können. Darüber hinaus beschreiben die beiden Begriffe, ob durch den Einsatz geeigneter Maßnahmen und Mechanismen des Modells auch unter diesen schwierigen Bedingungen der geplanten Termin der Auslieferung des Zielsystems eingehalten werden kann.

Die Vorgehensmodelle der Phasen-, Wasserfall- und Schleifenmodellen und die Modelle der prototypischen Vorgehensweise stellen sich als vergleichsweise unflexibel heraus. Durch die vorgegebene feste Abfolge der Phasen und Arbeitsschritte wird erst innerhalb der letzten Projektabschnitte ein lauffähiges System implementiert. Wird zu einem fortgeschrittenen Zeitpunkt ein Verzug gegenüber der Projektplanung erkannt, so kann auch durch die Implementierung lediglich einer Teilmenge der erarbeiteten Systemeigenschaften die Auslieferung eines lauffähigen Softwaresystems an den Auftraggeber nicht garantiert werden. Der Einsatz dieser Gruppen von Vorgehensmodellen ist deshalb unter dem Gesichtspunkt eines absolut fixen Liefertermins als kritisch zu beurteilen. Die Verwendung von aufwendigen Prototypen, wie es für Modelle der prototypischen Vorgehensweisen vorgesehen, ist birgt unter diesen Bedingungen zusätzlich das Risiko, dass der Kunde beziehungsweise der Auftraggeber den Prototyp bereits für das zu erstellende System hält. Durch die schrittweise ansteigende Zahl der Anforderungen, die durch Modelle der inkrementellen, evolutionären und iterativen Vorgehensmodelle in Zyklen realisiert werden, steht zu beinahe jedem Zeitpunkt der Entwicklung eine ausführbare Version des Softwaresystems für eine Auslieferung zur Verfügung. Auf diese Weise kann eine Verzögerung im zeitlichen Projektverlauf leicht durch die Reduzierung der noch zu durchlaufenden Inkremente ausgeglichen werden.

3.2.1.5 Produkteinführung

Die *Produkteinführung* befasst sich in diesem Kontext vor allem mit verschiedenen Vorgehensweisen bei der Installation und der Übergabe des fertigen Softwaresystems an den Auftraggeber. Hier sind unterschiedliche Strategien durchführbar, die verschiedene Vor- und Nachteile besitzen.

Grundsätzlich liegt bei den Vorgehensweisen, die durch die Phasen-, Wasserfall- und Schleifenmodelle und durch die prototypischen Vorgehensmodelle beschrieben werden, die Einführung beziehungsweise Umstellung auf das geschaffene System in nur einem Schritt nahe, da erst nach Abschluss aller Phasen und Arbeitsschritte das System verfügbar ist. Inkrementelle, evolutionäre und iterative Vorgehensmodelle erlauben an dieser Stelle eine zusätzliche Option. Bedingt durch die Entwicklung in Inkrementen und die Tatsache, dass am Ende jedes Inkrements eine lauffähige Programmversion steht, kann eine schrittweise Einführung den Anwendern den Umstieg beziehungsweise den Einstieg in das neue Softwaresystem erleichtern. Eine Einführung in kleinen Schritten ist prinzipiell ebenfalls für Anwendungen möglich, die mithilfe der ersten beiden Gruppen von Vorgehensmodellen geschaffen worden sind. Allerdings müsste hierbei die vollständige Applikation wieder in die gewünschten Einführungsversionen aufgeteilt werden. Dieser Vorgang ist inkrementellen, evolutionären und iterativen Vorgehensmodellen hingegen inhärent und wird schon frühzeitig mit einem geringeren Aufwand abgeschlossen.

3.2.1.6 Komplexität und vorausgesetztes Wissen

In dem Zusammenhang mit den hier erläuterten Gruppen von Vorgehensmodellen soll unter dem Begriff der *Komplexität* der Grad der Vielschichtigkeit sowie der Umfang von Arbeits- und Vorgehensanweisungen verstanden werden, die den Einsatz geeigneter Planungs- und Organisationsmaßnahmen erfordern können. Das *vorausgesetzte Wissen* beschreibt, wie erfahren die Projektmitarbeiter bezüglich der einzusetzenden Techniken beziehungsweise der beschriebenen Verfahrensweisen bereits zu Beginn eines Entwicklungsprozesses sein sollten, damit das jeweilige Vorgehensmodell in einem konkreten Projekt erfolgreich Anwendung finden kann.

Begründet durch die geringe Anzahl von Phasen, die bei Phasen-, Wasserfall- und Schleifenmodellen durchlaufen werden, ist auch die Zahl der Ergebnisse, die aus den Abschnitten resultieren, eher gering. Auch die Durchführung von Rückschritten erhöht diese Zahl nicht soweit, dass aufwendige Werkzeuge zur Verwaltung der Ergebnisse notwendig werden. Ganz ähnlich dazu verhalten sich die prototypischen Vorgehendmodelle. Hier steigt zwar die Anzahl unterschiedlicher Versionen von Ergebnisdokumenten durch die Verwendung von Prototypen zur ständigen Konkretisierung der Anforderungen, aber die Gesamtzahl an Konfigurationen und Versionen ist als vergleichsweise gering zu bewerten, so dass auch hier der Einsatz

von speziellen Hilfsmitteln unnötig erscheint. Diesem Vorteil stehen besonders hohe Ansprüche an die Mitarbeiter bezüglich ihrer persönlichen Erfahrung im Umgang mit den geplanten Techniken und Arbeitsweisen gegenüber. Die einmalige Abarbeitung der jeweiligen Projektphasen lässt ein Anwenden der Erfahrungen, die während des Durchlaufens der jeweiligen Phase gesammelt wurden, nicht zu. Lediglich im Falle von Rückschritten können die Erfahrung der ersten Bearbeitung in dem wiederholten Durchlauf der Phase zielgerichtet genutzt werden. Dies bedeutet, dass sämtliche Erkenntnisse, die ein Team über den zeitlichen Verlauf des Projektes sammelt, erst in nachfolgenden Projekten angewendet werden können. Speziell die prototypischen Vorgehensmodelle erfordern von den Mitarbeitern zusätzliche Fertigkeiten im Umgang mit Werkzeugumgebungen, die die Erstellung und die Weiterentwicklung von Prototypen erlauben.

Gegenüber den ersten beiden Gruppen von Modellen sehen Vorgehensmodelle der inkrementellen, evolutionären und iterativen Arbeitsweise durch ihre wiederholte Abarbeitung aller Phasen in Inkrementen die Erstellung einer Vielzahl von Ergebnisdokumenten und Systemkonfigurationen vor. Hierdurch wird ein umfangreiches Versionsmanagement unabdingbar. Dies lässt sich durch geeignete Anwendungen und Werkzeugumgebungen automatisieren und unterstützen. Gleichzeitig fördert das wiederholte Durchlaufen von Arbeitsprozessen die Sammlung und die Erweiterung von Erfahrungen im Umgang mit den eingesetzten Techniken und den durchgeführten Arbeitschritten bei den Mitarbeitern. Bei inkrementellen, evolutionären und iterativen Vorgehensmodellen können die gewonnenen Kenntnisse zudem unmittelbar in das laufende Projekt eingebracht und auf diesem Wege direkt nutzbar gemacht werden, da sie bereits im folgenden Inkrement Berücksichtigung finden können. Darüber hinaus ist somit die Notwendigkeit einer umfangreichen Vorbildung der Projektmitarbeiter bezüglich der Technik und Arbeitsweisen, in dem Maße wie sie beispielsweise bei der Verwendung von Vorgehensmodellen der ersten beiden Gruppen unerlässlich ist, nicht gegeben.

3.2.1.7 Risiko einer Fehlentwicklung

Das *Risiko einer Fehlentwicklung* beschreibt die Wahrscheinlichkeit, dass sich das komplette Softwaresystem, das während des Projekts entwickelt wurde, nach dessen Einführung bei dem Auftraggeber als unbrauchbar erweist. Dies kann verschiedene

Ursachen haben. So können beispielsweise Kommunikationsprobleme im Verlauf der Tätigkeiten zu unterschiedlichen Vorstellungen über das zu erstellende System bei den Entwicklern und dem Auftraggeber führen. Werden solche Fehler nicht frühzeitig erkannt und korrigiert, so weicht auch das spätere System unter Umständen soweit von den Vorstellungen des Auftraggebers oder dem vorgesehenen Einsatzzweck ab, dass auch mögliche Nachbesserungen der Software zu aufwendig oder sogar unmöglich erscheinen. Eine solche Fehlentwicklung ist immer mit einem großen Verlust finanzieller und sonstiger Ressourcen verbunden, da trotz großer Investitionen kein entsprechender Gegenwert geschaffen wurde.

Bei der Verwendung der Phasen-, Wasserfall-, und Schleifenmodelle ist die Gefahr einer Fehlentwicklung vergleichsweise groß. Nachdem die Anforderungen einmalig definiert und festgeschrieben sind, wird das Softwaresystem nach diesen Vorgaben erstellt. Es sind zwar so genannte Meilensteine am Ende einer Phase vorgesehen, diese dienen aber lediglich der Fortschrittskontrolle und zur Überprüfung, ob die festgelegten Anforderungen berücksichtigt werden. Bei dieser Vorgehensweise finden Rücksprachen und spätere Verständigungen mit dem Auftraggeber über die Korrektheit der Anforderung und Systemeigenschaften nach dem Abschluss der Analyse kaum mehr statt. Dies erhöht das Risiko einer Fehlentwicklung.

Prototypische Vorgehensmodelle bieten demgegenüber den Vorteil, dass mithilfe eines Prototyps die Anforderungen wesentlich genauer definiert werden können. Der Kunde beziehungsweise der Auftraggeber hat mit dem vorliegenden Prototyp als Diskussions- und Kommunikationsgrundlage immer eine konkrete Vorstellung von dem aktuellen Zielsystem. Grundsätzlich lassen sich auf diesem Wege genauere Beschreibungen und Anforderungsdefinitionen für das zu entwickelnde Softwaresystem finden. Auch die Diskrepanz zwischen den Vorstellungen der Entwickler und den Auffassungen des Auftraggebers bezüglich vereinbarter Systemeigenschaften lassen sich so vermeiden. Als weiteren Vorteil dieser Vorgehensmodelle lassen sich anhand des Prototyps die Anforderungen an das spätere System leicht priorisieren. So kann sichergestellt werden, dass im Falle eines Zeitverzuges nur auf die Realisierung unbedeutender beziehungsweise entbehrlicher Systemeigenschaften verzichtet wird. Auf diese Weise wird ebenfalls das Risiko minimiert, dass das gesamte erstellte System unbrauchbar ist.

Vorgehensmodelle der inkrementellen, evolutionären und iterativen Vorgehensweise bieten aufgrund ihrer schrittweisen Abarbeitung von Teilmengen der Anforderungen ebenfalls die Möglichkeit, diese einzelnen Teilmengen zu priorisieren und somit das Risiko einer vollständigen Fehlentwicklung zu vermindern. Die ständige Rückkopplung mit dem Auftraggeber und den Anwendern zum Beispiel in Form von Integrationstests am Ende jedes Zyklus' hilft darüber hinaus zu vermeiden, dass die Vorstellungen von Entwicklern und Auftraggebern voneinander abweichen. Der Kunde kann also eine Abweichung von seiner Sicht auf die Systemeigenschaften sofort feststellen und entsprechende Maßnahmen innerhalb des Projektteams initiieren. Daneben senkt die regelmäßige Suche nach möglichen Risikoquellen in kommenden Projektabschnitten zusätzlich die Gefahr, dass die erstellte Software nach Abschluss des Projekts für den Kunden nicht in seinem Sinne zu verwenden ist.

3.2.1.8 Zusammenfassung

Tabelle 1 fasst die Ergebnisse der vorangegangenen Abschnitte übersichtlich zusammen. Hierbei wird deutlich, dass die Vorgehensweise der Phasen-, Wasserfall und Schleifenmodelle am wenigsten Vorteile bietet, während die Modelle der prototypischen Vorgehensweise zusätzliche positive Beurteilungen im Bereich der Anforderungsermittlung, Kundenbeteiligung sowie dem vergleichsweise etwas geringerem Risiko einer Fehlentwicklung erreicht haben. Die inkrementellen, evolutionären und iterativen Vorgehensmodelle bieten in fast allen Vergleichskriterien den besten Lösungsweg an. Lediglich in dem Punkt der Komplexität bleiben diese Modelle in ihrer Bewertung hinter den ersten beiden Gruppen zurück. Zusammenfassend handelt es sich bei den grundlegenden Arbeitsweisen, die durch die inkrementellen, evolutionären und iterativen Vorgehensmodelle beschrieben werden, um das erfolgreichste Vorgehen dieses Vergleichs.

Kriterium	Phasen-, Wasserfall- u. Schleifenmodelle	Prototypische Vorgehensmodelle	Inkrementelle, evolutionäre und iterative Modelle
Anforderungsermittlung	-	+	+
Anforderungsstabilität	-	-	+
Integration des Auftraggebers	-	+	+
Qualitätsmanagement	+	+	+
Risikobewertung	Keine Angabe	Keine Angabe	+
Flexibilität	-	-	+
Termintreue	-	-	+
Produkteinführung	+	+	++
Komplexität	+	+	-
Vorausgesetzte Erfahrungen	-	-	+
Risiko einer Fehlentwicklung	-	+	++

Tabelle 1 - Gruppen von Vorgehensmodellen im Überblick[115]

3.2.2 Vergleich objektorientierter Vorgehensmodelle

Dieser Abschnitt vergleicht konkrete Vorgehensmodelle, die eine objektorientierte Softwareentwicklung unterstützen. Diese Modelle wurden bereits in dem Abschnitt 2.4.5 ausführlich beschrieben. Soweit diese Modelle Elemente oder Vorgehensweisen der Gruppen von Vorgehensmodellen der Abschnitte 2.4.2 bis 2.4.4 enthalten, wird auf die entsprechenden Kapitel Bezug genommen. Die speziellen Eigenschaften, die die Modelle auszeichnen, werden in den folgenden Kapiteln diskutiert und voneinander abgegrenzt. Das abschließende Kapitel gibt in der Form einer Zusammenfassung einen Überblick über die erläuterten Eigenheiten der unterschiedlichen Vorgehensmodelle. Das Objectory-Vorgehensmodell wird im weiteren Verlauf nicht explizit betrachtet, da es einerseits in seiner Struktur der Gruppe der inkrementellen, evolutionären und iterativen Vorgehensmodellen entspricht und deren Vor- und Nachteile bereits im Abschnitt 3.2.1 diskutiert worden sind. Andererseits dienen Elemente des Objectory-Vorgehensmodells als Grundlage für die Vorgehensweise Unified Process, die in diesem Abschnitt mit anderen Vorgehensmodellen verglichen wird. Somit finden die Vor- und Nachteile des Objectory-Modells indirekte Betrachtung in diesem Vergleich.

[115] Bewertung: ++ sehr gut, + gut, - schlecht, -- sehr schlecht

3.2.2.1 Komplexität und vorausgesetzte Erfahrung

Ähnlich zu dem Abschnitt 3.2.1.6 wird in diesem Kapitel ebenfalls die Vielschichtigkeit und Tiefe der einzelnen Vorgehensmodelle sowie deren Umfang an Arbeitspaketen und Vorgehensbeschreibungen verglichen und diskutiert. Des Weiteren wird beschrieben, wie umfangreich die Erfahrungen der Mitarbeiter eines Projektes mit den einzusetzenden Technologien und Arbeitsweisen sein sollten, damit die jeweilige Vorgehensweise erfolgreich eingesetzt werden kann.

Die Vorgehensmodelle Object Modeling Technique, KobrA und Extreme Programming verfügen über eine vergleichsweise geringe Anzahl von Phasen und Arbeitsschritten. KobrA verfügt zwar über einen rekursiven und damit sehr stark verzweigten Aufbau, der sich allerdings dadurch vereinfachen lässt, dass die durchzuführenden Arbeitsschritte alle identisch sind. Vorgehensmodelle hoher Komplexität sind somit angesichts ihrer hohen Anzahl an Arbeitspaketen die Modelle Catalysis, Unified Process sowie die Booch-Methode. Während Catalysis vor allem aufgrund der Berücksichtigung der drei verschiedenen Modelldimensionen und der unterschiedlichen Prozessmuster dieser Gruppe angehört, zeichnet Unified Process vor allem die Vorgabe lediglich eines Prozessrahmens aus. Die Anwender dieses Vorgehensmodells sind somit gezwungen, sich zunächst ihr spezielles Vorgehensmodell basierend auf dem gegebenen Prozessrahmen zu erarbeiten. Die Vielschichtigkeit der Booch-Methode äußert sich insbesondere in der Unterscheidung eines Mikro- und eines Makroprozesses innerhalb eines Projektes sowie in dem dort beschriebenen Einsatz einer großen Anzahl unterschiedlicher Diagrammformen.

Diese hohe Komplexität der Booch-Methode, die sogar den Einsatz von Werkzeugen zum Versions- und Konfigurationsmanagement begründet, erfordert von den Projektmitarbeiten umfangreiche Erfahrungen mit dieser Vorgehensweise. Fehlende Konsistenzregeln zwischen den eingesetzten Diagrammformen, die die Beziehungen zwischen den einzelnen Diagrammen eindeutig charakterisieren, setzen umfangreiches Vorwissen seitens der Mitarbeiter voraus. Eine ganz ähnliche Situation liegt bei den Modellen Extreme Programming, Catalysis und KobrA vor. Hier erfordern die Modelle umfangreiche Erfahrungen der Projektmitarbeiter mit den jeweils vorgeschriebenen Arbeitsweisen und Tätigkeiten, die im Abschnitt 2.4.5 genauer erläutert wurden. Insbesondere bei den Vorgehensmodellen Unified Process

und OMT muss der Tatsache, dass konkrete Arbeitsanweisungen für bestimmte Projektphasen teilweise oder vollkommen fehlen, durch einen höheren Erfahrungsstand der Mitarbeiter begegnet werden.

3.2.2.2 Schwerpunkt der Unterstützung

Alle Vorgehensmodelle, die im Abschnitt 2.4.5 beschrieben werden, unterstützen die einzelnen Phasen der Entwicklung eines Softwaresystems unterschiedlich intensiv. Die besonderen Schwerpunkte dieser Unterstützung durch die Modelle werden in diesem Abschnitt dargestellt und verglichen.

Da sämtliche Vorgehensmodelle die Modellierung mithilfe der Modellierungssprache UML vorsehen, wird die Phase der Analyse von allen hier diskutierten Vorgehensmodellen gut unterstützt. Besonders intensiv wird darauf in den Modellen OMT und Unified Process eingegangen. Unified Process besitzt neben dieser Eigenschaft zusätzlich einen zweiten Schwerpunkt, der auf der Definition der Systemarchitektur liegt. So ist die Erarbeitung von Objekt- und Klassenhierarchien zentraler Bestandteil des Vorgehensmodells Unified Process. Diesen Schwerpunkt hat das Modell von der Booch-Methode übernommen, deren Elemente teilweise auch innerhalb von Unified Process auftauchen. Die Vorgehensweise des KobrA-Modells ist aufgrund ihrer rekursiven Arbeitsweise vor allem auf die Bestimmung der Systemarchitektur des späteren Softwareprodukts beziehungsweise auf die Durchführung des Systementwurfes besonders fokussiert. Extreme Programming folgt hingegen einem Ansatz, der den ausführbaren und testbaren Programmcode außerordentlich hervorhebt. Das umfangreiche Teamkonzept und die große Eigenverantwortlichkeit der Entwickler bei der Implementierung sind weitere Schwerpunkte des Vorgehensmodells Extreme Programming.

3.2.2.3 Umfang der Unterstützung

Während der Abschnitt 3.2.2.2 den besonderen Fokus der einzelnen Modelle gegeneinander abgrenzt, wird in diesem Kapitel der Umfang der Unterstützung betrachtet. Insbesondere werden dabei die unterschiedlichen methodischen Hilfestellungen und Arbeitsanweisungen der vorliegenden Vorgehensmodelle bezogen auf die einzelnen Phasen verglichen. Dabei werden auch die Projektphasen betrachtet, die weniger oder kaum unterstützt werden. Auf diese Weise wird

herausgearbeitet, welches der hier erläuterten Modelle die umfassendste Unterstützung bietet.

Das Vorgehensmodell OMT bietet, wie die Modelle Catalysis, Unified Process, KobrA und die Booch Methode, eine umfangreiche Unterstützung der Analyse- und Entwurfsphase des Projektes mithilfe so genannter Use-Cases. Es handelt sich hierbei um typische Interaktionssequenzen zwischen dem Softwaresystem und dem Anwender. Darüber hinaus setzen diese Modelle eine Vielzahl von weiteren UML-Diagrammen ein, um die Anforderungen an das System zu charakterisieren beziehungsweise die Systemarchitektur zu beschreiben. Während das Modell OMT in der Phase der Analyse noch vielfältige Modellierungs- und Analysetechniken zur Bestimmung des Objektmodells, des dynamischen Modells und des funktionalen Modells unterstützt und durch einfach Konsistenzregeln absichert, sieht die Entwurfsphase des OMT-Modells lediglich eine Zerlegung des geplanten Systems sowie die Verbesserung des Objektmodells vor. Die hier gegebenen Arbeitsanweisungen reichen für eine umfassende Bearbeitung dieser Phase des Softwareentwurfs nicht aus.

Ähnlich dazu bietet die Booch-Methode in der Phase der Anforderungsanalyse und des Entwurfs kaum ausführliche Unterstützung in Form konkreter Arbeitsbeschreibungen. Insbesondere für die Phase des Entwurfs sind zwar zahlreiche Diagrammformen vorgesehen, um die Objekt- und Klassenhierarchie möglichst präzise zu erfassen. Allerdings werden die Zusammenhänge zwischen diesen Diagrammen durch die Booch-Methode nur unzureichend erklärt. In diesen beiden Fällen wäre ein eindeutiges Regelwerk für die Gestaltung von beispielsweise Use-Cases, wie es das Vorgehensmodell KobrA für die anfänglichen Phasen eines Projektes vorsieht, sehr hilfreich. Auch die dort gegebene ausführliche Beschreibung der einzelnen Diagrammformen könnte deren Beziehungen untereinander besser verdeutlichen.

Auch das Modell Catalysis sieht den Einsatz verschiedener Abhängigkeiten und Regeln in der Phase des Systementwurfs vor, um die Teilergebnisse zu überprüfen, die bereits während dieser Phase erarbeitet wurden. Catalysis empfiehlt neben dem bereits beschriebenen Einsatz von UML-Diagrammen die Verwendung eines Domänenmodells. Dieses Modell dient zur Identifikation und Beschreibung von

typischen Einsatzszenarien für das spätere Softwaresystem. Außerdem ist die Ausarbeitung von Testfällen hier bereits für die Analysephase vorgesehen.

Neben der beschriebenen Verwendung von Use-Cases und UML-Diagrammen zur Anforderungsanalyse und zum Systementwurf, setzen vor allem die Vorgehensmodelle KobrA und Unified Process zusätzliche Strategien ein, die die Systemarchitektur in den Fokus der Entwickler rückt. Während KobrA vor allem den komponentenorientierten Aufbau des Systems sehr detailliert auflöst, folgt Unified Process dem architekturzentrierten Ansatz der Booch-Methode, um die Objekt- und Klassenhierarchie sowie deren Schnittstellen zu identifizieren.

Extreme Programming hingegen führt die Tätigkeiten der Analyse und des Entwurfs zusammen. Mithilfe von Use-Cases werden so zuvor kreierte User-Stories modelliert. Diese Phase des Vorgehensmodells wird vergleichsweise wenig unterstützt. Allerdings ist dazu anzumerken, dass Tätigkeiten dieser Phase teilweise in die zweite Phase der Realisierung übernommen werden.

Während die Phase der Realisierung und Implementierung des Softwaresystems bei der Booch-Methode kaum berücksichtigt oder methodisch unterstützt wird, stehen Entwicklern, die das Vorgehensmodell OMT einsetzen, einige grundlegende Regelungen zur Verfügung. Diese Regelungen verfügen allerdings über keinerlei konkrete Arbeitsanweisungen oder Durchführungsbeschreibungen. Hier beschreibt zum Beispiel das Modell Catalysis die Einbeziehung des Auftraggebers in die Durchführung von Tests und in die Implementierung. Diese umfangreiche Unterstützung lässt sich auch bei dem Vorgehensmodell Unified Process wieder finden. Neben der explizit unterstützten Erstellung und Anwendung von Testfällen, liefert das Modell umfangreiche Hilfestellungen und Anleitungen während der Realisierung des Zielsystems.

Die Vorgehensweise des Modells KobrA ist hier wiederum stark auf den Einsatz von Softwarekomponenten ausgerichtet. Es wird der Einsatz von so genannten Übersetzungspattern vorgesehen, um aus dem Realisierungsmodell mithilfe von Klassendiagrammen, Pseudo-Code und Komponentenmodell den Programmcode schrittweise zu generieren. Der Begriff Pattern beschreibt in diesem Zusammenhang wieder verwendbare und ausführlich getestete Vorgehensmuster.

Extreme Programming nimmt auch bezüglich der Realisierungsphase eines Softwaresystems eine besondere Rolle ein. Es beschreibt eine Vielzahl von sehr

konkreten Richtlinien und Methoden zur Implementierung eines Systems. Bei dem größten Teil dieser Anweisungen handelt es sich um Regelungen zum Qualitätsmanagement, was somit vor allem auf die Verbesserung der Qualität des produzierten Programmcodes abzielt. Darüber hinaus existieren genaue Vorschriften über die Integration des Auftraggebers, die gemeinsam mit den regelmäßigen Integrationstests für eine besonders hohe Planungssicherheit sorgen.

3.2.2.4 Qualitätsmanagement

Dieser Abschnitt erläutert die Maßnahmen der einzelnen Vorgehensmodelle zur Sicherung der Produktqualität des erstellten Systems. Die einzelnen Regelungen und Arbeitsvorschriften werden hierbei in qualitativer und quantitativer Hinsicht gegeneinander abgegrenzt.

Gegenüber dem Vorgehensmodell OMT, das keine diesbezüglichen Anweisungen explizit vorsieht, bietet die Booch-Methode zumindest Ansätze, Mechanismen zur Qualitätssicherung zu etablieren. Insbesondere fehlen an dieser Stelle konkrete Aufgabenbeschreibungen und vorgegebene Verhaltensmuster.

Bei dem Vorgehensmodell Unified Process beziehen sich diese Regelungen vor allem auf die eindeutige Definition von Eingangs- und Ergebnisdokumenten zu allen Abschnitten eines Projekts. Auf diesem Weg ist eine ständige Fortschrittskontrolle und Überwachung der Qualität der erzielten Ergebnisse möglich. Des Weiteren sorgen die regelmäßig vorgesehenen Einführungen beziehungsweise Bereitstellungen früher Produktversionen für ständige Rückkopplung mit dem Auftraggeber, so dass mögliche Abweichungen des Softwareprodukts von den Vorstellungen und Anforderungen des Kunden früh korrigiert werden können.

Die Regelungen der Vorgehensmodelle Catalysis und KobrA bieten vor allem während der Tätigkeit des Programmierens noch weitere Möglichkeiten des Qualitätsmanagements an. So schreibt Catalysis neben regelmäßigen Tests der Zwischenprodukte der einzelnen Inkremente auch klare Konsistenzregeln und Abhängigkeiten zwischen den erzielten Ergebnissen vor. Ähnliche Regelungen bezogen auf die Beziehungen zwischen den erzielten Teilergebnissen sieht auch das Vorgehensmodell KobrA vor. Außerdem verfügt KobrA über vielfältige Vorschriften und Anweisungen, die sich auf die Implementierung und Fehlersuche beziehen.

Besonders ausgeprägt sind die Maßnahmen zur Qualitätssicherung des Modells Extreme Programming. Umfangreiche und ständige Tests und die stetige Optimierung der Software tragen neben der Einteilung des Entwicklungsprozesses in möglichst kurze Zyklen zur fortwährenden Verbesserung des Programmcodes bei. Auch dieses Vorgehensmodell sieht eine intensive Beteiligung des Auftraggebers vor. So kann auch hier eine Abweichung von den Vorgaben des Kunden schnell und unmittelbar berichtigt werden.

3.2.2.5 Gruppe der Vorgehensmodelle

Die objektorientierten Vorgehensmodelle des Abschnitts 2.4.5 beschreiben Vorgehensweisen, die den Arbeitsweisen der Modellgruppen der Kapitel 2.4.2 bis 2.4.4 vollständig oder teilweise entsprechen. Dieses Kapitel erläutert die Zugehörigkeit der Vorgehensmodelle zu den einzelnen Modellgruppen.

Durch den Einsatz sequenzieller Phasen gehört das Modell OMT der Gruppe der Phasen-, Wasserfall- und Schleifenmodelle an. Auch der Makroprozess, den die Booch-Methode beschreibt, gehört dieser Gruppe an. Hingegen lässt sich der Mikroprozess aufgrund der beschriebenen Iterationen eindeutig der Gruppe der inkrementelle, evolutionären und iterativen Modellen zuweisen. Das Vorgehensmodell Unified Process vereint ebenfalls diese beiden Familien von Modellen. Während die Unterteilung des Projekts in separate Phasen der Vorgehensweise der Gruppe der Phasen-, Wasserfall- und Schleifenmodelle folgt, entsprechen die Regelungen der Kernarbeitsabläufe innerhalb dieser Phasen vor allem denen der inkrementellen, evolutionären und iterativen Vorgehensmodelle. Die Modelle Catalysis, KobrA und Extreme Programming können aufgrund ihrer Vorgehensweise ebenso dieser Gruppe von Vorgehensmodellen zugeordnet werden. Dabei ist zu beachten, dass KobrA außerdem eine rekursive Komponente ausweist, während Extreme Programming zugleich einen prototypischen Ansatz in sich vereint.

3.2.2.6 Besondere Eigenschaften

Unter den besonderen Eigenschaften werden die Charakteristika der Modelle subsumiert, die den vorherigen Aspekten nicht zugeordnet werden konnten. Diese besonderen Eigenheiten werden im nachfolgenden Abschnitt beschrieben.

Eine Besonderheit des Modells OMT ist die umstrittene Trennung von Daten- und Funktionsebene. Diese funktionale Zerlegung des Softwaresystems entspricht nicht dem Paradigma der objektorientierten Softwareentwicklung. Bezüglich des Modells Catalysis ist zu beachten, dass für den geplanten Einsatz auch ein entsprechendes Prozessmuster vorhanden sein muss. Unified Prozess zeichnet sich als generischer Ansatz aus. Soll dieses Modell Anwendung finden, so muss zunächst der Prozessrahmen dem vorliegenden Projekt angepasst werden. Besondere Eigenheiten von Extreme Programming liegen einerseits in dem maximalen Umfang der unterstützten Projekte. Es wird hierbei empfohlen, eine Anzahl von 12 bis 15 Projektmitarbeitern nicht zu überschreiten, da ansonsten verschiedene Prinzipien des Modells unwirksam werden. Andererseits ist Extreme Programming nur für Projekte geeignet, die noch nicht initiiert wurden. Ein Wechsel zum Modell Extreme Programming in einem bereits laufenden Projekt ist somit nicht vorgesehen.

3.2.2.7 Zusammenfassung

Zusammenfassend ist anzumerken, dass vor allem das Modell Unified Process, mit den Schwerpunkten auf den Projektphasen der Analyse und des Entwurfs sowie deren umfangreicher methodischer Unterstützung eine besonders breite Unterstützung der Projektmitarbeiter liefert und in den weiteren Bereichen ebenfalls gute Beurteilungen erzielen kann. Lediglich die Aspekte der Komplexität und der vorausgesetzten Erfahrung bei den Mitarbeitern sind negativ zu bewerten. Als zweites Vorgehensmodell fällt das Modell Extreme Programming auf, das vor allem die Komponenten der Implementierung und das Qualitätsmanagement besonders in den Vordergrund stellt und vielfach unterstützt. Komplementär zu dem Modell Unified Process werden allerdings die Phasen der Analyse und des Entwurfs vergleichsweise gering berücksichtigt. Die Modelle KobrA und Catalysis stellen in diesem Vergleich das Mittelfeld dar. Catalysis unterstützt fast alle hier verglichenen Kriterien gut, allerdings ist eine Fokussierung beziehungsweise eine besonders gute Umsetzung von bestimmten Projektphasen oder Projektbereichen nicht erkennbar. Dagegen ist bei dem Modell KobrA die Phase des Entwurfs und der Implementierung besonders herausgearbeitet. Allerdings konzentriert sich das Modell dabei stark auf die zu verwendenden Softwarekomponenten. Das Modell OMT und die Booch-Methode können vergleichsweise wenige Vorteile auf sich

vereinen. Dabei fallen insbesondere die fehlende Unterstützung von Projektphasen und die teilweise fehlenden Mechanismen zur Qualitätssicherung negativ auf.

Kriterium	OMT	Booch	UP	Catalysis	XP	KobrA
Komplexität	+	-	-	-	+	+
Erfahrung der Mitarbeiter	-	-	-	-	-	-
Schwerpunkt - Analyse	++	+	++	+	+	+
Schwerpunkt - Entwurf		++	++			++
Schwerpunkt - Implementierung					++	
Unterstützung - Analyse	++	-	++	+	-	-
Unterstützung - Entwurf	-	-	++	+	-	+
Unterstützung - Implementierung	-	--	+	+	++	++
Qualitätsmanagement	--	-	+	+	++	+

Tabelle 2 - Vergleich von Vorgehensmodellen der OOP[116]

3.3 Vorgehensmodell zur Entwicklung kollaborativer Applikationen

Nachdem in den vorangehenden Kapiteln die verschiedenen Gruppen von Vorgehensmodellen sowie eine Reihe von Vorgehensmodellen der objektorientierten Softwareentwicklung mit ihren Vor- und Nachteilen ausführlich erläutert und abgewogen wurden, wird nun aufgrund dieser Ergebnisse ein Vorgehensmodell beschrieben, das vor allem die Entwicklung kollaborativer Applikationen unterstützt.

In einem ersten Schritt wird zunächst das Vorgehensmodell in seinen Bestandteilen und Bausteinen sowie deren Ursprung erläutert. Danach werden zusätzliche Modifikationen des Modells dargestellt, die sich aus dem spezifischen Einsatzgebiet des Vorgehensmodells ergeben, aber im ersten Schritt noch nicht berücksichtigt werden konnten.

3.3.1 Beschreibung des Vorgehensmodells KollApps

Dieses Kapitel beschreibt ausführlich das Vorgehensmodell für die Entwicklung kollaborativer Applikationen, das auf der Grundlage der Vorüberlegungen und Vergleiche vorangegangener Abschnitte erarbeitet wurde. Das Vorgehensmodell wurde insbesondere aus den Bestandteilen der existierenden Modelle geschaffen, die

[116] Aus Gründen der fehlenden Vergleichbarkeit wurde an dieser Stelle auf die Darstellung der Merkmale der besonderen Eigenschaften (siehe Abschnitt 3.2.2.6) und der Zugehörigkeit der Vorgehensmodelle zu einer Modellgruppe (siehe Abschnitt 3.2.2.5) verzichtet.
Bewertung: ++ sehr gut, + gut, - schlecht, -- sehr schlecht

besonders viele Vorteile bieten und den Entwicklern das größte Maß an Unterstützung zukommen lassen. Die Nachteile dieser Bausteine sollten dabei so gering wie möglich ausfallen beziehungsweise von den gleichzeitig gebotenen Vorteilen stark überwogen werden.

Die Bezeichnung des Vorgehensmodells ergibt sich aus seinem speziellen Einsatzbereich in der Entwicklung **koll**aborativer **App**likationen. Zugleich stellt die phonetische Verwandtschaft mit dem Begriff *Kollaps* in seiner ursprünglichen Bedeutung eines wirtschaftlichen Zusammenbruchs[117] eine ironische Anspielung auf die hohe Zahl von Projekten der Softwareentwicklung, die nicht rechtzeitig oder nie erfolgreich abgeschlossen werden, dar.[118]

3.3.2 Aufbau von KollApps

Die grundlegende Struktur von KollApps wurde von dem Vorgehensmodell Unified Process übernommen, das im Abschnitt 2.4.5.6 ausführlich beschrieben wird. Unified Process bietet insbesondere in den Bereichen der Analyse und des Entwurfs gegenüber den anderen Modellen des Vergleichs viele Vorteile und eine besonders umfangreiche Unterstützung der Projektmitarbeiter während dieser Zeitabschnitte (siehe Abschnitt 3.2.2). Um diese Vorteile zu nutzen, übernimmt KollApps diese beiden Phasen fast vollständig. KollApps weist somit ebenfalls die Unterteilung des Projektverlaufes in vier Phasen auf. Deren inhaltliche Beschreibung bleibt weitgehend erhalten. Allerdings wird vor allem die Phase der Implementierung beziehungsweise der Konstruktion vollkommen umgestaltet. An ihre Stelle tritt die *Iterative Phase* des Vorgehensmodells Extreme Programming (siehe Abschnitt 2.4.5.4). Während der Diskussion der unterschiedlichen Vorgehensmodelle konnte die iterative Phase durch ihre vorteilhafte Vorgehensweise überzeugen. Diese entspricht dem klassischen Vorgehen der Gruppe der inkrementellen, evolutionären und iterativen Vorgehensmodelle, die sich im Vergleich zu den anderen Gruppen als besonders positiv herausgestellt hat (siehe Abschnitt 3.2.1). Auf diese Weise bleibt auch nach dem Austausch der Phase der Implementierung der Ablauf der einzelnen Phasen in ihrer inkrementellen und iterativen Vorgehensweise erhalten. Darüber hinaus hat die iterative Phase des Vorgehensmodells Extreme Programming,

[117] Vgl. Duden 1994 S. 734.
[118] Nach internationalen Untersuchungen können zwischen 30 und 40 Prozent aller initiierten IT-Projekte nicht erfolgreich abgeschlossen werden. Vgl. Heise 2004.

verglichen mit den Vorgehensweisen anderer Modelle, hervorragende Ergebnisse in den Bereichen der Unterstützung der Mitarbeiter und der Qualitätssicherung bei der Erstellung des Programmcodes erzielt (siehe Abschnitt 3.2.2.7). Damit wird zentralen Anforderungen an die Entwicklung kollaborativer Applikationen, wie beispielsweise der ständigen Optimierung und Überprüfung des Programmcodes, entsprochen. Diese speziellen Anforderungen werden in Abschnitt 3.1.7 zusammengefasst. Auch die Phase der Übergabe des Produkts wird leicht verändert von dem Vorgehensmodell Unified Process für das Modell KollApps übernommen. Die Struktur und die detaillierten Arbeitsanweisungen der Phasen des Vorgehensmodells KollApps werden neben zusätzlichen Modifikationen und Erweiterungen in den folgenden Kapiteln erläutert.

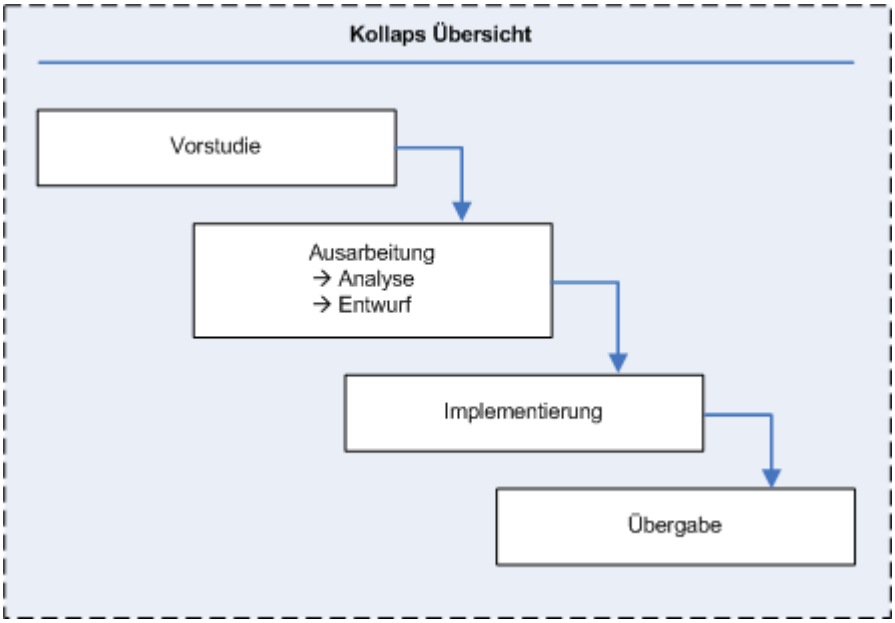


Abbildung 3 - KollApps Prozessphasen

3.3.3 Vorstudie

Die Phase der Vorstudie entspricht der ersten Phase des Vorgehensmodells Unified Process. In iterativer Vorgehensweise wird das Arbeitspaket der Anforderungsermittlung mehrfach durchlaufen. Im Vordergrund stehen während dieser Phase die Frage der Durchführbarkeit des Projektes sowie die Skizzierung des Funktionsumfangs des geplanten Produkts. Grundlegende Überlegungen bezüglich der Systemarchitektur, die Identifizierung möglicher Risiken bei deren Realisierung und die Präsentation aller Ergebnisse vor dem Kunden beziehungsweise dem Auftraggeber sind ebenfalls Tätigkeiten dieser ersten Phase.

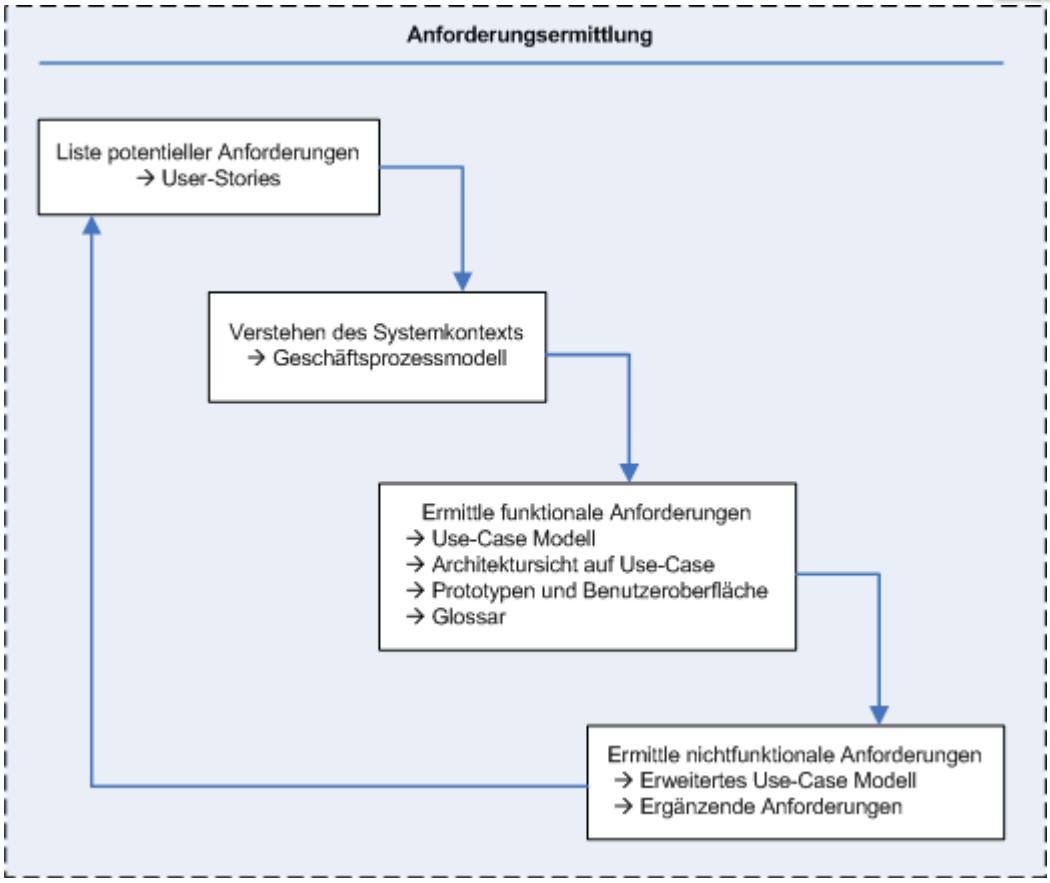


Abbildung 4 - KollApps: Phase der Vorstudie - Anforderungsermittlung[119]

Entgegen dessen ursprünglicher Definition innerhalb von Unified Process, werden im ersten Schritt der Anforderungsanalyse des Vorgehensmodells KollApps so genannte User-Stories gesammelt (siehe Abbildung 4). Diese User-Stories werden in dem Vorgehensmodell Extreme Programming eingesetzt (siehe Abschnitt 2.4.5.4) und beschreiben in wenigen Sätzen die Erwartungen der Benutzer beziehungsweise des Kunden an das zu erstellende Softwaresystem. Entsprechend ihrer Aufgabe innerhalb des Vorgehensmodells Extreme Programming dienen sie auch dem Modell KollApps zur Dokumentation der Anforderungen und als Grundlage späterer Akzeptanztests.

Der zweite Schritt dieses Arbeitspakets gilt dem Verständnis der Systemumgebung, in der die Software verwendet werden soll. Unified Process sieht hierbei für Informationssysteme die Modellierung von Geschäftprozessen vor. Dieser Schritt dient der Identifikation der Geschäftsobjekte, der Bestimmung der Systembenutzer und ihrer Aufgaben.

[119] In Anlehnung an Bunse 2001 S. 79 und Jacobson 1999 S. 342 bis S. 345.

Im weiteren Vorgehen werden die funktionalen Anforderungen der bereits ermittelten User-Stories mithilfe von Use-Cases erarbeitet und dokumentiert. Nachdem die Benutzerrollen und Use-Cases erfasst wurden, werden die Use-Cases entsprechend ihrer Priorität geordnet. Auf diese Weise entsteht eine erste Vorstellung über die Systemarchitektur, die so bereits zu einem frühen Zeitpunkt besonders wichtige oder kritische Anforderungen berücksichtigt. Der Einsatz von Zustands-, Aktivitäten- und Sequenzdiagrammen erleichtert die weitere Ausarbeitung der Use-Cases und die genaue Bestimmung der Informationen, die zwischen den Benutzern und dem System ausgetauscht werden. Der Entwurf eines Prototyps schafft mit den bereits im Abschnitt 3.2.1.8 erläuterten Vorteilen dieser Vorgehensweise zusätzliche Klarheit über die Schnittstellen zwischen dem System und den späteren Anwendern. Mit dieser Tätigkeit kann schon frühzeitig ein erster Eindruck bezüglich der späteren Benutzeroberfläche gewonnen werden. So kann den speziellen Anforderungen kollaborativer Applikationen an die Gestaltung dieser Schnittstelle beispielsweise in Form von aufbereiteten Übersichten Rechnung getragen werden (siehe hierzu Abschnitt 3.1.4).

Abschließend wird auf der Grundlage der vorangegangenen Arbeitsschritte das Use-Case Modell erstellt. Dieses beschreibt das Zusammenwirken der einzelnen Use-Cases. Darüber hinaus umfasst dieser Schritt die Erstellung eines Glossars, das einen Überblick über das Fachvokabular schafft, das in dem Projekt verwendete wird, und auf diese Weise eindeutige Begriffsdefinitionen bereitstellt. Das Glossar eines Projekts wird insbesondere von dem Geschäftsprozessmodell abgeleitet.

Danach werden die nichtfunktionalen Anforderungen ermittelt, die sich aus den gesammelten User-Stories ergeben. Hierbei kann es sich beispielsweise um besondere Anforderungen an die Performanz des Zielsystems handeln. Können diese speziellen Anforderungen einem Use-Case zugeordnet werden, so werden sie der Beschreibung des Use-Cases hinzugefügt. Ist dies nicht der Fall, werden diese Anforderungen in Form eines gesonderten Dokumentes ergänzend darstellt.

3.3.4 Ausarbeitung

Die Phase der Ausarbeitung umfasst zwei Arbeitsabläufe, die mehrfach hintereinander ausgeführt werden. Hierbei handelt es sich um die Arbeitspakete der Analyse (siehe Abbildung 5) und des Entwurfs (siehe Abbildung 6), die beide dem Vorgehensmodell Unified Process entnommen wurden. Für das Modell KollApps

wurden Modifikationen vorgenommen, die sich aus den speziellen Anforderungen für die Entwicklung kollaborativer Applikationen ergeben. Ziel dieser Phase ist, neben der Entwicklung einer soliden Systemarchitektur auf der Basis der architekturrelevanten Funktionalitäten, die Identifikation und die Bewertung von Risiken sowie die eindeutige Bestimmung der Qualitätsattribute des Zielsystems. Die Vervollständigung der Dokumentation der funktionalen Anforderungen und die Planung der Inkremente ergänzen die Vorbereitungen für die sich anschließende Phase der Implementierung.

3.3.4.1 Analyse

Die Tätigkeiten der Analyse werden in vier Schritten wiederholt durchlaufen. Ziel dieses Arbeitspakets ist die Erstellung eines Analysemodells, das sämtliche Anforderungen an das Softwaresystem berücksichtigt. Zu diesem Zweck sollen die Anforderungen weiter konkretisiert und strukturiert werden, um so eine wartbare und stabile Systemstruktur zu erarbeiten.

Der erste Schritt umfasst die Architekturanalyse. Dabei werden so genannte *Analyse-Teilsysteme* identifiziert, die auch als *Analyse-Packages* bezeichnet werden. Diese Packages enthalten neben den Analyseklassen auch Use-Case Realisierungen und weitere Analyse-Packages. Analyseklassen charakterisieren hierbei Abstraktionen einer oder mehrer Klassen des Entwurfs, die ihrerseits funktionale Anforderungen umsetzen. Nachdem die Bestandteile des kompletten Analysemodells in Form von Analyse-Packages beschrieben sind, liegt eine Architekturbeschreibung des Systems als Ergebnis dieses Schritts vor.

Der zweite Schritt der Analyse befasst sich mit der Bestimmung und der Konkretisierung der Analyseklassen sowie der Erarbeitung der Analyse-Objektinteraktionen mithilfe von Kollaborationsdiagrammen. Diese Objektinteraktionen können zusätzlich auch in Form von kurzen Erläuterungen verfasst werden. Hinsichtlich der expliziten Unterstützung der Entwicklung kollaborativer Systeme sieht das Vorgehensmodell KollApps an dieser Stelle die Berücksichtigung der Anforderungen bezüglich differenzierender Sicherheitsmechanismen vor. Dieser Vorgang kann durch die Ergebnisse der Geschäftsprozessmodellierung der vorherigen Phase der Vorstudie sowie der vorliegenden Use-Case Diagramme unterstützt werden. Auf diese Weise wird das entworfene Rollen- und Zugriffskonzept für das Softwaresystem in der Form von

Analyseklassen und zusätzlichen Dokumenten in das Analysemodell eingebracht. Außerdem wird die Entwicklung der Datenstruktur mithilfe von Diagrammen des Entity-Relationship Modells[120] (ERM) sowie verschiedener Normalisierungsschritte durchgeführt. Die hierbei definierten Entitäten und deren Eigenschaften werden in entsprechende Methoden und Objekte der Analyseklassen überführt und dienen ebenfalls als Grundlage folgender Schritte.

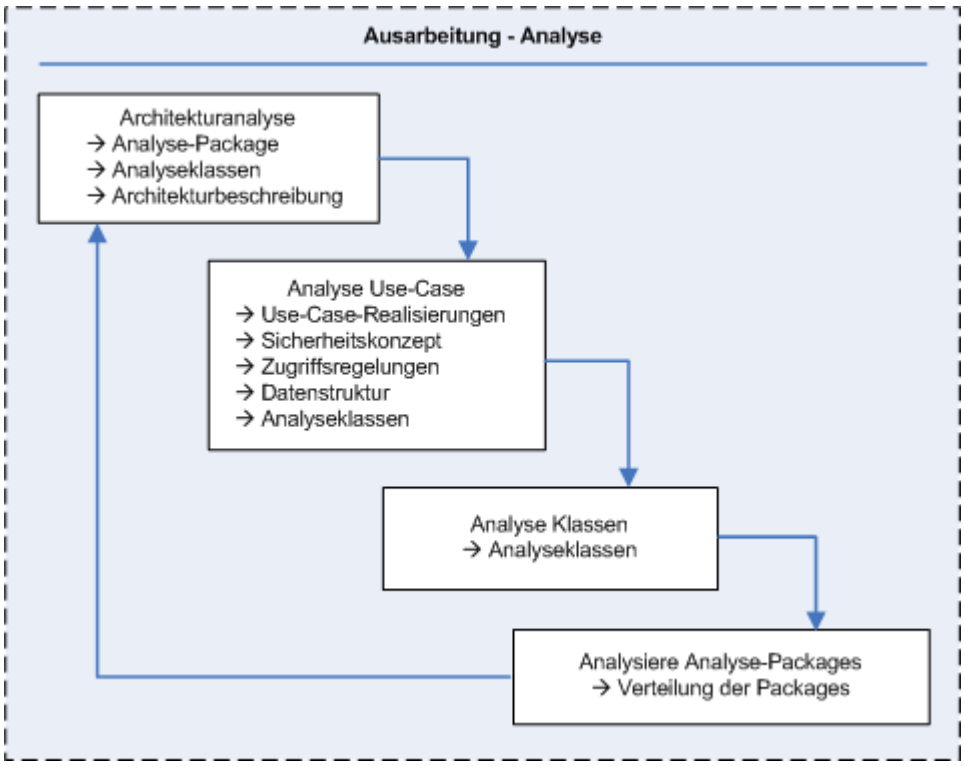


Abbildung 5 - KollApps: Phase der Ausarbeitung – Analyse[121]

Im dritten Schritt werden die so ermittelten Analyseklassen der Use-Case Realisierungen weiter untersucht und genauer bestimmt. Durch den Einsatz von ausführlichen Klassendiagrammen sollen die Verantwortlichkeiten, die unterschiedlichen Attribute und die besonderen Anforderungen an die einzelnen Klassen sowie an ihre Objekte analysiert und dokumentiert werden. Hierbei sind insbesondere die Konzepte zur Wahrung der Daten- und Zugriffssicherheit zu beachten, die bereits im ersten Schritt der Analyse definiert wurden.

[120] Weitere Informationen zu dem Entity-Relationship Modell können bei Wagner 2001 auf den Seiten 201 f. nachgelesen werden.
[121] In Anlehnung an Bunse 2001 S. 82 und Jacobson 1999 S. 173 bis S. 178.

Abschließend werden alle Teilsysteme beziehungsweise Analyse-Packages in einfach zu verwaltende Bausteine aufgespaltet. Diese Analyse-Packages dienen als Arbeitsgrundlage für die kommenden Tätigkeiten des Entwurfs.

3.3.4.2 Entwurf

Ziel des Entwurfs ist die eindeutige Bestimmung der Struktur des Softwaresystems. Diese Struktur soll dabei neben allen funktionalen und nichtfunktionalen Anforderungen auch die Systemarchitektur beschreiben. Auf der Grundlage der Ergebnisse der Analyse wird in vier Schritten (siehe Abbildung 6) ein eindeutiges Verständnis über die unterschiedlichen Anforderungen, mögliche technische Einschränkungen und über die Untergliederung des Gesamtsystems in Teilbereiche geschaffen. Die Ergebnisse der einzelnen Tätigkeiten des Entwurfs beantworten auch die Frage, ob das geplante Softwaresystem als Fat- oder Thin-Client Lösung realisiert wird.

Der erste Schritt gibt zunächst in der Form von Entwurfs- und Einsatzdiagrammen eine Übersicht. Um anschießend die Systemarchitektur zu definieren, ist die Erfassung verschiedener Bereiche notwendig. Neben der Identifikation von Netzwerkknoten und Netzwerkkonfiguration in einem Einsatzmodell, müssen auch die Entwurfsteilsysteme und deren Schnittstellen erarbeitet werden. Auch die Entwurfsklassen, die für die Architektur von besonderer Bedeutung sind, müssen in diesem Abschnitt erfasst und berücksichtigt werden.

Der zweite Schritt des Entwurfs dient der Bestimmung der Entwurfsklassen und der Teilsysteme. Im Einzelnen wird an dieser Setelle untersucht, wie die Vorgänge, die in den Use-Cases beschriebenen sind, auf die daran beteiligten Teilsysteme verteilt werden. Darüber hinaus werden die Anforderungen an die Entwurfsklassen definiert, die sich aus deren geplanten Verwendungen ergeben. Auf diese Weise können die Implementierungsanforderungen an die Use-Cases erweitert und konkretisiert werden. Auf der Basis dieser Beschreibungen werden daraufhin Testfälle kreiert, die in späteren Arbeitsschritten die Korrektheit der Implementierung dieser Use-Cases überprüfen können. Dieser spezielle Aspekt entstammt dem Vorgehensmodell Extreme Programming und dient verschiedenen Maßnahmen im späteren Verlauf des Vorgehensmodells.

Anschließend werden die Klassen entsprechend ihrer Rolle innerhalb der vorliegenden Use-Cases und nichtfunktionalen Anforderungen detailliert entworfen. Dabei werden insbesondere Operationen, Attribute, Beziehungen, Zustände und Abhängigkeiten zu den Implementierungsanforderungen festgelegt. Die konkrete Beschreibung der Realisierung von Schnittstellen schließt die Tätigkeiten dieses Arbeitsschritts ab. Die Möglichkeit einer Rückkehr zu den vorherigen Arbeitspaketen der Identifizierung dieser Entwurfsklassen wird durch das Vorgehensmodell angeboten.

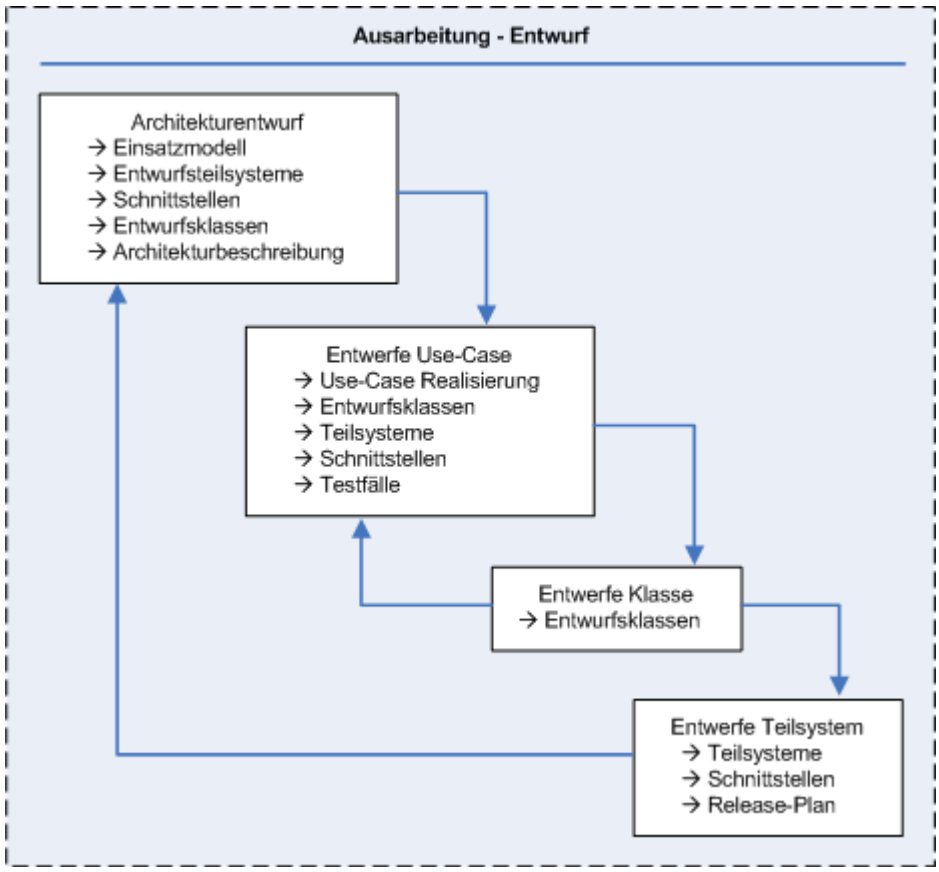


Abbildung 6 - KollApps: Phase der Ausarbeitung – Entwurf[122]

Als letzten Vorgang innerhalb des Entwurfs werden die Teilsysteme einer eingehenden Überprüfung unterzogen. Dabei soll sichergestellt werden, dass ein Teilsystem möglichst wenige Abhängigkeiten zu anderen Teilsystemen aufweist. Außerdem wird geprüft, ob die Teilsysteme die richtigen Schnittstellen aufweisen, und ob diese die angestrebten Operationen korrekt realisieren. Daneben wird ein so genannter *Release-Plan* erstellt. Diese Tätigkeit hat ebenfalls ihren Ursprung in dem Vorgehensmodell Extreme Programming und beschreibt die Zeit- und

[122] In Anlehnung an Bunse 2001 S. 83 und Jacobson 1999 S. 215 bis S. 216.

Entwicklungsplanung für die Phase der Realisation des Softwaresystems. Während die Entwickler vor allem den Aufwand für die Teilsysteme abschätzen, weist der Auftraggeber der Realisierung der einzelnen Teilsysteme und Use-Cases, entsprechend ihrer Bedeutung für das gesamte System, gesonderte Prioritäten zu.

3.3.5 Implementierung

Die Phase der Implementierung wurde von der iterativen Phase des Vorgehensmodells Extreme Programming abgeleitet. Während dieser Phase werden die User-Stories aus der Phase der Vorstudie realisiert, die nach den durchgeführten Vorarbeiten in Form von detaillierten Teilsystemen und Entwurfsklassen vorliegen. Hierbei lassen sich fünf Aktivitäten unterscheiden, die wiederholt durchlaufen werden. Akzeptanztests bilden den Übergang zu dem jeweils nachfolgenden Iterationsschritt (siehe Abbildung 7).

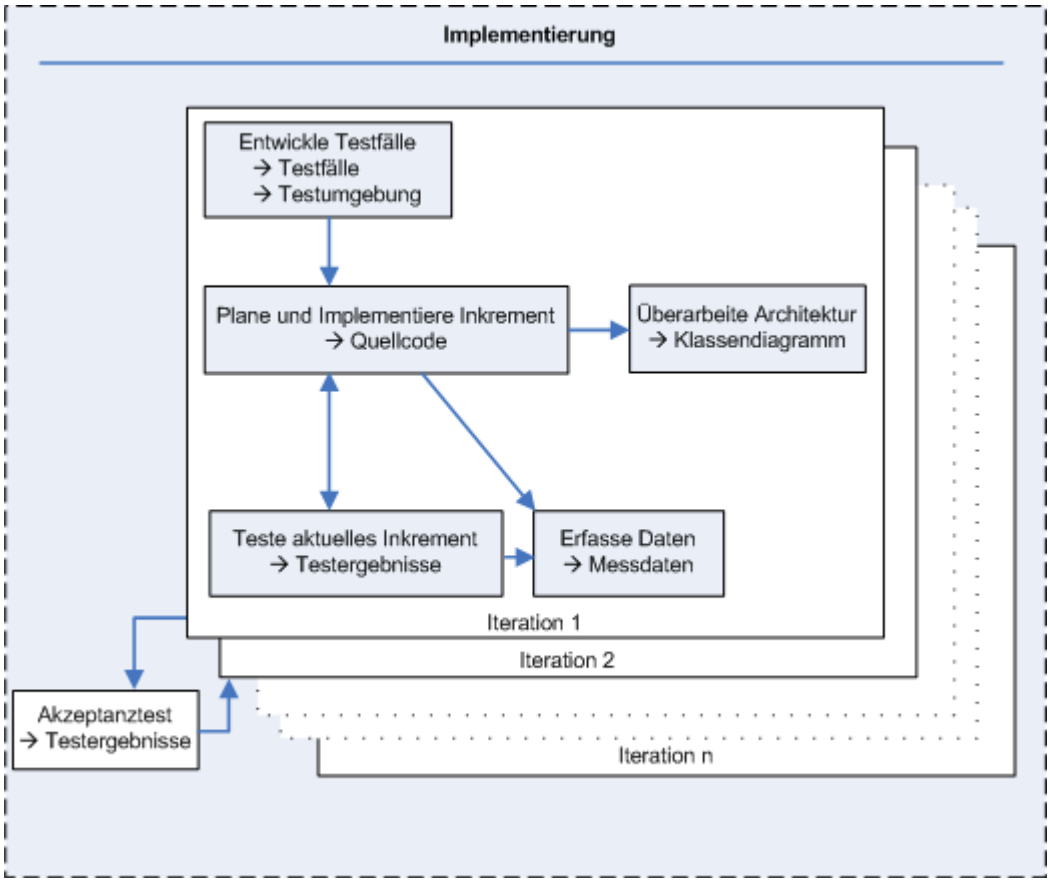


Abbildung 7 - KollApps: Phase der Implementierung[123]

Zu Beginn einer Iteration werden zunächst die Testfälle und eine entsprechende Testumgebung erzeugt, da die Tätigkeit des Testens eines der Grundprinzipien bei

[123] In Anlehnung an Bunse 2001 S. 60.

dieser Vorgehensweise ist. Automatisierbare Unit-Tests werden für die Klassen, die zu erstellen sind oder aus vorhergehenden Iterationen bereits vorliegen, implementiert beziehungsweise modifiziert. Neben der Überprüfung der Korrektheit einer Klasse dienen diese Tests auch zu deren Schutz, wenn die Klassen beispielsweise im Verlauf der weiteren Entwicklungstätigkeiten von Veränderungen direkt oder indirekt betroffen sind. Die Testfälle ergeben sich aus der konkreten Programmieraufgabe, die als Use-Case und Entwurfsklasse bereits vorliegt.

Der nächste Schritt umfasst die Planung und die konkrete Implementierung des aktuellen Inkrements. In jedem Iterationszyklus wird auf der Grundlage des Release-Plans die gerade aktuelle Iteration vorbereitet. Bei dieser Planung sollte die maximale Zykluslänge von drei Wochen nicht überschritten werden. Die User-Stories beziehungsweise Use-Cases, die in einem Iterationsschritt zu realisieren sind, werden ebenfalls anhand des Release-Plans ausgewählt. Auch die User-Stories vorhergegangener Iterationen, die den Akzeptanztest nicht bestanden haben, werden hier berücksichtigt. Die so bestimmten User-Stories und Use-Cases werden in Programmieraufträge aufgeteilt, die mit einem Aufwand von maximal drei Tagen realisierbar sind. Diese Aufträge werden den einzelnen Programmierteams zugewiesen, die die programmatische Umsetzung vornehmen. Während der Implementierung finden sämtliche Grundprinzipien des Vorgehensmodells Extreme Programming Anwendung. So wird die Entwicklung typischerweise in Teams von jeweils zwei Softwareentwicklern an jeweils einem Computer realisiert. Im Verlauf der Implementierung findet vor allem die komponentenbasierte Softwareentwicklung Berücksichtigung, deren Vorteile im Kapitel 2.2.2.5 beispielhaft anhand der Technologie der JavaServer Faces dargestellt werden.

Während der Realisierung übernommener Programmieraufträge können Anpassungen und Veränderungen an der Architektur oder bereits bestehenden Codefragmenten notwendig werden. Die Programmierteams sind deshalb in der Lage, auch Programmcode zu ändern, den sie nicht selber entwickelt haben. Allerdings muss der so veränderte Code auch nach den Modifikationen erfolgreich getestet werden können und auf diese Weise die weitere Gewährleistung seiner fehlerfreien Funktionsfähigkeit beweisen.

Ein Programmierauftrag gilt hierbei erst als vollständig abgeschlossen, wenn alle Tests erfolgreich durchlaufen werden. Diese Tests werden in dem vierten

Arbeitsschritt durchgeführt. Alle neu entwickelten beziehungsweise modifizierten Klassen müssen die entsprechenden Unit-Tests durchlaufen. Sollten während des Testens Fehler auftreten, so sind die einzelnen Programmierteams für deren Lösung verantwortlich. In diesen Fällen wird die Tätigkeit in dem zweiten Arbeitschritt der Planung und Implementierung des Inkrements wieder aufgenommen. Dieser Zyklus wiederholt sich, bis der vollständige Programmcode alle Tests erfolgreich besteht.

In dem vorletzten Arbeitspaket werden vor allem die Aufwands- und Fehlerdaten der aktuellen Aktivitäten aufgenommen, damit der aktuelle Status des Projekts jederzeit zur Verfügung steht. Treten einmal Probleme auf, können auf diese Weise unmittelbar Lösungsansätze ermittelt und durchgeführt werden.

Der Akzeptanztest bildet den Abschluss einer Iteration und zugleich den Übergang zum nächsten Iterationsschritt. Während die Unit-Tests die Implementierung einzelner Klassen überprüfen, wird hierbei das vollständige Softwaresystem am Ende jeder Iteration einem Akzeptanztest unterzogen. Dabei dienen die in der Phase der Vorstudie erstellten Testfälle als Grundlage. Extreme Programming gibt an dieser Stelle das Ziel vor, dass alle Akzeptanztests einer realisierten User-Story einwandfrei abgearbeitet werden. Treten während dieses Tests Probleme auf, so sind die jeweiligen Programmierteams für deren Lösung verantwortlich. Liegen hingegen schwerwiegende Probleme vor, so wird die aktuelle User-Story in der nachfolgenden Iteration erneut implementiert.

3.3.6 Übergabe

Die Phase der Übergabe schließt die Durchführung des Projekts ab. Hierbei steht die Durchführung von Vorbereitungsaufgaben im Mittelpunkt, so dass die Einführung des Softwaresystems problemlos erfolgen kann. Zu diesen Tätigkeiten gehören neben der Anpassung beziehungsweise Aktualisierung der vorhandenen Umgebung, auch die Fertigstellung von Benutzerhandbüchern und sonstigem Dokumentationsmaterial für das Softwareprodukt. Letzte Anpassungen der Software an die Umgebung des Kunden sowie die Behebung von möglicherweise dabei auftretenden Fehlern gehören ebenfalls in den Tätigkeitsbereich dieser Phase.

4 Prototypische Realisierung des K-Pool Everyplace

Dieses Kapitel beschreibt die Anwendung des in Abschnitt 3 erarbeiteten Vorgehensmodells KollApps bei der Entwicklung des Prototyps *K-Pool Everyplace*. Einführend wird zunächst die Applikation beschrieben und ihre Programmfunktionalitäten erläutert. Daraufhin werden die Abschnitte des Entwicklungsvorganges dargestellt, die vor allem bezüglich kollaborativer Applikationen eine besondere Bedeutung haben. Abschließend werden die Tätigkeiten, die von dem Vorgehensmodell KollApps vorgesehen sind, mit den typischen Tätigkeiten der Entwickler verglichen, die auf der Basis einer klassischen Groupware-Plattform Anwendungen implementieren.

4.1 K-Pool Everyplace

Bei diesem Softwareprodukt handelt es sich um eine Portierung der Applikation *Knowledge-Pool*[124], die auch verkürzend als *K-Pool* bezeichnet wird. Der K-Pool wird am Groupware Competence Center[125] der Universität Paderborn entwickelt und vor allem als Plattform in dem Bereich des Wissensmanagements genutzt. Da der K-Pool viele der typischen Eigenschaften aufweist, die im Abschnitt 2.1 dargestellt sind, kann diese Anwendung ebenfalls als kollaborative Applikation charakterisiert werden. Somit ist die grundlegende Bedingung für die Anwendung des Vorgehensmodells KollApps auf den vorliegenden Softwareentwicklungsprozess gegeben. Dieser Prozess beschreibt die Neuentwicklung des Knowledge-Pools unter dem Einsatz der J2EE-Technologie, die im Kapitel 2.2.2 vorgestellt wurde. Dabei dient die bereits für die Groupware-Plattform IBM Lotus Notes/Domino implementierte Anwendung als Vorlage. Der Name der neuen Applikation ergibt sich aus ihrer hohen Verfügbarkeit. Sämtliche Funktionen der Anwendung sind von beliebigen Orten aus überall (engl. everyplace) mithilfe eines Webbrowsers ausführbar. Der K-Pool Everyplace dient somit der prototypischen Realisierung des in Kapitel 3 erarbeiteten Konzeptes eines Vorgehensmodells zur Entwicklung

[124] Weitere Informationen bezüglich der Applikation Knowledge-Pool können bei Nastansky 2005 nachgelesen und der Webseite „http://gcc.uni-paderborn.de/k-pool“ entnommen werden.
[125] Weitere Informationen bezüglich des Groupware Competence Centers der Universität Paderborn können der Webseite „http://gcc.uni-paderborn.de“ entnommen werden.

kollaborativer Applikationen und liefert so den Beweis der Anwendbarkeit des Modells.

4.2 Vorstudie – Benutzeroberfläche

Während der Durchführung der Phase der Vorstudie wurde insbesondere im dritten Schritt erstmals die Benutzeroberfläche entworfen, die im weiteren Verlauf des Projekts konkretisiert und erweitert wurde. Mithilfe eines ersten Prototyps und in enger Zusammenarbeit mit dem Auftraggeber des Projekts wurde hierbei die Aufteilung des Bildschirms, die Anordnung der einzelnen Maskenelemente sowie die Auswahl des Farbschemas definiert.

Das vorliegende Kapitel beschreibt und erläutert die Ergebnisse dieser Vorgehensweise bezogen auf die Benutzeroberfläche der Applikation. Zunächst wird die Darstellungsform der Anwendung erklärt. Auch die Benutzerführung wird hierbei als Teil der Benutzeroberfläche vorgestellt.

4.2.1 Gestaltung der Benutzeroberfläche

Beim Entwurf der Benutzeroberfläche wurde auf die klassische dreigeteilte Bildschirmgestaltung zurückgegriffen, die vor allem aus dem Bereich der Web-Seiten und Web-Applikationen bekannt ist. Hierbei wird die Bildschirmfläche zunächst horizontal geteilt (siehe Abbildung 8). Der schmale obere Abschnitt dient der Darstellung des Logos der Organisation, die die Anwendung einsetzt, und der Darstellung des Produktnamens. Der breite untere Abschnitt wird daraufhin in vertikaler Richtung geteilt. Der linke Bereich wird dabei deutlich schmaler gewählt als der rechte. Dieser linke Bereich dient der Navigation. Der Benutzer kann hier die gewünschten Funktionen auswählen und so auf die von ihm vorgesehene Weise durch das Programm navigieren.

Der große Abschnitt auf der rechten Seite stellt die eigentliche Arbeitsfläche der Applikation dar. Auf dieser Fläche kann der jeweilige Inhalt betrachtet und gegebenenfalls verändert werden. Außerdem können hier mögliche untergeordnete Aktionen ausgelöst werden. Durch die Sicherheitsmechanismen, die für die geplanten Funktionen der Anwendung notwendig sind, wurde ein weiterer separater Bereich des Bildschirms bestimmt. Von dem Bereich des Logos wurde auf der rechten Seite ein kleiner Abschnitt zu dem Zweck der Anmeldung und der Abmeldung des Benutzers abgetrennt.

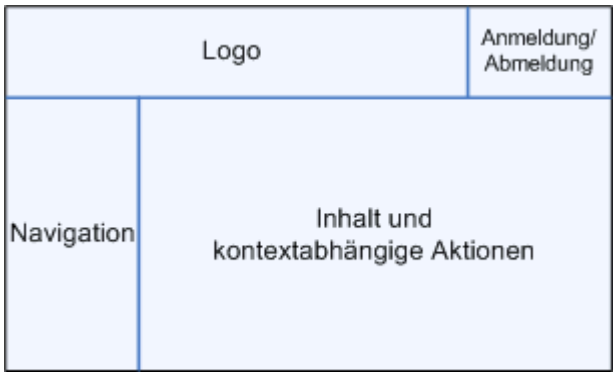


Abbildung 8 - Aufteilung der Benutzeroberfläche

4.2.2 Benutzerführung

Mithilfe der gesammelten Use-Cases und User-Stories war es möglich, eine intuitive und anwenderfreundliche Benutzerführung zu erarbeiten. Zunächst wurden die Möglichkeiten der Strukturierung und Aufbereitung der Datensätze überdacht. Hierbei hat sich eine Auswahl an übergeordneten Filterregeln als praktikabel herausgestellt, um dem Benutzer eine erste Orientierung bei der Navigation durch die zahlreichen hinterlegten Datensätze anzubieten. Somit werden die Daten mithilfe bestimmter Eigenschaften zur Anzeige gebracht. Anschließend konkretisiert der Benutzer auf der Grundlage der ersten Vorauswahl seine Auswahl der Datensätze. In einer abschließenden Übersicht werden alle verfügbaren Datensätze in Abhängigkeit zu den Themen (engl. themes), mit denen sie assoziiert wurden, dargestellt. Der Benutzer hat so die Möglichkeit, die Daten der teilweise mehrstufigen Vorauswahl aus dem Blickwinkel verschiedener Themenbereiche zu betrachten beziehungsweise zu filtern. Somit ist es beispielsweise möglich, sich zunächst alle verfügbaren Typen von Datensätzen anzeigen zu lassen, um daraufhin nur die Einträge der Datenbank auszufiltern, die beispielsweise als eine *Diplomarbeit* kategorisiert und mit dem Begriff *Groupware* als Thema assoziiert worden sind.

Neben der Möglichkeit, durch unterschiedliche und mehrstufige Filtermechanismen die gewünschten Datensätze auszuwählen, wurde auch eine Suchfunktion als eine weitere Zugriffsoption für den Anwender erarbeitet. Anhand des Suchbegriffs werden die Datensätze angezeigt, die diesen Begriff als einen Teil ihres Titels aufweisen.

Als weiteres Element wurden kleine Hilfetexte zur Vereinfachung der Benutzerführung und Hilfestellung für den Benutzer erdacht. Diese so genannten *Tooltips* erscheinen als kurze Hinweise und Erläuterungen zu Bildschirmelementen,

wenn der Benutzer für eine kurze Zeitspanne über dem jeweiligen Element mit der Maus verweilt. Es werden dem Anwender somit kompakte Hilfestellungen gegeben, die die Benutzung des entsprechenden Elements erläutern und vereinfachen.

Als letzter Punkt der Benutzerführung wurde während der Phase der Vorstudie auch die Notwendigkeit von kontextbezogenen Aktionen beziehungsweise Schaltflächen definiert. Durch den Einsatz eines Prototyps wurde herausgearbeitet, dass bestimmte Schaltflächen nur zu Zeitpunkten sichtbar und ausführbar sein sollten, wenn das aktuelle Umfeld dies sinnvoll erscheinen lässt. So ist es beispielsweise nicht sinnvoll, eine Schaltfläche zum Speichern von Änderungen an einem Datensatz anzubieten, wenn der Benutzer in der vorliegenden Ansicht gar nicht die Möglichkeit hat, die Einträge zu ändern.

4.3 Ausarbeitung

Die Vorgehensweisen und Tätigkeitsbeschreibungen dieser Phase wurden in dem Abschnitt 3.3.4 ausführlich beschrieben. In dem vorliegenden Kapitel werden die Bereiche des Sicherheits- und Zugriffsmanagements und die Strukturierung der zu speichernden Daten als wichtige Aspekte der Entwicklung kollaborativer Applikationen ausführlich betrachtet. Daneben werden auch die konkreten Entwürfe bezüglich der Systemarchitektur und der Konfigurationsmöglichkeiten erläutert.

4.3.1 Sicherheit

Wie bereits in Abschnitt 3.1.1 erläutert wurde, bilden Sicherheitsmechanismen eine der wichtigsten Grundlagen für die gemeinsame Arbeit von Anwendern auf einer gemeinsamen Datenbasis, indem sie beispielsweise für bestimmte Teilmengen von Informationen nur autorisierten Anwendern den Zugang gewähren. Dieser Abschnitt beschreibt mithilfe von Use-Cases und User-Stories die Definition der Sicherheitsarchitektur. Anschließend wird die identifizierte Architektur in dem Schritt des Entwurfs detaillierter ausgearbeitet und dargestellt.

4.3.1.1 Erarbeitung des Sicherheitskonzepts

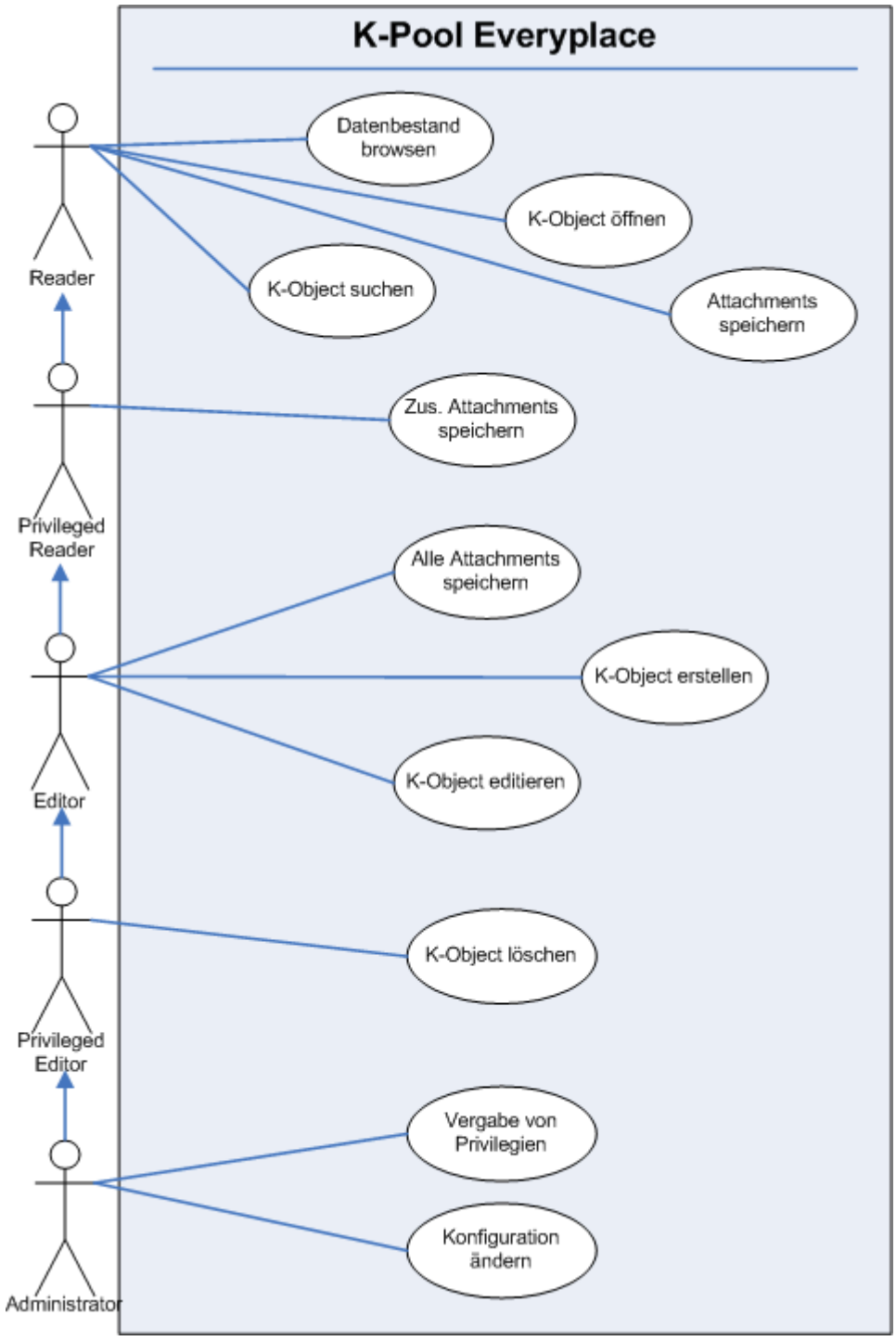


Abbildung 9 - Use-Case Diagramm des K-Pool Everyplace

Aus den Use-Cases, die in Abbildung 9 zusammengefasst dargestellt werden, lassen sich die verschiedenen Sicherheitsstufen ablesen, denen ein Benutzer zugeordnet werden kann. Es handelt sich hierbei um fünf unterschiedliche Funktionen, die in Form von abstrakten Rollen berücksichtigt werden. Diese Rollen werden über eine interne Zuordnungstabelle den jeweiligen Benutzern anhand ihres Benutzernamens

zugeordnet. Jeder Benutzer darf somit ausschließlich die Tätigkeiten wahrnehmen können, die für die Rolle zugelassen sind, die ihm innerhalb des Softwaresystems zugewiesen wurde. Zu diesem Zweck ist eine Anmeldung des Benutzers am Softwaresystem notwendig. Authentifiziert sich ein Anwender mit seinem korrekten Benutzernamen und Passwort, so wird ihm für die Zeit der Arbeitssitzung die entsprechende Rolle zugeordnet. Erst ein korrekt authentifizierter Benutzer kann somit die Tätigkeiten ausführen, die seiner Rolle entsprechen.

4.3.1.2 Der anonyme Benutzer

Anhand einer User-Story (siehe Abbildung 10) wurde eine weitere Regelung bezüglich der Sicherheitsmechanismen erarbeitet. Meldet sich ein Anwender nicht an dem Softwaresystem an, so kann das System dem Anwender nicht die Befugnisse einräumen, die ihm möglicherweise zuständen. Ein solcher Anwender wird auch als *anonymer Benutzer* bezeichnet. Diesem nicht authentifizierten Benutzer wird automatisch das niedrigste Zugriffsniveau eingeräumt. Dabei handelt es sich um die Rolle des *Readers*, dessen Tätigkeiten und Befugnisse in dem nachfolgenden Abschnitt näher erläutert werden.

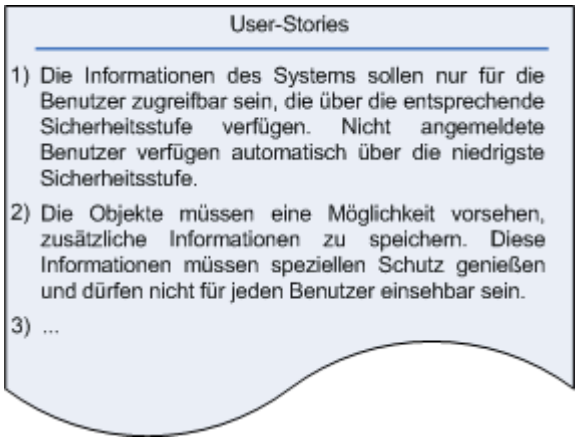


Abbildung 10 - Beispiele für User-Stories

4.3.1.3 Der mehrstufige Zugriff auf Objekte

Das System von Zugriffsregelungen, welches sich aus den Use-Cases und verschiedenen User-Storys ergibt, besteht aus fünf unterschiedlichen Abstufungen beziehungsweise Rollen. Dabei ist zu beachten, dass die Rollen höherer Zugriffsebenen alle Befugnisse und Möglichkeiten der Ebenen einschließen, die sich in der Rollen-Hierarchie unterhalb dieser Rolle befinden. Der Umfang an

zugeordneten Befugnissen nimmt also von der Rolle des *Readers*, *Privileged Readers*, *Editors*, *Privileged Editors* bis hin zur Rolle des *Administrators* weiter zu.

Die Rolle des *Readers* stellt lediglich die Grundfunktionen des Softwaresystems bereit. Der Benutzer ist in der Lage, alle Dokumente zu durchsuchen, die für diese niedrigste Zugriffsebene freigegeben sind. Die Datensätze können zwar geöffnet und betrachtet werden, aber Veränderungen der Feldinhalte sind in diesem so genannten *Read-Modus* nicht zugelassen. Innerhalb der Datenobjekte ist der Zugriff ebenfalls nur auf die Felder und Datenressourcen möglich, die für diese Zugriffsstufe freigegeben sind. So ist beispielsweise das Register „Additional information“ mit weiterführenden und möglicherweise sensiblen Inhalten überhaupt nicht zugänglich. Auch das Abspeichern von hinterlegten Dateiressourcen beschränkt sich auf die Dateien, deren Zugang für die Rolle Reader explizit eingeräumt wurde.

Die Rolle des *Privileged Readers* erweitert die Befugnisse eines Benutzers, dem die Rolle eines Readers zugewiesen wurde. Zusätzlich ist ein Privileged Reader in der Lage, auch Dateiressourcen lokal zu speichern, die für diese Sicherheitsebene freigegeben wurden.

Ein Anwender, der anhand seines Benutzernamens und seines Passwortes die Rolle des *Editors* zugewiesen bekommt, genießt gegenüber dem Privileged Reader zusätzliche, weiterführende Privilegien. Ein Editor ist in der Lage, auf alle gespeicherten Dokumente sowie deren Feldinhalte und Dateianhänge zuzugreifen. Hierzu gehört auch das besonders beschützte Register „Additional information“ und die in diesem Register hinterlegten Dateiressourcen. Darüber hinaus kann ein Anwender, der die abstrakte Funktion eines Editors wahrnimmt, die Datenobjekte im so genannten *Edit-Modus* betrachten und modifizieren. Sämtliche Felder bestehender Datenobjekte können in diesem Modus frei verändert werden. Außerdem können hier alle weiteren Eigenschaften eines Datenobjektes, wie beispielsweise zugeordnete Stichworte oder Themenverweise, verändert und erweitert werden. Diese Veränderungen können anschließend in die Datenquellen zurück geschrieben werden. Eine weitere Fähigkeit dieser Anwender ist es, neue Datenobjekte zu erstellen und so die Datenbasis zu erweitern.

Benutzer des Softwaresystems, denen als Rolle *Privileged Editor* zugeordnet wird, haben neben den Befugnissen der Rolle Editor auch das Privileg, bestehende Datenobjekte zu löschen. Diese Fähigkeit wurde innerhalb der Rollenhierarchie

besonders weit oben vorgesehen, damit nur ein autorisierter und entsprechend qualifizierter Personenkreis Daten dauerhaft entfernen darf.

Die Rolle des *Administrators* schließt alle Befugnisse der bisher genannten Rollen ein und steht somit an der Spitze der Rollenhierarchie. Ein Administrator ist darüber hinaus in der Lage, die gesamte Konfiguration der Anwendung zu verändern. Dazu gehört einerseits das Hinterlegen eines entsprechenden Logos der Organisation, die Spezifikation des Servers, der das Adressbuch mit den Benutzernamen zur Authentifizierung bereitstellt sowie die umfangreichen Veränderungen und Vorgaben, die ein Administrator für Stichworte, Themenverweise und Objekttypen vornehmen kann. Des Weiteren erlaubt die Rolle des Administrators, die so genannte *Access control list* zu modifizieren. Hier werden alle Benutzernamen und deren zugehörige Rolle verwaltet. Ein Anwender, der über die Privilegien der Rolle Administrator verfügt, hat damit die Möglichkeit, sämtliche Benutzerberechtigungen zu verändern und zu erweitern. Die Rolle des Administrators sollte aufgrund dieser umfangreichen Befugnisse nur einem eingeschränkten Benutzerkreis eingeräumt werden.

4.3.1.4 Authentifizierung via LDAP

Alle Sicherheitsmechanismen, die in den vorangehenden Abschnitten beschrieben wurden, basieren auf der Realisierung eines Prozesses zur Authentifizierung des Benutzers durch das Softwaresystem. Die Analyse der nichtfunktionalen Vorgaben beziehungsweise Anforderungen ergab, dass die Daten, auf denen diese Benutzerverwaltung aufbaut, nicht in einem zusätzlichen Benutzerverzeichnis gespeichert werden. Vielmehr ist die Integration eines bereits bestehenden Benutzerverzeichnisses der Organisation, die das System einsetzt, vorgesehen. Diese Vorgehensweise vermeidet redundante Datenspeicherung und die damit verbundenen Problemen der Dateninkonsistenz sowie die des Mehraufwands für die Wartung und Sicherung der zusätzlichen Datenbestände. Das gesamte System wird mit der Anbindung bestehender Verzeichnisse flexibler und transparenter für die Administratoren, die die entsprechende Systemumgebung pflegen. Um den Zugriff

auf ein bestehendes Benutzerverzeichnis zu realisieren, wurde eine Verbindung mithilfe des *Lightweight Directory Access Protocol*[126] (LDAP) entworfen.

Bei LDAP handelt es sich um ein Netzwerkprotokoll, das die Kommunikation zwischen einem *LDAP-Client* und einem *Directory-Server* herstellt. Hierbei entspricht der Computer des Benutzers dem LDAP-Client und der Directory-Server dem Server, der das jeweilige Benutzerverzeichnis der Organisation über einen entsprechenden LDAP-Service den Client-Rechnern anbietet. Der Entwurf des Prototyps sieht die Anbindung eines IBM Lotus Domino Servers in der Version 6.5.2 vor, der das Benutzerverzeichnis, das auch als *Domino Directory* bezeichnet wird, mithilfe eines LDAP-Services anbietet. Bei dieser Integration wird der Domino Server zur Verifizierung des eingegebenen Benutzernamens und des entsprechenden Passwortes genutzt. Diese Vorgehensweise bietet einen erweiterten Bedienungskomfort für die Anwender des Systems, da ihnen kein zusätzlicher Benutzername und Kennwort zugewiesen werden. Vielmehr können die Benutzer den bereits bekannten Namen und das entsprechende Internet-Passwort des Domino Benutzerverzeichnisses ebenfalls für den K-Pool Everyplace verwenden.

4.3.2 Datenstruktur

Der Datenbankentwurf beschreibt die Struktur, in der die Daten dauerhaft gespeichert werden. Dieser Entwurf wird auf der Grundlage vieler Einflussfaktoren erarbeitet. Hierbei spielen neben den klassischen Feldinhalten auch nichtfunktionale Anforderungen wie zum Beispiel technische Vorgaben eine große Rolle. Darüber hinaus müssen auch die Informationen entsprechend abgelegt werden, die für das Sicherheitskonzept wichtig sind. Durch die Vorgabe, so genannte *Service Data Objects*[127] (SDO) als gekapselte Datenobjekte auf der Basis der Java-Technologie einzusetzen, war zugleich eine relationale Datenbank als Speicherort festgelegt, da eine Anbindung von beispielsweise Datenbanken der Groupware-Plattform IBM Lotus Notes/Domino zum Zeitpunkt der Erstellung dieser Ausarbeitung noch nicht möglich ist. Die Datenstruktur des hier vorgestellten Prototyps wurde somit auf dem Wege des klassischen Datenbankentwurfs erstellt. Anhand aller vorliegenden

[126] Weitere Informationen zum Lightweight Directory Access Protocol sowie dessen programmatische Implementierung können bei Howes 1997 nachgelesen werden.
[127] Weitere Informationen zu der Technologie der Service Data Objects können bei Wahli 2004 auf den Seiten 297 bis 378 nachgelesen werden.

Informationen wurde ein Entity-Relationship Diagramm erarbeitet und als Grundlage für das Datenschema genutzt. Normalisierung und weitere Anpassungen lieferten den finalen Datenbankentwurf, der in Form einer IBM DB2 Universal Database[128] realisiert wurde.

4.3.3 Thin- vs. Fat-Client

Als Teil der Entwurfsphase wurde ebenfalls die Entscheidung zwischen einer Thin- oder Fat-Client-Architektur getroffen. Insbesondere die nichtfunktionale Vorgabe, die Technik der *JavaServer Faces*[129] (JSF) bei der Realisierung des Prototyps zu berücksichtigen, ließ beim aktuellen Stand der Entwicklung nur eine Thin-Client-Lösung sinnvoll erscheinen. Diese Lösung spiegelt dabei die typischen Vor- und Nachteile einer Architektur wider, die den größten Teil ihrer Programm- und Geschäftlogik auf dem Server ausführen lässt und den Client lediglich zu Zwecken der Berechnung der Benutzeroberfläche sowie deren Darstellung nutzt.[130]

4.3.4 Konfigurationsumgebung

Die Konfigurationsumgebung des K-Pool Everyplace bleibt den Benutzern vorbehalten, die die Rolle des Administrators innerhalb des Softwaresystems wahrnehmen. Die Analyse der vielfältigen Eigenschaften und Verknüpfungen, die ein Datenobjekt aufweisen kann, ergab die Notwendigkeit einer zentralen Umgebung zur Wartung und Erweiterung der zugrunde liegenden Datenbasis. Der Entwurf dieser Umgebung sieht verschiedene Bereiche zur Konfiguration vor. Auf der einen Seite werden unter den Navigationspunkten „Application“ und „Access Control List“ vor allem Eigenschaften des K-Pool Everyplace definiert, die dem organisatorischen Bereich zuzuordnen sind. Beispielsweise kann hier das zu verwendende Logo der Organisation hochgeladen werden, die Netzwerkadresse und der Port des Domino Servers angegeben werden und die Zuordnung der unterschiedlichen Rollen der Anwender zu ihren Benutznamen verändert und erweitert werden. Die andere Seite der Konfigurationsumgebung bezieht sich vor allem auf die inhaltlichen Grundlagen der Applikation. So können zum Beispiel unter anderem neue Typen von

---

[128] Weitere Informationen zu dem Produkt IBM DB2 könne der Webseite „http://www-306.ibm.com/software/data/“ entnommen werden.
[129] Die Technologie der JavaServer Faces wird im Kapitel 2.2.2.5 ausführlich beschrieben und erläutert.
[130] Eine ausführliche Diskussion der Vor- und Nachteile von Fat- und Thin-Clients kann bei Erdmann 2003b nachgelesen werden.

Datenobjekten erstellt und verändert, das Verzeichnis der Themenverweise gewartet und die Sammlung der Stichworte den aktuellen Umständen und Anforderungen angepasst werden.

4.4 Implementierung - Komponentenorientierter Aufbau

Die erarbeitete Vorgehensweise dieser Phase wurde im Kapitel 3.3.5 ausführlich erläutert. Um die vielfältigen Vorteile, die eine komponentenorientierte Entwicklung bietet, nutzen zu können, wurde der Einsatz von Komponenten der JSF-Technologie (siehe Abschnitt 2.2.2.5) als Bausteine der Benutzeroberfläche bereits vorgegeben. Die nachfolgenden Abschnitte beschreiben die bereits vorhandenen JSF-Komponenten, die zur Realisierung der Präsentationsebene des Softwaresystems genutzt wurden. Darüber hinaus wird die Erstellung einer weiteren JSF-Komponente erläutert. Anhand dieses Vorganges soll die Vorgehensweise, die das Modell KollApps bezüglich der Phase der Implementierung vorgibt, beispielhaft dargestellt und belegt werden.

4.4.1 Standard- und IBM-Komponenten

Zur Implementierung des Prototyps wurde die Integrierte Entwicklungsumgebung (engl. Integrated Development Environment, IDE) *WebSphere Studio Application Developer* eingesetzt, die in nachfolgenden Abschnitten verkürzend mit WSAD bezeichnet wird. WSAD bietet in der verwendeten Version 5.1.2 eine Vielzahl vorgegebener JSF-Komponenten an, die unmittelbar zur Implementierung von Applikationen verwendet werden können. Dabei handelt es sich einerseits um so genannte *Standard-Komponenten*, die bereits durch die Referenzimplementierung des JSF-Standards vorgegeben sind. Auf der anderen Seite existiert eine Sammlung zusätzlicher Komponenten, die von den Entwicklern der IDE hinzugefügt wurde. Diese zweite Gruppe von Komponenten wird auch als Gruppe der *IBM-Komponenten* bezeichnet. Abbildung 11 gibt einen Überblick über die Komponenten, die von der IDE WebSphere Studio Application Developer bereitgestellt werden.

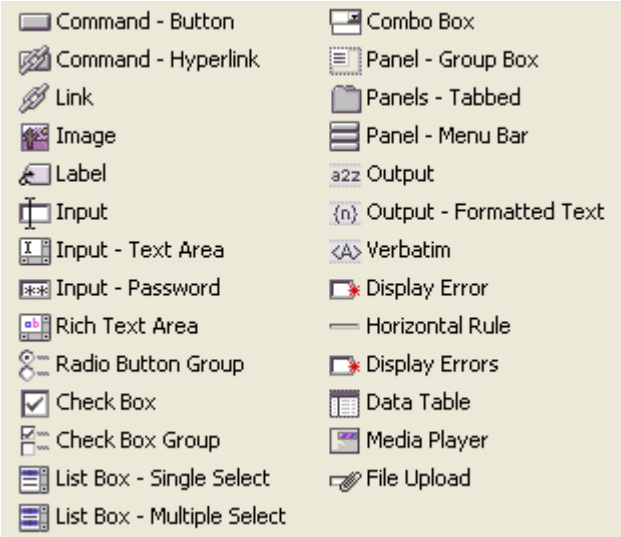


Abbildung 11 - Übersicht JSF-Komponenten

4.4.2 TreeStructure Komponente

Bei der TreeStructure Komponente handelt es sich um eine JSF-Komponente, die zusätzlich zu den Standard- und IBM-Komponenten (siehe Abschnitt 4.4.1) entworfen und implementiert wurde. Die Komponente stellt die Themenverweise, die ein Datenobjekt des K-Pool Everyplace ausweist, in Form eines hierarchischen Baumes dar (siehe Abbildung 12). Zunächst werden von den hierarchisch untergliederten Themenverweisen nur die Einträge der obersten Ebene angezeigt. Interessiert sich der Benutzer für Verwiese, die unterhalb der ersten Ebene liegen, so öffnet ein Klick mit der Maus auf das entsprechende kleine Pluszeichen vor dem gelben Ordner die darunter liegende Ebene. Dieser Vorgang kann bis in fast beliebige Tiefe fortgesetzt werden. Ein weiterer Klick auf den gewünschten Themenverweis öffnet die entsprechende Übersicht über alle Datenobjekte, denen der gerade ausgewählte Themenverweis zugeordnet wurde.

Diese Art der Darstellung und das interaktive Verhalten der TreeStructure Komponente entsprechen unter anderem denen des Windows-Dateiexplorers, der die gespeicherte Ordnerstruktur in derselben Weise anzeigt. Somit sollte die Verwendung der TreeStructure Komponente auch Anwendern, die erstmalig mit dem K-Pool Everyplace arbeiten, einen einfachen und direkten Einstieg in die Bedienung der Applikation ermöglichen.

Da die Funktionen der TreeStructure Komponente an verschiedenen Stellen der Applikation benötigt werden, bietet sich die Realisierung in Form einer wieder verwendbaren JSF-Komponente an. In den nachfolgenden Abschnitten wird die

Entwicklung der JSF-Komponente TreeStructure entsprechend der Phase der Implementierung des Vorgehensmodells KollApps erläutert und beschrieben.

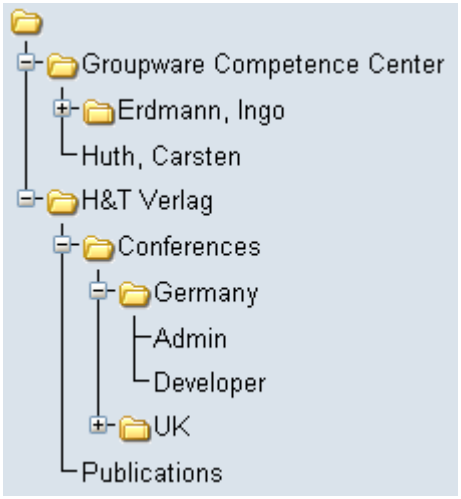


Abbildung 12 - JSF-Komponente TreeStructure

4.4.2.1 Testfälle mit JUnit

Der erste Schritt der Implementierung sieht die Erarbeitung von Testfällen vor (siehe Abschnitt 3.3.5). Diese Testfälle überprüfen und sichern die korrekte Ausführung der erstellten und der zu erstellenden Klassen. Alle implementierten Testfälle wurden mithilfe von *JUnit*[131] entwickelt. Bei JUnit handelt es sich um ein Java-Framework, dass die Realisierung, Ausführung und Wiederverwendung von Testfällen vielfach unterstützt. Die einzelnen Testfälle werden auch als *TestCases* bezeichnet. Mehrere solcher Testfälle lassen sich zu so genannten *TestSuites* zusammenfassen und in einem Schritt überprüfen.

Die Version 3.8.1 von JUnit[132], die innerhalb von WSAD integriert ist und in dieser Arbeit verwendete wird, bietet neben zusätzlichen Hilfestellungen bei der Erstellung der Testklassen ebenfalls eine detaillierte grafische Auswertung der Testergebnisse an. Auf diese Weise lassen sich mögliche Fehler, die während der Durchführung der einzelnen Tests auftreten, leicht lokalisieren und unmittelbar innerhalb der Klassen der Applikation korrigieren. Die Testabläufe können hierbei weitgehend automatisiert durchlaufen werden.

---

[131] Weitere Informationen zum Java-Framework JUnit können der Webseite „http://www.junit.org“ entnommen werden.
[132] Weitere Informationen zum Einsatz von JUnit innerhalb von WSAD können bei Carew 2003 ausführlich nachgelesen werden.

Im Rahmen dieser Ausarbeitung wird das Erstellen von Testfällen beispielhaft anhand einer Klasse gezeigt, die als Grundlage für die TreeStructure Komponente dient. Es handelt sich dabei um die Klasse *Node.java*, deren Instanzen die einzelnen Knoten des gesamten hierarchischen Baumes von Themenverweisen sowie deren Eigenschaften charakterisieren.

```
public void testHasChild()
{
    _node.setHasChild(true);
    assertTrue("Please review node's method setHasChild()"
    ,_node.hasChild());
}
public void testGetId()
{
    int expected = 8;
    _node.setId(expected);
    assertEquals("Please review node's methods getId() and setId()"
    ,expected, _node.getId());
}
```

Abbildung 13 - JUnit Testfälle

Die Abbildung 13 zeigt den typischen Aufbau von JUnit Testfällen. Zunächst wird einer Instanzvariablen des bereits abgeleiteten Objekts der zu testenden Klasse durch eine Methode ein bestimmter Testwert zugewiesen. Mit einer so genannten *assert-Methode*[133] kann auf unterschiedlichen Wegen überprüft werden, ob die Verarbeitung des Testwerts korrekt abgelaufen ist. So ist es beispielsweise möglich, bestimmte Instanzvariablen mithilfe der entsprechenden Methoden auszulesen und den zurück gelieferten Wert mit dem ursprünglich eingetragenen Testwert zu vergleichen. Stimmen diese beiden Werte nicht überein, so wird eine Fehlermeldung bezüglich der Diskrepanz zwischen diesen Werten während der Testausführung ausgegeben. Darüber hinaus erlaubt ein Großteil der assert-Methoden, individuelle Fehlernachrichten zu definieren, die den Entwicklern zusätzliche Hinweise zur Korrektur geben können.

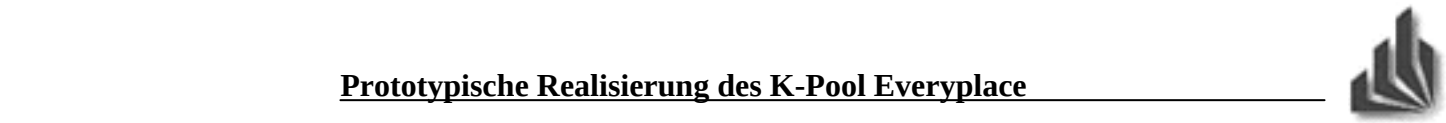
Abbildung 13 zeigt zwei konkrete Testfälle, die die Methoden *setHasChild()*, *hasChild()*, *setId()* und *getId()* der Klasse *Node* überprüfen. Der erste Fall überprüft die korrekte Ausführung der Methode *setHasChild()*, die der Instanzvariablen *hasChild* einen Wahrheitswert zuordnet und somit anzeigt, ob der vorliegende Knoten weitere untergeordnete Knoten besitzt. Zu diesem Zweck wird der

[133] Einen ausführlichen Überblick über die unterschiedlichen assert-Methoden gibt die Webseite http://www.junit.org/junit/javadoc/3.8.1/junit/framework/Assert.html

Instanzvariablen *hasChild* einer Testinstanz durch die Methode *setHasChild()* der Wert *true* zugewiesen. Die Methode *hasChild()* liest den Wert eben dieser Variable aus und liefert ihn an die Methode *assertTrue()* zurück. Diese überprüft ihrerseits, ob der gelieferte Rückgabewert dem Wahrheitswert *true* entspricht. Ist dies nicht der Fall, wird ein Fehler zurückgeliefert und die zusätzlich übergebene Fehlernachricht wird auf dem Bildschirm zur Anzeige gebracht. Der zweite Testfall zeigt die Überprüfung der schreibenden und lesenden Methoden des Attributs *Id*, das eine eindeutige Identifikationsnummer der jeweiligen Instanz darstellt. Die Methode *AssertEquals()* prüft somit, ob der Wert, der zuvor in die Instanzvariable geschrieben wurde, auch wieder korrekt ausgelesen werden kann. Treten hierbei Abweichungen auf, so wird ein Fehler zurückgeliefert. Außerdem wird eine angepasste Fehlermeldung ausgegeben, die der Methode zum Zeitpunkt des Aufrufs übergeben worden ist.

4.4.2.2 Release-Plan und Realisierung

Der zweite Schritt der Implementierung sieht die Erstellung eines Release-Plans und die Durchführung der Implementierungsarbeiten vor (siehe Abschnitt 3.3.5). Abbildung 14 zeigt in Form eines abgewandelten Gantt-Diagramms den erarbeiteten Release-Plan für die Realisierung der TreeStructure Komponente. Die vorgegebene maximale Gesamtdauer eines Zyklus von drei Wochen wurde dabei eingehalten. Die Arbeitspakete wurden so gewählt, dass deren Implementierung innerhalb von maximal drei Tagen durchzuführen ist. Die ersten drei Arbeitsprozesse beschreiben die Realisierung des Imports der Informationen bezüglich der Knoten und deren Verknüpfungen sowie die Aufbereitung der Daten für die kommenden Verarbeitungsschritte. Während des vierten Schritts wird der hierarchische Baum in rekursiver Vorgehensweise zusammengesetzt und auf dem Bildschirm angezeigt. Anschließend werden die ersten Bedienungselemente in Form von Plus- und Minussymbolen vor den Knotennamen hinzugefügt. Diese Symbole werden zugleich mit einem interaktiven Klappmechanismus versehen. Dieser Mechanismus erlaubt es dem Benutzer, tiefer liegende Ebenen von Themenverweisen zu minimieren oder zu maximieren. Der sechste Schritt erweitert die grafische Anzeige der TreeStructure Komponente um kleine Ordnersymbole vor den Themenverweisen. Diese Symbole zeigen dem Benutzer an, ob ein Themenverweis sich aktuell in maximiertem oder minimiertem Zustand befindet. Das vorletzte Arbeitspaket beschreibt die



Realisierung der Hyperlinks, die hinter den Themenverweisen hinterlegt sind. Die repräsentierten Adressen der Hyperlinks sind an den jeweiligen Themenverweis angepasst. Um den Bedienungskomfort weiter zu verbessern, sieht der letzte Arbeitsschritt die Realisierung eines so genannten *MouseOver* Effekts vor. Dieser Effekt erhöht durch den Einsatz farblicher Hervorhebungen ebenfalls die Benutzerfreundlichkeit der Komponente.

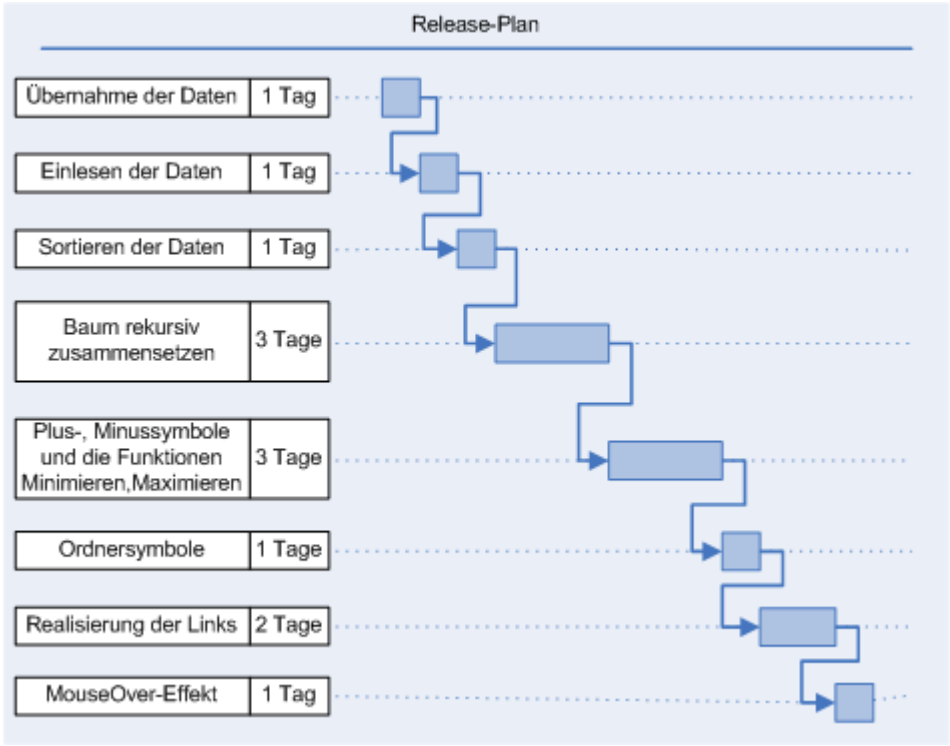


Abbildung 14 - Release-Plan der TreeStructure Komponente

4.4.2.3 Erhebung von Messdaten

Die Erfassung von Aufwands- und Fehlerdaten ergab, dass vor allem die Schritte der rekursiven Zusammensetzung des hierarchischen Baumes mehr Zeit in Anspruch nehmen, als zunächst laut des Release-Plans für diese Aufgabe kalkuliert wurde. Vor allem unerwartete Probleme in der exakten Darstellung der TreeStructure Komponente innerhalb des Browsers ließen die Entwicklungszeit für dieses Arbeitspaket stark ansteigen.

Die auftretenden Fehler in der genauen Positionierung der einzelnen Teilgrafiken, die gemeinsam den darzustellenden Baum ergeben, waren Anlass, den Release-Plan und die gesamte Kalkulation des Projektes zu überarbeiten. Hierbei stellte sich heraus, dass der Mehraufwand dieser Teilaufgabe durch die Verkürzung des Zeitaufwandes für nachfolgende Tätigkeiten fast vollständig kompensiert werden kann. Da während

der Realisation der rekursiven Zusammensetzung des Baumes auch gleichzeitig einige Vorarbeiten für die nachfolgenden Tätigkeiten der Implementierung der Hyperlinks und des MouseOver Effektes durchgeführt worden sind, kann für diese beiden nachfolgenden Tätigkeiten ein entsprechend geringerer Zeitaufwand bestimmt werden. Der Zeitverzug ließ sich auf diese Weise nahezu vollständig ausgleichen.

4.4.2.4 Durchführung des Akzeptanztests

Der Akzeptanztest für die Realisierung der TreeStructure Komponente ist erfolgreich verlaufen. Die erarbeitete Komponente weist alle Eigenschaften auf, die in der zugrunde liegenden User-Story beschrieben sind. Die Funktionalität der Hyperlinks ist korrekt implementiert und sowohl das Design, als auch die Bedienung der gesamten TreeStructure Komponente entspricht den erarbeiteten Vorgaben.

4.5 Übergabe

Die Phase der Übergabe beschreibt sämtliche Tätigkeiten zur Vorbereitung und Durchführung der Systeminstallation (siehe Abschnitt 3.3.6). Zunächst wurde ein Computersystem zur Verfügung gestellt, welches über die nötige Hardware verfügt. Neben einem Betriebssystem und einem *IBM WebSphere Application Server*[134] in der Version 5.1.1 wurde zur Datenspeicherung eine *IBM DB2 Universal Database*[135] als Express Edition V8.1 installiert. Als weitere Vorarbeiten wurde eine Datenbank mit dem entsprechenden Datenbankschema erstellt, und die Applikation K-Pool Everyplace auf dem WebSphere Application Server als zusätzliche Anwendung installiert. Spezielle Einstellungen verknüpften die K-Pool Everyplace Anwendung mit der bereits installierten Datenbank und dem Domino Server, der das Benutzerverzeichnis für die Anwendung über das erläuterte LDAP-Protokoll zur Verfügung stellt. Der Übergabeprozess wurde durch eine abschließende Präsentation sämtlicher Funktionen und Systemeigenschaften erfolgreich abgeschlossen.

[134] Weitere Informationen zu dem Produkt *IBM WebSphere Application Server* können der Webseite „http://www-306.ibm.com/software/websphere/“ entnommen werden.
[135] Weitere Informationen zu dem Produkt IBM DB2 Universal Database können der Webseite „http://www-306.ibm.com/software/data/db2/“ entnommen werden.

4.6 Abgrenzung zur Implementierung auf der Basis einer klassischen Groupware-Plattform

Der vorliegende Abschnitt stellt eine Abgrenzung zwischen der Entwicklung kollaborativer Applikationen auf Basis einer klassischen Groupware-Plattform wie zum Beispiel IBM Lotus Notes/Domino und der Entwicklung kollaborativer Applikationen mithilfe des Vorgehensmodells KollApps, das in Kapitel 3 erarbeitet wurde, dar. Die nachfolgenden Kapitel vergleichen diese beiden Vorgehensweisen und arbeiten sowohl die Unterschiede als auch mögliche Gemeinsamkeiten heraus.

4.6.1 Sicherheit

Wie in Kapitel 2.2.1.3 beschrieben, verfügen klassische Groupware-Plattformen über umfangreiche bereits integrierte Sicherheitskonzepte. Diese Konzepte lassen sich während der Entwicklung komfortabel in neue Applikationen einbetten. Vergleichsweise leicht können beispielsweise differenzierende Rollenkonzepte verwirklicht werden. Darüber hinaus wird die gesamte Benutzerverwaltung bereits durch die zugrunde liegende Groupware-Plattform bereitgestellt. Alle diese Funktionen müssen bei der Vorgehensweise, die das KollApps Vorgehensmodell beschreibt, neu entworfen und programmiert werden, da typischerweise den Projekten keine der beschriebenen Mechanismen anfänglich zur Verfügung stehen. Dem Mehraufwand für den Entwurf und die Implementierung dieser Systemeigenschaften steht eine erhöhte Flexibilität bei dem Entwurf des gesamten Softwaresystems entgegen. So kann der Entwurf des Systems sich ausschließlich an den Anforderungen orientieren, ohne die Vorgaben der bereits integrierten Konzepte berücksichtigen zu müssen. Außerdem kann auf diese Weise sichergestellt werden, dass nur genau der Funktionsumfang implementiert wird, der tatsächlich durch die Applikation benötigt wird.

4.6.2 Verfügbarkeit

Die Vorgehensweise, die durch das Modell KollApps beschrieben wird, und die Arbeitsschritte eines Entwicklers einer klassischen Groupware-Umgebung ähneln sich bezogen auf die Verfügbarkeit der Applikation. Beide Strategien sehen eine Abwägung der Vor- und Nachteile von Fat- beziehungsweise Thin-Client Lösungen vor. Bei den klassischen Groupware-Umgebungen liegt zwar der Fokus deutlicher auf der Fat-Client-Lösung, aber es existieren vereinzelte Situationen, in denen auch

in diesem Umfeld eine Thin-Client Lösung sinnvoll erscheinen kann.[136] Das Vorgehensmodell KollApps sieht die situationsbezogene Abwägung der Argumente, die für beziehungsweise gegen eine Fat- oder Thin-Client Architektur sprechen, explizit in der Phase der Ausarbeitung (siehe Abschnitt 3.3.4) vor.

4.6.3 Sessionmanagement und Skalierbarkeit

Das Management der einzelnen Sitzungen (engl. sessions) der Anwender wird innerhalb von Groupware-Umgebungen intern vorgenommen. Auf diese Prozesse kann auch bei der Entwicklung kollaborativer Applikationen komfortabel zurückgegriffen werden, während die Entwicklungsarbeiten mithilfe von KollApps die Implementierung von Mechanismen zur Verwaltung der unterschiedlichen Sessions vorsehen. Es existieren zwar technische Erweiterungen, die dieses Sessionmanagement vereinfachen (siehe Abschnitt 2.2.2.5), jedoch erreichen sie nicht den Umfang und den Integrationsgrad, der bei klassischer Groupware vorgegeben wird. Der Grad der Skalierbarkeit der kollaborativen Systeme, die mithilfe von KollApps geplant und realisiert werden, ist als vergleichsweise hoch zu definieren. Die Möglichkeit, einzelne Komponenten und Module des Systems frei auf den verfügbaren Serversystemen zu verteilen, ist bei klassischen Groupware-Umgebungen nicht vorgesehen. Allerdings kann hier einer wachsenden Anzahl von Benutzern durch zusätzliche, entsprechend vernetzte Hardware begegnet werden. Dieser Vorgang wird auch als *Clustering* bezeichnet.

4.6.4 Benutzeroberfläche und Mailsystem

Die Gestaltung der Oberfläche von kollaborativen Applikationen ist bei der Verwendung des KollApps Modells sehr flexibel und durch keinerlei Vorgaben eingeschränkt. Allerdings müssen diese Elemente der Benutzeroberfläche vollständig ausprogrammiert werden. Diesem Mehraufwand können entsprechende Erweiterungen wie beispielsweise die JavaServer Faces Technologie entgegengesetzt werden. Allerdings wird durch die Verwendung vorgegebener UI-Komponenten die zuvor betonte Flexibilität stark eingeschränkt. Ein ähnlicher Effekt ist auch bei der Softwareentwicklung in klassischen Groupware-Umgebungen zu beobachten, da auch hierbei vorgegebene Komponenten kombiniert werden. Die integrierten Mailsysteme klassischer Groupware-Umgebungen bieten den Entwicklern zahlreiche

[136] Vgl. Erdmann 2003b S. 43.

Schnittstellen und Zugriffsmöglichkeiten, die innerhalb eines Entwicklungsprozesses des KollApps Modells erst aufwendig entworfen und implementiert werden müssten.

4.6.5 Datenstruktur

Die Datenstruktur typischer Groupware-Umgebungen speichert die Daten in so genannten Message-Objekten (siehe Abschnitt 2.2.1.2). Diese dokumentenorientierte Art der Datenhaltung bietet ein vergleichsweise hohes Maß an Flexibilität bei der Programmierung entsprechender Anwendungen. Während bei der vornehmlich relationalen Datenspeicherung der Anwendungen, die mithilfe von KollApps entworfen wurden, sehr aufwendige Entwurfs- und Optimierungsschritte zur Erstellung des Datenbankschemas notwendig sind, können die Datenbanken klassischer Groupware-Umgebungen unmittelbar und sehr flexibel mit Inhalten gefüllt werden.

Dieser Komfort der Groupware-Plattformen hat gegenüber der relationalen Datenspeicherung allerdings den Nachteil, dass bei Veränderungen der Feldinhalte ein hoher Wartungs- und Migrationsaufwand generiert wird. Relational gespeicherte Datensätze hingegen vollziehen diese Änderungen ganzheitlich und in einem Schritt.

Ein entscheidender Vorteil der klassischen Groupware-Umgebungen ist die Möglichkeit zur Replikation (siehe Abschnitt 2.2.1.2) der Datenbestände. Ein solcher Mechanismus zum Datenabgleich und die Möglichkeit, auch mit sämtlichen Daten zu arbeiten, wenn keine Datenverbindung zum Server besteht, lässt sich mit Systemen, die dem KollApps Modell folgen, nur sehr schwer beziehungsweise mit einem erheblichen Aufwand von Zeit und weiteren Ressourcen realisieren.

5 Ausblick

Das Konzept eines Vorgehensmodells für die Entwicklung kollaborativer Applikationen wurde in Kapitel 4 angewandt. Obwohl der realisierte Prototyp in seinen Eigenschaften und in seinem Funktionsumfang bereits eine vollständige Applikation darstellt und somit über einen prototypischen Charakter hinausgeht, muss sich das Konzept insbesondere in dem Umfeld größerer Entwicklungsprojekte unter der Beteiligung eines vollständigen Projektteams bewähren. Die Durchführung von Fallstudien und deren Auswertung könnte hierbei den Erfolg des Vorgehensmodells belegen und zu der Gewinnung neuer Ansatzpunkte zur weiteren Optimierung beitragen.

Eine weitere Möglichkeit zur Erweiterung des Konzepts bietet dessen Umsetzung in Form einer Software beziehungsweise eines Softwarepakets. Dies könnte die Projektmitarbeiter über den gesamten Verlauf des Entwicklungsprozesses hinweg umfassend sowohl inhaltlich als auch methodisch unterstützen. Insbesondere im Bereich der Organisation der zahlreichen untergeordneten Prozesse und Iterationsschritte sowie im Bereich des Versionsmanagements könnte diese Applikation das Projektteam anleiten. Denkbar wäre ebenfalls eine Bereitstellung von entsprechenden Werkzeugen für die Erstellung der verschiedenen Typen von Diagrammen, die vor allem der Phase der Ausarbeitung als Grundlage dienen.

Eine zusätzliche Erweiterung des erarbeiteten Konzepts könnte die Ausweitung des in Kapitel 3 konzipierten Vorgehensmodells auf die Entwicklung beliebiger Applikationen vorsehen. Hierbei müsste die Fokussierung des vorliegenden Modells auf die speziellen Eigenschaften der kollaborativen Applikationen und der sich daraus ergebenden Anforderungen an ein unterstützendes Vorgehensmodell aufgelöst werden. Auf diese Weise ließe sich ein allgemeingültiges Vorgehensmodell zur Durchführung von Softwareprojekten realisieren.

Zusätzlich bietet sich auch die Erarbeitung einer besonders vorteilhaften Vorgehensweise zur Einführung des beschriebenen Vorgehensmodells in Projektteams oder ganzen Softwarehäusern an. Mithilfe einer optimierten Einführungsstrategie ließe sich das Modell leichter integrieren und der Aufwand für dessen Einführung weiter senken.

6 Zusammenfassung

Die vorliegende Ausarbeitung zeigt, in welcher Form die Entwicklung von kollaborativen Applikationen in ihren kennzeichnenden Eigenschaften auf der Grundlage eines generischen Vorgehensmodells optimal unterstützt werden kann. Ein Vorgehensmodell liefert dabei einen Leitfaden zur Durchführung von Softwareprojekten. Mithilfe umfangreicher Beschreibungen der auszuführenden Tätigkeiten und Vorgehensweisen liefert das Modell über den gesamten Entwicklungsprozess hinweg sowohl inhaltliche als auch methodische Hilfestellungen für die Mitarbeiter des ausführenden Projektteams.

Die ausführliche Betrachtung der besonderen Merkmale von Applikationen, die den Mitarbeitern Unterstützung im Sinne der Kommunikation, Kooperation und Koordination zukommen lassen, verdeutlicht die Charakteristika dieser Gruppe von Anwendungen. Auf diesen Eigenschaften aufbauend sind spezielle Anforderungen an eine Vorgehensbeschreibung für die Entwicklung dieser Applikationen herausgearbeitet worden. Die Anforderungen dienen als Grundlage für die umfassende Diskussion von bereits existierenden Vorgehensmodellen zur Softwareentwicklung. Insbesondere die Projektphasen *Beginn*, *Ausarbeitung* und *Übergang* des Vorgehensmodells Unified Process und die *iterative Phase* der Implementierung des Modells Extreme Programming weisen in verschiedenen Bereichen Übereinstimmungen mit den gewünschten Eigenschaften auf. Eine Synthese der besonders zutreffenden und positiven Aspekte der beiden Modelle bildet so die Basis des neuen Vorgehensmodells zur Realisierung kollaborativer Applikationen.

Die prototypische Anwendung des erarbeiteten Konzeptes bei der Erstellung einer typischen kollaborativen Applikation gibt einen beispielhaften Einblick in die grundlegenden Prozesse und die vorgesehenen Tätigkeiten zur Implementierung der Beispielsapplikation auf der Basis der J2EE-Technologie.

Abschließend wird in einer Gegenüberstellung gezeigt, dass die Tätigkeiten, die durch das neue Vorgehensmodell definiert sind, gegenüber der Softwareentwicklung auf einer klassischen Groupware-Plattform in vielen Aspekten eine Abwägung der Faktoren der Flexibilität und dem gleichzeitigen Mehraufwand für die Programmierung gegenüber den Faktoren des Programmierkomforts und den damit

oft eingeschränkten Möglichkeiten der Programm- und Funktionsgestaltung darstellen.

Der zunehmende Einsatz kollaborativer Applikationen beziehungsweise die wachsenden Anforderungen an bereits existierende Applikationen lassen die Erweiterungen, die im Abschnitt des Ausblicks beschrieben sind, als realistische und notwendige Szenarien erscheinen. Diesen anknüpfenden Projekten kann das Konzept des Vorgehensmodells, das innerhalb dieser Arbeit erstellt wurde, als Grundlage dienen.

7 Literaturverzeichnis

Armstrong 2004:

Armstrong, Eric et al.: The J2EE 1.4 Tutorial, 2004; Aus: http://java.sun.com/j2ee/1.4/docs/tutorial/doc/J2EETutorial.pdf am 10.12.2004

Austin 2000:

Austin, Calvin; Pawlan, Monica: Advanced Programming for the Java 2 Platform. Addison-Wesley, Boston usw. 2000

Balzert 2001:

Lehrbuch der Software-Technik - Software-Entwicklung. 2. Auflage, Spektrum, Heidelberg usw. 2001

Beck 2000:

Beck, Kent: Extreme Programming – Das Manifest. Addison-Wesley, München usw. 2000

Boehm 1988:

Boehm Barry W.: A Spiral Model of Software Development and Enhancement. In: IEEE Computer. Nr. 5, Washington DC usw. 1988, S. 61-72

Brockhaus 2003:

Barnert, Silvia et al.: Der Brockhaus Computer und Informationstechnologie. Brockhaus GmbH, Leipzig, Mannheim 2003

Budde 1992:

Budde, R.; Kautz, K.; Kuhlenkamp, K.; Züllighoven, H.: Prototyping – An Approach to Evolutionary System Development. Springer-Verlag, Berlin usw. 1992

Bunse 2002:

Bunse, Christian; Von Knethen, Antje: Vorgehensmodelle kompakt. Spektrum, Heidelberg 2002

Carew 2003:

Carew, David; Desai, Sandeep: Keeping critters out of your code: How to use WebSphere and JUnit to prevent programming bugs, 2003; Aus: http://www-128.ibm.com/developerworks/edu/i-dw-wes-junit-i.html?S_TACT=105AGX02 am 15.02.2005

Coleman 1994:

Coleman, Derek; Arnold, Patrick; Bodoff, Stephanie; Dollin, Chris; Gilchrist, Helena; Hayes, Fiona; Jeremaes, Paul: Object-Oriented Development – The Fusion Method. Prentice Hall, New Jersey usw. 1994

Coleman 1997:

Coleman, David; Hyman Kutner, Abby: Collaborating on the Internet and Intranets. In: Groupware - Collaborative Strategies for Corporate LANs and Intranets, Hrsg.: Coleman, David, Prentice Hall PTR, Upper Saddle River 1997, S. 39 - 55

Crawford 2003:

Crawford, William; Kaplan, Jonathan: J2EE Design Pattern. O'Reilly&Associates, Sebastopol 2003

D'Souza 1999:

D'Souza, Desmond Francis; Wills, Alan Cameron: Objects, Components, and Frameworks with UML – The Catalysis Approach. Addison-Wesley, Massachusetts 1999

Dewan 1997:

Prasun, Dewan: Distributed Collaboration, 2004; Aus: http://www.cs.unc.edu/~dewan/290/f04/notes/intro.PDF am 22.11.2004

Duden 1994:

Drosdowski, Günther (Hrgs.): Duden, Das große Fremdwörterbuch – Herkunft und Bedeutung der Fremdwörter. Dudenverlag, Mannheim usw. 1994

Eckel 1998:

Eckel, Bruce: Thinking in Java – The Definite Introduction To Object-Oriented Programming in the Language of the World Wide Web. Prentice Hall, Upper Saddle River 1998

Ehlers 1997:

Ehlers, Peter: Integriertes Projekt- und Prozessmanagement auf Basis innovativer Informations- und Kommunikationstechnologien: Das GroupProject-System - Referenzrahmen, Architekturen, Konzept, Systemdesign und empirische Einsatzerfahrungen eines verteilten, prozessorientierten Projektinformationssystems. Shaker, Aachen 1997

Erdmann 2003a:

Erdmann, Ingo: JavaServer Faces (JSF) - Rapid Application Development mit WebSphere Studio 5.1.x, 2003; Aus: http://pbfb5www.uni-paderborn.de/www/WI/WI2/wi2_lit.nsf/L5All/24202fcb5cc53604c1256e90007c0848?OpenDocument&Login&TableRow=3.0#3. am 22.11.2004

Erdmann 2003b:

Erdmann, Ingo: Dick und … dünn – Fat- oder Thin-Client, wem gehört die Zukunft? In: Groupware Magazin – Expertenwissen für IT-Entscheider. 03/2003, H&T Verlagsgesellschaft, München 2003, S. 42-44

Fields 2001:

Fields, Duane K.; Kolb, Mark A.: JavaServer Pages – Dynamische Webservices entwickeln. Addison-Wesley, München usw. 2001

Fraunhofer 2001:

o. V.: KobrA-Method - Frequently Asked Questions, 2001 Aus: http://www.iese.fhg.de/Kobra_Method/Faq/ am 22.01.2005

Haas 2002:

Haas, Roland; Schreiner, Ulrich: Java-Technologien für Unternehmensanwendungen – Konzepte, Methode und Anwendungen der Java 2 Enterprise Plattform. Carl Hanser, München 2002

Haiges 2003:

Haiges, Sven: JavaServer Faces FAQ, 2003; Aus: http://javaserverfaces.flavor.de/faq.html am 10.12.2004

Heise 2004:

Meyer, Angela: Das Casino-Prinzip – Warum so viele IT-Großprojekte scheitern. In: c't - magazin für computer technik, 23/2004, S. 218 - 223

Hesse 1992:

Hesse, W.; Merbeth, G.; Frölich, R.: Software-Entwicklung – Vorgehensmodelle, Projektführung, Produktverwaltung. R. Oldenbourg, München 1992

Horn 1993:

Horn, Erika; Schubert, Wolfgang: Objektorientierte Software-Konstruktion – Grundlagen – Modelle – Methoden – Beispiele. Carl Hanser Verlag, München 1993

Howes 1997:

Howes, Timothy A.; Smith, Mark C.: LDAP – Programming Directory-Enabled Applications with Lightweight Directory Access Protocol. Macmillan Technical Publishing, Indianapolis 1997

IBM 2004a:

o.V.: About IBM, 2004; Aus: http://www.ibm.com/ibm/us/ am 02.12.2004

IBM 2004b:

o.V.: Der offizielle IBM Business Partner Katalog 2005. 13. Auflage, WIN-Verlag GmbH & Co. KG, Vaterstetten 2004

IBM 2004c:

o.V.: Servlet and JSP Development with IBM Websphere Studio 5.1.1. IBM Software Group, Cambridge 2004

Jacobson 1992:

Jacobson, Ivar; Christerson, Magnus; Jonsson, Patrik; Övergaard, Gunnar: Object-Oriented Software Engineering – A Use Case Driven Approach, Addison-Wesley, München usw. 1992

Jacobson 1999:

Jacobson, Ivar; Booch, Grady; Rumbaugh, James: The Unified Software Development Process – The complete guide to the Unified Process from the original designers. Addison-Wesley, Massachusetts usw. 1999

Kitain o.J.:

Kitain, Roger; Visvanathan, Jayashri: Guidelines for Designing Reusable Custom Components Using JavaServer Faces Technology, o. J.; Aus: http://java.sun.com/j2ee/javaserverfaces/customcomponents.html am 13.12.2004

Kneuper 1998:

Kneuper, Ralf; Müller-Luschat, Günther; Oberweis, Andreas: Vorgehensmodelle für die betriebliche Anwendungsentwicklung. Teubner, Leipzig 1998

Koslowski 1988:

Koslowski, Knut: Unterstützung von partizipativer Systementwicklung durch Methoden des Software Engineering. Westdeutscher Verlag GmbH, Opladen 1988

Lemay 1999:

Lemay, Laura; Cadenhead, Rogers: Java 2 in 21 Tagen. Markt und Technik, München 1999

Liu o. J.:

Liu, Jeffrey; Wong, Helen: Understanding JavaServer Faces - Assemble your Web applications effectively, o. J.; Aus: http://sys-con.com/story/?storyid=46159 am 13.12.2004

Lotus 1995:

o.V.: Groupware - Communication, Collaboration, Coordination, 1995; Aus: http://pbfb5www.uni-paderborn.de/www/WI/WI2/wi2_lit.nsf/a9d9ba5440dfda62c1256bc900519bf1/5098c20fcf549d15412564ca00333bc2/$FILE/Groupwar.pdf am 23.11.2004

Lotus 2003:

o.V.: The History of Notes and Domino, 2003; Aus: http://www-10.lotus.com/ldd/today.nsf/8a6d147cf55a7fd385256658007aacf1/bc684f3a96b4efd185256b9c0064ae9c?OpenDocument#Release%201.0%3A%20A%20star%20is%20born am 23.11.2004

Lotus Development 1999:

o.V.: Domino Designer Fundamentals. Lotus Development Corporation, o.O. 1999

Lotus Development 2004a:

o.V.: Lotus Domino Designer 6.5 Help, 2004; Aus: http://www-12.lotus.com/ldd/doc/domino_notes/6.5/help65_designer.nsf/Main?OpenFrameSet am 28.11.2004

Lotus Development 2004b:

o.V.: Lotus Notes 6.5 Help, 2004; Aus: http://www-12.lotus.com/ldd/doc/domino_notes/6.5/help65_client.nsf am 30.11.2004

Nastansky 1998:

Nastansky, Ludwig: Message-Objekte und Team-Kommunikation – Systembausteine für die Unternehmensführung in neuen Organisationsformen. In: Unternehmensführung und Kapitalmarkt – Festschrift zum 65. Geburtstag von Prof. Dr. Dr. h.c. Herbert Hax. Hrsg.: Laux, H.; Franke, G., Springer Verlag, Berlin 1998

Nastansky 2000:

Nastansky, Ludwig; Bruse, Thomas; Haberstock, Philipp; Huth Carsten; Smolnik, Stefan: Büroinformations- und Kommunikationssysteme: Groupware, Workflow-Management, Organisationsmodellierung und Messaging-Systeme. In: Bausteine der Wirtschaftsinformatik – Grundlagen, Anwendungen, PC-Praxis. Hrsg.: Fischer, Joachim et al., 2. Auflage, Erich Schmidt Verlag, Berlin 2000, S. 235 bis S. 323

Nastansky 2005:

Nastansky, Ludwig: K-Pool: A Process-driven Knowledge Management System for Contextual Collaboration Spanning Intranet to Internet. In: Accepted IEEE/WIC WI-2003, Halifax 2005

Oestereich 2001:

Oestereich, Bernd: Objektorientierte Softwareentwicklung – Analyse und Design mit der Unified Modeling Language. 5. Auflage, Oldenbourg, München, Wien 2001

Papows 1997:

Papows, Jeff: Deploying Second-Generation Intranets with Lotus Notes. In: Groupware - Collaborative Strategies for Corporate LANs and Intranets, Hrsg.: Coleman, David, Prentice Hall PTR, Upper Saddle River 1997

Pomberger 1996:

Pomberger, Gustav; Blaschek, Günther: Software-Engineering – Prototyping und objektorientierte Software-Entwicklung. 2. Auflage, Carl Hanser Verlag, München 1996

Raasch 1992:

Raasch, Jörg: Systementwicklung mit strukturierten Methoden – Ein Leitfaden für Praxis und Studium. 2. Auflage, Carl Hanser Verlag, München 1992

Riemann 2001:

Riemann, Walter O.: Systemplanung / Vorgehen. In: Wirtschaftsinformatik – Anwendungsorientierte Einführung. Hrsg.: Riemann, Walter O., 3. Auflage, Oldenbourg Wissenschaftsverlag GmbH, München usw. 2001, S.328 bis 350

Rothhardt 1987:

Rothhardt, Günter: Praxis der Software Entwicklung. Alfred Hüthig, Heidelberg 1987

Rumbaugh 1993:

Rumbaugh, James; Blaha, Michael; Premerlani, William; Eddy, Frederick; Lorensen, William: Objektorientiertes Modellieren und Entwerfen. Carl Hanser, München, Wien 1993

Schäffer 2002:

Schäffer, Stefan; Schilder Walter: Enterprise Java mit IBM WebSphere – J2EE-Applikationen effizient entwickeln. Addison-Wesley, München usw. 2002

Scheibl 1989:

Scheibl, Hans-Jürgen: Kommerzielle Software-Entwicklung: Systems-Engineering/ Software-Engineering im Überblick und in praktischen Anwendungen. Expert, Ehningen bei Böblingen 1989

Schryen 2001:

Schryen, Guido: Komponentenorientierte Softwareentwicklung in Softwareunternehmen: Konzeption eines Vorgehensmodells zur Einführung und Etablierung. Deutscher Universitäts-Verlag GmbH, Wiesbaden 2001

Sneed 1986:

Sneed, Harry M.: Software Entwicklungsmethodik. 5. Auflage, Rudolf Müller, Köln 1986

Steinmetz 2001:

El Saddik, Abdulmotaleb; Steinmetz, Ralf: Ein Java-basiertes Werkzeug für transparente Kollaboration über das Internet, 2001; Aus: http://www.kom.e-technik.tu-darmstadt.de/publications/abstracts/ES01-1.html am 22.11.2004

Suhl 2002:

Knechtel, Thomas; Suhl, Leena; Toschläger, Markus: Skript zur Veranstaltung IT-Consulting – Management von IT-Projekten. 8. Auflage, Universität Paderborn, Paderborn 2002

Turau 2001:

Turau, Volker; Pfeiffer, Ronald: Java Server Pages. 2. Auflage, dpunkt, Heidelberg 2001

Wagner 1995:

Wagner, M.: Groupware und neues Management: Einsatz geeigneter Softwaresysteme für flexiblere Organisationen. Vieweg, Braunschweig usw. 1995

Wagner 2001:

Wagner, Lothar; Sauer, Petra: Datenbanksysteme. In: Wirtschaftsinformatik – Anwendungsorientierte Einführung. Hrsg.: Riemann, Walter O., 3. Auflage, Oldenbourg Wissenschaftsverlag GmbH, München usw. 2001, S.194 bis 223

Wahli 2004:

Wahli, Ueli et al.: WebSphere Studio 5.1.2 JavaServer Faces and Service Data Objects, 2004; Aus: http://www.redbooks.ibm.com/redbooks/pdfs/sg246361.pdf am 10.12.2004

White 1994:

White, Insult: Using the Booch Method: A Rational Approach, Benjamin/Cummings Publishing Company, Redwood City usw. 1994

Whitehead 1996:

Whitehead, Roger: Lotus Notes – An Introduction. In: Transforming Organisations Through Groupware. Hrsg.: Lloyd, Peter; Whitehead, Roger. Springer Verlag, Berlin usw. 1996

Whitehead 2002:

Whitehead, Katharine: Component-based Development – Principles and Planning for Business Systems. Pearson Education Ltd, London 2002

Wille 2001:

Wille, Stefan: Java Server Pages – Der Weg zum Profi – Flexible Lernmodule –Arbeitsbuch und Referenz. Addison-Wesley, München usw. 2001

Wilson 2004:

Wilson, Doug: New opportunities for Lotus Domino developers - plus a new set of tools, 2004; Aus: http://www-306.ibm.com/software/swnews/swnews.nsf/n/jmae65krek?OpenDocument&Site=lotus am 30.11.2004

Winkelmann 2001a:

Winkelmann, Sabine: Rechnerverbundsysteme und Kommunikation. In: Wirtschaftsinformatik – Anwendungsorientierte Einführung. Hrsg.: Riemann, Walter O., 3. Auflage, Oldenbourg Wissenschaftsverlag GmbH, München usw. 2001, S. 103 bis S. 157

Winkelmann 2001b:

Winkelmann, Sabine: Einführung in die Programmierung. In: Wirtschaftsinformatik – Anwendungsorientierte Einführung. Hrsg.: Riemann, Walter O., 3. Auflage, Oldenbourg Wissenschaftsverlag GmbH, München usw. 2001, S. 249 bis S. 276

Anhang A Herstellerverzeichnis

Folgende Produkte der angegebenen Hersteller wurden für die prototypische Realisierung des Konzepts eingesetzt.

Hersteller	Softwareprodukt
IBM Corporation (http://www.ibm.com)	- DB2 Universal Database V8.1 - WebSphere Application Server V5.1.1 - WebSphere Studio Application Developer 5.1.2 - Lotus Notes 6 - Lotus Domino 6
Sun Microsystems Inc. (http://www.sun.com)	- Java Software Development Kit 1.4 - JavaServer Faces 1.0

Tabelle 3 - Herstellerverzeichnis

Anhang B Aufbau und Verwendung der begleitenden CD-ROM

Die CD-ROM, die dieser Ausarbeitung beiliegt, enthält verschiedene Dateiressourcen sowie eine Installationsanleitung für den erarbeiteten Prototyp. In dem folgenden Text werden dieser Aufbau und die Verwendung der CD-ROM erläutert.

Wird die CD in das CD-ROM Laufwerk eingelegt, so wird automatisch die HTML-Oberfläche zu den hinterlegten Inhalten geöffnet. Im linken Bereich ist ein Navigator erkennbar, der aus zwei Kategorien besteht. Die erste Kategorie der *Ausarbeitung* enthält Verweise auf die vorliegende Ausarbeitung zum einen im Format des Softwareprodukts Microsoft Word, zum anderen im Format *Portable Document Format* (PDF) der Firma Adobe. Die zweite Kategorie weist neben dem Enterprise Archive des K-Pool Everyplace auch eine vollständige Sicherheitskopie der verwendeten IBM DB2 Datenbank auf. Die Installationsanleitung erläutert welche verschiedenen Schritte und Einstellungen nötig sind, damit der K-Pool Everyplace erfolgreich installiert werden kann. Bei den beiden letzten Einträgen der Kategorie handelt es sich um Verweise auf die Programmressourcen des K-Pool Everyplace und der TreeStructure Komponente als WSAD-Projekte.

Entsprechend der Struktur des Navigators der HTML-Oberfläche gliedert sich der Inhalt der CD-ROM in ein Verzeichnis *Ausarbeitung*, das diese Arbeit in unterschiedlichen Dateiformaten vorhält, und in einen Ordner *Prototyp*, in dem alle Dateiressourcen bezüglich des praktischen Bestandteils dieser Arbeit gespeichert sind. In dem Ordner *GIF* sind alle Grafiken, die von der HTML-Oberfläche benutzt werden, hinterlegt. Mit einem Klick auf den entsprechenden Eintrag des Navigators beziehungsweise mit der Funktion *Speichern unter…*des Kontext-Menüs lassen sich alle Ressourcen lokal speichern.

Die HTML-Oberfläche der begleitenden CD-ROM lässt sich auch mithilfe der Datei *index.html* manuell starten.