Möglichkeiten der Software-Wiederverwendung durch komponentenbasierte Anwendungsentwicklung in einem Versicherungsunternehmen

Diplomarbeit

vorgelegt am: Institut für Informatik
Universität Leipzig
im Dezember 1997

bearbeitet von: André Weede
Innenring 13
04509 Delitzsch OT Poßdorf

Erstprüfer: Prof. Dr. habil. E. Rahm
Abteilung Datenbanken, Institut für Informatik, Universität Leipzig

Zweitprüfer: Dr. Frank Noack
R+V Allgemeine Versicherung AG

Inhaltsverzeichnis

Abbildungsverzeichnis

Kapitel 1: Grundlagen der Wiederverwendung

Kapitel 2: Komponentenbasierte Anwendungsentwicklung

Kapitel 3: Wiederverwendung in der R+V-Versicherung

Tabellenverzeichnis

Kapitel 2: Komponentenbasierte Anwendungsentwicklung

Kapitel 3: Wiederverwendung in der R+V-Versicherung

0. Motivation

Vergleicht man die Software-Entwicklung von heute mit Produktionsmethoden in anderen Wirtschaftszweigen, so stellt man fest, daß Software-Erstellung eher einem Handwerk gleicht denn einer Industrie [Po93]. Methoden, wie Mehrfachverwendung von Teilen und Halbfabrikaten, Automatisierung von Abläufen und Fließbandproduktion, die zum Beispiel in der Automobilherstellung nicht mehr wegzudenken sind, haben sich bei der Software-Herstellung noch nicht durchgesetzt.
Das Problem ist seit längerer Zeit bekannt, wie folgende Aussage bestätigt:
"Was momentan geschieht, ist, wie ich fürchte, daß wir uns selbst und gegenseitig sagen, daß die Software-Entwicklungstechnik entscheidend verbessert werden sollte, weil sie in einer Krise steckt. Aber es existieren einige Randbedingungen, die anscheinend vorher überwunden werden müssen. Ich werde sie für Sie auflisten:

1. Wir können unsere Denkgewohnheiten nicht ändern.
2. Wir können unsere Programmierwerkzeuge nicht ändern.
3. Wir können unsere Hardware nicht ändern.
4. Wir können unsere Aufgaben nicht ändern.
5. Wir können unsere organisatorischen Strukturen, in denen die Arbeit getan wird, nicht ändern."[1]

(Dijkstra auf der "NATO-conference on SE-techniques '69": aus [Bö89])

Obwohl diese Äußerung aus dem Jahre 1969 stammt, besitzt sie heute noch zu großen Teilen Gültigkeit. Die Software-Entwicklung steckt in vielen Bereichen in einer Krise, diese ist gekennzeichnet durch die Unfähigkeit:

- Anwendungen dem Benutzer zum gewünschten Zeitpunkt zur Verfügung zu stellen,
- geforderte Änderungen in vertretbaren Zeiträumen durchzuführen,

---

[1] "What is actually happening, I am afraid, is that we tell each other and ourselves that software engineering techniques should be improved considererably, because there is a crisis. But there are a few boundary conditions which apparently have to be satisfied. I will list them for you:

1. We may not change our thinking habits.
2. We may not change our programming tools.
3. We may not change our hardware.
4. We may not change our tasks.
5. We may not change the organizational set-up in which the work has to be done."

- Systeme auszuliefern, die den tatsächlichen Anforderungen entsprechen,
- Anforderungen der Benutzer fachlich richtig zu spezifizieren,
- Kostengrenzen für Entwicklung und Wartung einzuhalten [Du93].

Immer neue Wege aus der Krise werden propagiert, dennoch bestehen die Probleme bis heute. Es ist unwahrscheinlich, daß durch eine neue Methode oder ein neues Werkzeug die Software-Entwicklung revolutioniert und die Software-Krise überwunden wird. Trotzdem existieren eine Reihe von Ansätzen, mit deren Hilfe der Software-Entwicklungsprozeß verbessert und die Software-Produktion effektiver gestaltet werden können. Analog zur industriellen Fertigung erhofft man sich insbesondere von einer breiten Wiederverwendung eine erhebliche Steigerung von Produktivität und Qualität.
Vor allem das objektorientierte Vorgehen versprach ein hohes Maß an Wiederverwendbarkeit. Erste Untersuchungen der objektorientierten Software-Entwicklung im Versicherungsumfeld haben jedoch gezeigt, daß die Kosten der objektorientierten Entwicklung deren Nutzen noch weit übersteigen. Vor allem der Mangel an betriebswirtschaftlich nutzbaren Werkzeugen und Umgebungen verhindert die Durchsetzung der Objektorientierung. Besonders deutlich wird dies im Hinblick auf Datenbanken. So existieren momentan keine Datenbanksysteme, die eine effektive Verwaltung der großen in einem Versicherungsunternehmen anfallenden Datenmengen gewährleisten. Vielmehr muß man davon ausgehen, daß in absehbarer Zeit in Versicherungsunternehmen die relationale Datenbankwelt dominierend bleiben wird.
Die Idee der komponentenbasierten Anwendungsentwicklung besteht darin, einen Teil der Möglichkeiten der Objektorientierung in die bestehende Software-Entwicklung einzubringen. Insbesondere der Kapselung und der Überwindung der Trennung der Daten und Funktionen kommen im Hinblick auf Wiederverwendbarkeit große Bedeutung zu.
Die Möglichkeiten der komponentenbasierten Anwendungsentwicklung im bestehenden Umfeld der R+V-Versicherung sollen in dieser Arbeit untersucht werden.
Ausgehend von den Erwartungen an Wiederverwendung sollen die verschiedenen Wiederverwendungsarten gegenübergestellt werden. Die Kenntnis der unterschiedlichen Möglichkeiten der Wiederverwendung stellt eine wichtige Grundlage für die unternehmensspezifische Entscheidung für die erfolgversprechendste Strategie dar.

Ein weiterer wichtiger Faktor für erfolgreiche Wiederverwendung von Software ist das Bewußtsein, daß eine Reihe von Hindernissen und Widerständen zu erwarten sind. Diese sollen in Abschnitt 1.3 aufgestellt und kurz charakterisiert werden.

Aufgrund der gemachten Erfahrungen in der R+V-Versicherung und der Struktur der Anwendungsentwicklung sowie der vorliegenden Werkzeuge steht die Wiederverwendung durch komponentenbasierte Anwendungsentwicklung im Mittelpunkt der vorliegenden Arbeit. Dieser Ansatz wird in Abschnitt 2 näher betrachtet. Nach der Darstellung der allgemeinen Prinzipien sollen die Anforderungen an wiederverwendbare Komponenten aufgestellt werden, weiterhin soll der Prozeß der Entwicklung, Speicherung und Wiederverwendung von Komponenten beschrieben werden. Daraus werden Schlußfolgerungen für das Life-Cycle-Modell gezogen.

Abschließend werden mögliche Wege zum Übergang zu einer Wiederverwendungskultur aufgezeigt.

Die in den ersten beiden Abschnitten gewonnenen Erkenntnisse werden in Abschnitt 3 benutzt, um die Möglichkeiten der Wiederverwendung am Beispiel der R+V-Versicherung zu untersuchen. Dazu wird die momentane Situation im Unternehmen analysiert. Dies schließt sowohl die Analyse der Software-Entwicklungsumgebung als auch des Vorgehensmodells ein. Ausgehend von den gewonnenen Erkenntnissen wird ein Weg zur Einführung von Wiederverwendung aufgezeigt.

Anhand einer prototypischen Implementation einer Anwendung werden Vorgehensweise und zu schaffende Voraussetzungen gezeigt. Abschließend werden die gewonnenen Erkenntnisse zusammengefaßt.

Verwiesen sei an dieser Stelle auf das Glossar im Anhang B. Dort werden insbesondere Begriffe aus dem untersuchten Umfeld der R+V-Versicherung definiert. Alle im Glossar erläuterten Begriffe erscheinen im Text unterstrichen.

1. Wiederverwendung

Auf der Suche nach verbesserten Methoden der Software-Entwicklung wurden eine Reihe von Software-Projekten und Anwendungen untersucht. Unterscheiden sich auch die genauen Zahlenwerte der Ergebnisse, so kann man doch den gemeinsamen Tenor aller Untersuchungen erkennen: es besteht ein hohes Potential an mehrfach nutzbaren Programmteilen.

Untersuchungen von T.C.Jones zum Beispiel zeigen, daß nur 15% des 1983 geschriebenen Codes neu waren [Bö89], das heißt, 85% waren zu einem früheren Zeitpunkt bereits in anderen Zusammenhängen programmiert worden.

Der hohe Anteil redundanten Codes legt die Idee nahe, diesen bereits bestehenden Code wiederzuverwenden. Man sollte jedoch die Wiederverwendung nicht auf Code-Module beschränken, wie die Definition der Wiederverwendung nach Biggerstaff/ Perlis verdeutlicht:

"**Software-Wiederverwendung ist die Wiederanwendung einer Reihe verschiedener Arten von Wissen über ein System auf ein anderes, ähnliches System, um die Aufwände zur Entwicklung und Wartung dieses anderen Systems zu reduzieren.**

Dieses wiederverwendete Wissen schließt Dinge, wie Domänenwissen, Entwicklungserfahrungen, Design-Entscheidungen, Architekturstrukturen, Anforderungen, Pläne, Code, Dokumentationen und so weiter ein."[1]

(Biggerstaff, Perlis 1989; aus [He93])

Insbesondere bei der Weiterentwicklung von Anwendungen ist das Potential der Wiederverwendung hoch. McClure schätzt, daß 40-60% des Gesamtcodes von einer Anwendung zur nächsten und 60% des Entwurfs aller kommerziellen Anwendungen wiederverwendbar sind, 75% der Programmfunktionen sind vielen Programmen gemeinsam und nur 15% des Programmcodes sind ausschließlich für eine bestimmte Applikation verwendbar [McC93]. Analog zu Biggerstaff/ Perlis schließt auch McClure in die wiederverwendbaren Teile des Entwicklungsprozesses Prototypen, Daten, Entwürfe für Programmarchitekturen und Datenstrukturen, Datenmodelle, Programm-Code-Fragmente,

[1] "Software reuse is the reapplication of a variety of kinds of knowledge about one system to another similiar system in order to reduce the effort of development and maintenance of that other system. This reused knowledge includes artifacts such as domain knowledge, development experience, design decisions, architectural structures, requirements, designs, code, documentation and so forth."

Softwarepakete und Lebenszyklusprozesse mit ein [McC93]. Die Verwendung von Ergebnissen der Analyse- und Entwurfsphase (siehe auch Phasenmodell in 2.2.5) stellt bei der Wiederverwendung einen besonders effizienten Ansatz dar [Ze95], da mit der Wiederverwendung von Elementen sehr früher Entwicklungsphasen auch deren Folgeprodukte in hohem Umfang wiederverwendet werden können.

"Die Wiederverwendung von Ergebnissen des Software-Entwicklungsprozesses ist heute anerkannterweise eine der effizientesten Möglichkeiten, Produktivität und Qualität zu steigern." [Ka96]

In den folgenden Abschnitten sollen nun die Grundlagen der Wiederverwendung dargestellt werden. Ausgehend von den Erwartungen und Zielen, die mit Wiederverwendung verbunden werden, sollen unterschiedliche Arten von Software-Wiederverwendung dargestellt werden. Eine Reihe von Hindernissen, die Wiederverwendung in der Software-Entwicklung entgegenstehen und eine breite Umsetzung behindern werden im dritten Abschnitt dieses Kapitels aufgezeigt.

Abschließend werden die Erkenntnisse zur Wiederverwendung aus diesem Kapitel zusammengefaßt.

1.1 Erwartungen und Ziele der Wiederverwendung

Der Einführung einer neuen Strategie in der Software-Erstellung sollte eine genaue Analyse der Erwartungen und Ziele, die mit der Umstellung verbunden werden, sowie die anschließende klare Definition des angestrebten Zielzustandes vorausgehen, um eine spätere Bewertung zu ermöglichen.

An die Einführung einer breiten Wiederverwendung werden verschiedene Erwartungen geknüpft, insbesondere die *Steigerung der Qualität* bei *reduzierten Kosten, Risiken und Zeitbedarf* [Ka96], aber auch mehr *Flexibilität* bei der Anwendungsgestaltung in Form von verbesserter Anpaßbarkeit der Anwendungen an neue Anforderungen [AF93].

1.1.1 Erwartungen und Ziele hinsichtlich der Qualitätsverbesserung:

Qualität von Software schließt die in Abbildung 1.1 dargestellten Aspekte ein:

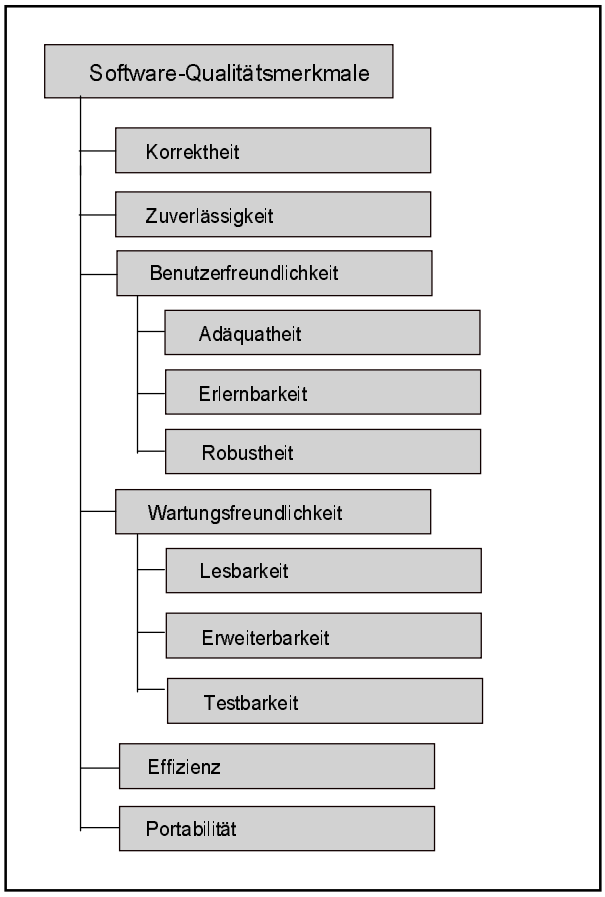

Abb. 1.1: Software-Qualitätsmerkmale (nach [PB93])

Von einer breiten Umsetzung von Wiederverwendung erwartet man insbesondere Verbesserungen im Hinblick auf Korrektheit (Erfüllung der spezifizierten Aufgabe unabhängig von der tatsächlichen Verwendung), Effizienz (bestmögliche Ausnutzung der verwendeten Ressourcen), Portabilität (Übertragbarkeit auf verschiedene Hardware- und Betriebssystemsplattformen) und Wartbarkeit.
Dabei stellt die Verbesserung der Wartbarkeit einen wesentlichen Aspekt der Verbesserung der Qualität dar. Wartbarkeit ist heute ein entscheidendes Kriterium für gute Software, Wartung beansprucht einen Großteil der Budgets von DV-Abteilungen, in den meisten Unternehmen liegt der Anteil bei etwa 50%, in großen Firmen sogar bei 70-80% des Gesamtetats. Die Ursachen dafür sind vielfältig. Zum einen sind Programme und Programmteile teilweise bis zu 20 Jahre alt, so daß sie im Laufe der Zeit häufigen Änderungen unterworfen waren. Diese erfolgten meist durch unterschiedliche Personen und wurden schlecht oder gar nicht dokumentiert. Die Folge sind schlecht lesbare Programme, oft ist unbekannt *was* das Programm *wie* tut. Die Pflege derartiger Programme erfordert hohe Aufwendungen, eine Ablösung ist schwierig aber unumgänglich (nach [You93]).
Wartung beinhaltet nach Swanson in [Sn91] die folgenden Aspekte (Anteil am Gesamtaufwand geschätzt):

- Korrektion	Beheben von Fehlern	ca. 24%
- Anpassungen	Änderungen der Umwelt	ca. 22%
- Erweiterung	funktionelle Ergänzung	ca. 40%
- Optimierung	Performance-Verbesserung	ca. 14%.

Die Wartungsaufwände steigen in immer stärkerem Maße. Da große Teile der Aufwände in die Wartung bestehender Software fließen, werden Neuentwicklungen unter starkem Kosten- und Zeitdruck durchgeführt, darunter leidet die Qualität. Die Konzentration auf den kurzfristigen Erfolg verhindert oft die Berücksichtigung der Wartbarkeit, es ist häufig sogar ein gegenteiliger Effekt zu beobachten. Nicht bewältigte Aufgaben werden in die Wartungsphase verlagert, somit beginnt Wartung bereits mit der Einführung neuer Software. Dieser Teufelskreis kann nur durch Verbesserung der Wartbarkeit neuer Programme durchbrochen werden [Sn91].

Dazu soll Wiederverwendung einen Beitrag leisten. *Anpaßbarkeit* und *Erweiterbarkeit* sollen verbessert werden, die *Fehlerzahl* soll verringert werden.

Ein Beispiel für gravierende Mängel in der Wartbarkeit trat vor einiger Zeit in Versicherungsunternehmen zutage. Die unerwartete Veränderung der Versicherungssteuer von 10% auf 15% stellte eine Reihe deutscher Versicherer vor große Probleme. Da die Versicherungssteuer über Jahre hinweg konstant geblieben war, rechneten einige Entwickler nicht mit einer Veränderung. So fanden sich in einer Vielzahl von Programmen Anweisungen der Form:

Zahlbetrag:=Versicherungsbetrag*1.1.

Das Auffinden aller zu ändernden Programmstellen und deren Änderung stellte einen erheblichen Aufwand dar. Die aktuellen Diskussionen um die Jahrtausendwende und die Währungsumstellung zeigen, daß es sich nicht um einzelne Probleme handelt, sondern daß der Berücksichtigung der Wartbarkeit bei der Software-Erstellung eine entscheidende Bedeutung zukommt.

Wiederverwendung ist ein erfolgversprechender Ansatz, um der zunehmenden Behinderung der Neuentwicklung entgegenzuwirken, indem zum einen Wartbarkeit bei der Software-Erstellung berücksichtigt wird und zum anderen durch den Einsatz bereits erstellter, qualitativ hochwertiger Komponenten die Gesamtkosten des Software-Lebenszyklus zusätzlich gemindert werden [BFB95].

Ein Umfrageergebnis zu den erwarteten Auswirkungen der Wiederverwendung auf die Qualität der Software findet sich in [Ni96]:

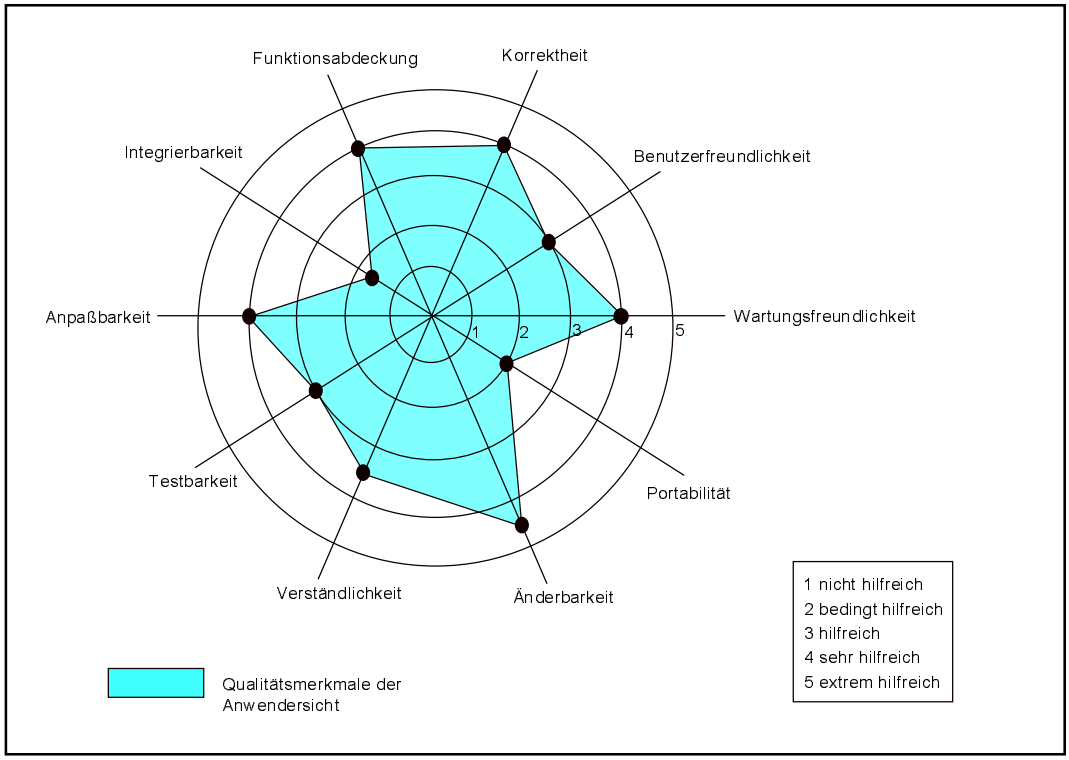

Abb. 1.2 Einfluß der Wiederverwendung auf die Qualität (nach [Ni96])

Aus Abbildung 1.2 können unterschiedliche Schlußfolgerungen gezogen werden.

Wartbarkeit wird als wichtiges Qualitätskriterium angesehen. Davon zeugen die hohen Bewertungen der Erwartungen an Wartungsfreundlichkeit, Anpaßbarkeit und Änderbarkeit. Die Trennung dieser drei Aspekte zeigt aber auch, daß oft unklar ist, was unter Wartung zu verstehen ist. Die Trennung soll die hohen Aufwände für Wartung in den Unternehmen verdecken, sie zeigt das fehlende Bewußtsein für die Ursachen hoher Wartungskosten.

Die geringen Erwartungen an die Portabilität zeigen, daß das Wissen über die Möglichkeiten der Wiederverwendung unzureichend ist (vgl. generative Wiederverwendung in Abschnitt 1.2.2.1).

Die schlechte Bewertung des Einflusses der Wiederverwendung auf Integrierbarkeit zeigt ebenfalls die unzureichenden Kenntnisse der Möglichkeiten von Wiederverwendung.

1.1.2 Reduzierung der Kosten, Risiken und des Zeitbedarfs

Von breiter Wiederverwendung erwartet man eine **termingerechtere Auslieferung** der Produkte [Ka96], da der Aufwand für Neuentwicklungen durch die Verwendung bestehender Ergebnisse abnimmt. Wenn wiederverwendete Bauteile bereits bewährt sind, sinkt die Gefahr von Fehlern, der Aufwand zur Einbindung ist im Vergleich zur Neuentwicklung gering.
Der Anteil nicht erfolgreich beendeter Projekte in der Software-Entwicklung ist hoch, die dadurch entstehenden Kosten sind immens. Das Risiko einer Entwicklung steigt, je weniger bekanntes Wissen aus vorhergehenden Projekten und Anwendungen genutzt wird. Das heißt, wird organisiert auf bestehende Erfahrungen und Kenntnisse zugegriffen, erhöhen sich die Erfolgsaussichten. Werden bereits getestete oder in anderen Anwendungen bewährte Programmteile wiederverwendet, erreicht man nicht nur Einsparungen wegen des geringeren Entwicklungsaufwandes, es verringert sich auch die Wahrscheinlichkeit von Programmfehlern und somit das Entwicklungsrisiko [Ka96].
Insbesondere bei Graphischen Oberflächen erreicht man durch Wiederverwendung von Anwendungsteilen den Effekt eines einheitlichen "Look and Feel", das heißt, der Benutzer erkennt aufgrund der Ähnlichkeit zu anderen oder früheren Anwendungen eher Funktionsweise und Bedienung neuer Anwendungen, es erhöht sich die Akzeptanz der neuen Anwendung, die nötigen Aufwände für Schulung und das Risiko einer Ablehnung sinken.

1.1.3 Weitere Ziele der Wiederverwendung

In vielen Fällen können durch **Einbindung kommerzieller Produkte**, wie z.B. Tabellenkalkulationen und Textverarbeitungsprogrammen, in Eigenentwicklungen zum einen Entwicklungsaufwand und -risiko gesenkt und zum anderen die Vertrautheit der Nutzer mit den eingebundenen Produkten ausgenutzt werden. Die mögliche Einbindung von "best-in-class"-Komponenten [1] [TIIP] führt zu qualitativ besseren Lösungen und zu verkürzten Entwicklungszeiträumen.
Die Wiederverwendung von Teilen bestehender Systeme ist zwingend erforderlich, eine vollständige Ablösung komplexer Anwendungen in einem Schritt stellt sich oft als nicht

[1] "best-in-class"-Komponente - beste Komponente ihrer Art

realisierbar dar. Eine Wiederverwendungsumgebung muß folglich Möglichkeiten der **Einbindung bestehender Systeme** (legacy-sytems) oder Systemteile bieten [Li96].
Ein wesentliches Ziel liegt in der **Erhaltung der Konkurrenzfähigkeit** [Ka96]. Wege zur Erreichung dieses Zieles sind eine schnellere Reaktion auf Veränderungen der Umwelt und die kostengünstige Produktion. Beides soll durch Wiederverwendung unterstützt werden.

1.1.4 Erfolgsaussichten

Den hohen Erwartungen entgegen stehen eine Vielzahl von nicht erfolgreich umgesetzten Wiederverwendungsprojekten. Mögliche Ursachen für das Scheitern werden in Abschnitt 1.3 aufgezeigt. Daß aber die genannten Erwartungen auch erfüllt werden können, zeigen Projekte hauptsächlich aus Japan und den USA. Dort hat sich Software-Wiederverwendung in einigen Unternehmen als sehr erfolgreich erwiesen. So berichtet NEC Tokio über deutliche Produktivitätssteigerungsraten und Qualitätsverbesserung [Ka96].
Bei FUJITSU konnten nach Einführung einer organisierten Wiederverwendung 70% der Projekte termingerecht beendet werden. Vor Einführung der neuen Methodik waren es lediglich 20% [Ka96].
Bei Toshiba werden seit 1977 wiederverwendbare Komponenten entwickelt. Die erzielten Wiederverwendungsraten liegen zwischen 32 und 48% [Bö89].
Insbesondere für die Systemlandschaft von Banken und Versicherungen werden der Wiederverwendung hohe Erfolgsaussichten vorausgesagt. So wird in [Bö89] von Wiederverwendungsraten von etwa 75% ausgegangen, dies ist in der Tatsache begründet, daß aufgrund gesetzlicher Vorgaben viele sehr ähnliche Systeme bestehen.
In [McC93] wird als Beispiel die Hartford Insurance Group angeführt, bei der durch Wiederverwendungsraten von 30-40% Kostenreduzierungen von 250 auf 25 Manntage je Monat für die Wartung der erstellten Systeme erzielt wurden.
Appelfeller berichtet von Untersuchungen, bei denen durch Wiederverwendung eine Erhöhung der Produktivität erzielt wurde [Ap96].

Trotz dieser positiven Erfahrungen besteht kein Grund zur Euphorie. Veröffentlichte Zahlen stammen häufig aus Pilotprojekten oder wurden in Umgebungen mit sehr guten

Erfolgsaussichten erzielt. Eine breite Anwendung der Wiederverwendung wird in den meisten Fällen keine so dramatischen Steigerungsraten erzielen, der Erfolg besteht in einer neuen Qualität der Software, die sich vor allem in den Phasen nach der Erstellung zeigen wird.

1.2 Einteilung der Arten von Wiederverwendung

Um Wiederverwendung erfolgreich umzusetzen, müssen zum einen unternehmensspezifische Gegebenheiten berücksichtigt werden, zum anderen ist die Kenntnis der verschiedenen möglichen Wiederverwendungsarten wichtig, um die erfolgversprechendste Strategie auszuwählen. Aus diesem Grunde sollen zunächst die verschiedenen Möglichkeiten der Wiederverwendung unterschieden werden.

Generell muß man zwischen Ad Hoc-Wiederverwendung und geplanter Wiederverwendung unterscheiden.

1.2.1 Ad Hoc-Wiederverwendung

Ad Hoc-Wiederverwendung findet in jedem Software-erstellenden Unternehmen statt. Das besondere Merkmal des Unternehmens ist, daß es keinerlei Maßnahmen getroffen hat, um die Wiederverwendung zu systematisieren [Re95].
Ad Hoc-Wiederverwendung ist zufallsgesteuert und unsystematisch . Die Wiederverwendung erfolgt individuell, informell, unkontrolliert, undokumentiert und unkoordiniert [Re95], es wird gelegentlich in Abhängigkeit von der Arbeitsweise der Projektmitarbeiter wiederverwendet. Der Erfolg der Ad Hoc-Wiederverwendung ist stark begrenzt, da die Elemente aufgabenspezifisch entwickelt werden [Pr96] und im allgemeinen aufwendig an neue Anforderungen angepaßt werden müssen. Damit wird die Ad Hoc-Wiederverwendung zu teuer, es kann zu einem kontraproduktiven Effekt kommen [Pr96]. Von der Ad Hoc-Wiederverwendung sind also weder große Einsparungen noch hohe Produktivitätssteigerungen zu erwarten [Pr96].
Beispiel:
Eine aus früheren Programmen vorhandene Funktion wird in ein neues Programm kopiert, da der Entwickler an deren Erstellung beteiligt war und erkennt, daß Ähnlichkeiten in den Anforderungen bestehen. Die der Funktion zugrundeliegenden Datenstrukturen müssen eingefügt bzw. angepaßt werden. Einige Anweisungen werden verändert, zusätzliche Funktionalitäten werden eingefügt. Konsistenzkriterien müssen erweitert werden. Nachfolgend muß die Funktion im neuen Kontext getestet werden. Über die

Wiederverwendung werden keinerlei Nachweise geführt. Infolgedessen wird der Verwender nicht über Veränderungen des Originals informiert.

Die mit Einführung von Wiederverwendung verknüpften Erwartungen können durch Ad Hoc-Wiederverwendung nicht erfüllt werden. Es sind weder Qualitätsverbesserungen noch erhöhte Produktivität zu erwarten.

1.2.2 Geplante Wiederverwendung

Aus den Mängeln der Ad Hoc-Wiederverwendung wurde die Notwendigkeit erkannt, Bauteile von vornherein wiederverwendbar zu entwickeln. Dies bedingt, daß die Wiederverwendung im Unternehmen geplant und organisiert umgesetzt wird. Dazu werden die Ausführung der Wiederverwendungsaktivitäten formalisiert sowie Handbücher und Richtlinien entwickelt.
Voraussetzung ist eine generelle, gemeinsame Struktur im Unternehmen. Geplante Wiederverwendung wird ständig anhand von Metriken kontrolliert und verbessert. Geplante Wiederverwendung erfordert einen höheren Aufwand, die zu erwarteten Erfolge liegen jedoch weit über denen der Ad Hoc-Wiederverwendung. [Pr96]
Aufgrund der untergeordneten Bedeutung der Ad Hoc-Wiederverwendung soll im weiteren nur noch strukturierte Wiederverwendung betrachtet werden. Zur Verdeutlichung der Eigenschaften der einzelnen Wiederverwendungsarten soll im folgenden der Vergleich mit der Hardware-Herstellung herangezogen werden.

Die nachfolgend aufgeführten Arten der Wiederverwendung schließen sich nicht aus, vielmehr sind erst im Zusammenspiel der unterschiedlichen Möglichkeiten die erfolgversprechendsten Strategien zu sehen.

Grundlegend kann man zwei Arten von geplanter Wiederverwendung unterscheiden.

1.2.2.1 Generative Wiederverwendung

Die Grundidee der generativen Wiederverwendung besteht in der Formalisierung bzw. Automatisierung sich wiederholender Problemlösungen. Nach dem Grad der Automatisierung unterscheidet man zwischen Design Patterns und Reusable Patterns.

a) Design Patterns (Entwurfsmuster)

Entwurfsmuster sind keine Software-Bausteine, aus denen durch bloße Montage Anwendungen gebildet werden können, Entwurfsmuster sind vielmehr Lösungsvorschriften, die festlegen, wie typische, sich wiederholende, komplexe Entwurfsproblem zu lösen sind [SO97]. Entwurfsmuster sollen Erfahrungen beim Entwurf von Software festhalten und damit dazu beitragen, Software schneller richtig zu entwerfen [Ga96]. Design Patterns bestehen aus der Beschreibung eines Entwurfsproblems und des Kontextes, in dem die Lösung gültig ist [Li96]. Weiterhin sollten Informationen zu Konsequenzen ihrer Anwendung und zu Art und Weise der Verwendung vorliegen [Ga96].

Beispiel:

Ein Beispiel für Entwurfsmuster sind Unternehmensdatenmodelle. Diese legen generelle Design-Entscheidungen fest, so daß zum einen eine unternehmensweit ähnliche Datenbankwelt entsteht, zum anderen werden Aufwände zur Spezifikation reduziert. Entwurfsmuster sind programmiersprachenunabhängig, d.h. im genannten Beispiel nicht auf ein DBMS beschränkt.

Entwurfsmuster in der Hardware-Produktion sind beispielsweise die Schaltungslehre allgemein oder Schaltpläne für einzelne Strukturen, wie etwa arithmetische Einheiten. Dabei werden lediglich Aussagen über den logischen Aufbau getroffen, die genaue Anordnung einzelner Bauelemente bleibt offen.

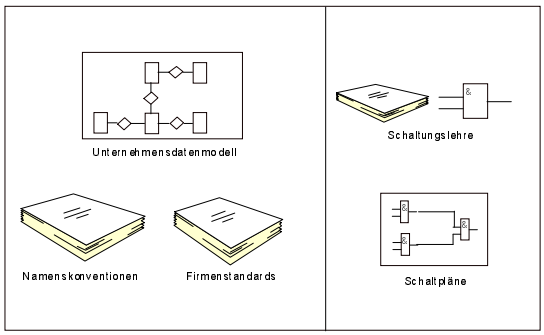

Abb. 1.3: Analogie Design Pattern in Software und Hardware

Die Verwendung von Design Patterns hat insbesondere Einfluß auf Kosten und Risiken der Software-Entwicklung. Die Vorgabe möglicher Lösungswege und Designentscheidungen mindert das Risiko von Fehlentwicklungen erheblich. Da mit der Übernahme von Designentscheidungen häufig auch Folgeergebnisse wiederverwendet werden, sinken die Entwicklungskosten und der Zeitbedarf.

b) Reusable Patterns

Es existieren automatische Verfahren zur Software-Erstellung, die abstrakte Spezifikationen in das Zielsystem überführen [Ni96].

Dabei werden in einer abstrakten, programmiersprachenunabhängigen Beschreibungssprache Anwendungen entwickelt, mit Hilfe von Anwendungsgeneratoren werden aus den abstrakten Beschreibungen unterschiedliche Quellcodes für unterschiedliche Zielumgebungen erzeugt, diese werden mit Hilfe der jeweiligen Compiler zu den ausführbaren Systemen übersetzt und zusammengefügt.

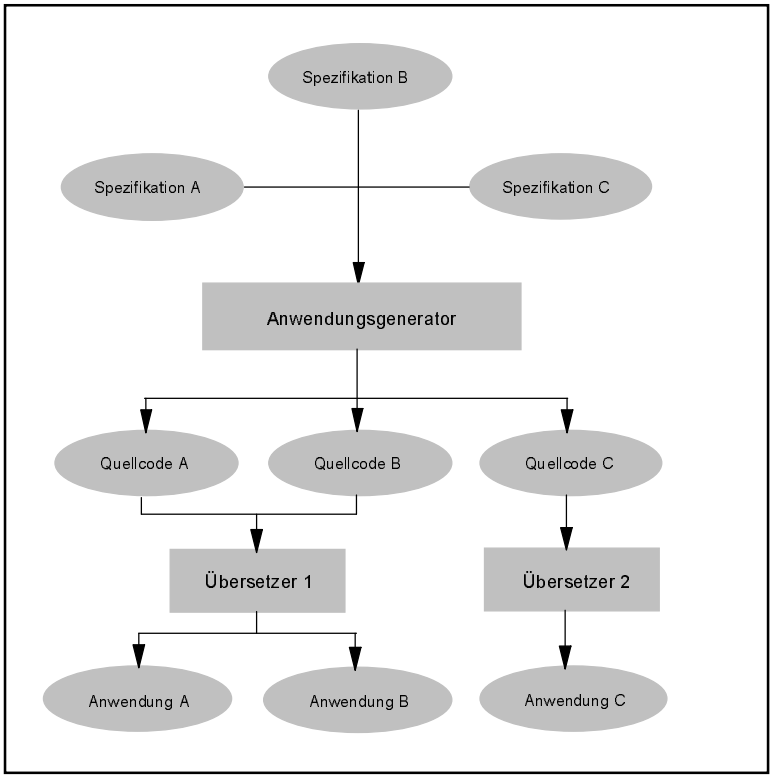

Abb. 1.4: Generative Wiederverwendung (nach Biggerstaff in [Ni96])

Beispiele für die Umsetzung der generativen Wiederverwendung stellen moderne Anwendungsgeneratoren wie zum Beispiel COMPOSER von Texas Instruments dar. Dieses Werkzeug ermöglicht es, ausgehend von einer abstrakten Beschreibungssprache automatisch Anwendungen für verschiedene Betriebssystemsplattformen (WINDOWS, OS/2, WINDOWS NT, aber auch MVS), unterschiedliche Zielsprachen (C, COBOL) und unterschiedliche Datenbankmanagementsysteme (INFORMIX, DB2, ORACLE, SYBASE) zu generieren. Der erzeugte Quellcode kann direkt durch einen Compiler in ein ausführbares Programm umgesetzt werden. Änderungen werden in der abstrakten Beschreibung ausgeführt, durch neue Generierung für die unterschiedlichen Zielsysteme werden alle Anwendungen auf den aktuellen Stand gebracht.

Ein Beispiel für die Anwendung generierender Lösungsverfahren in der Hardware-Produktion stellt die Verdrahtung komplexer Schaltungen dar. Dabei werden lediglich die logischen Verbindungen zwischen den Bauelementen festgelegt. Die

Umsetzung erfolgt automatisch, indem mit Hilfe von Lösungsvorschriften und -regeln die physische Anordnung der logischen Strukturen festgelegt wird. Dadurch sinkt die Fehlerwahrscheinlichkeit deutlich. Es können unterschiedliche Optimierungen erreicht werden.

Die Anwendung der generativen Wiederverwendung führt zur Verbesserung der Software-Qualität. Die automatische Erzeugung des Quellcodes bringt eine verbesserte Korrektheit mit sich, da Änderungen in der Spezifikation durchgeführt werden, verbessern sich die Lesbarkeit, Änderbarkeit und Erweiterbarkeit der Programme und damit auch deren Wartbarkeit. Die Möglichkeit der Erzeugung der Software für unterschiedliche Zielsysteme bedeutet ein hohes Maß an Portabilität.
Die Auswirkungen auf die Kosten von Entwicklung und Betrieb hängen ab von den Kosten für Werkzeuge, die Vorteile im Hinblick auf die Qualität versprechen jedoch auch hier eine Verbesserung. Entwicklungsrisiken sinken aufgrund der Vermeidung syntaktischer Fehler, die Automatisierung von Teilen der Entwicklung führt zu Zeitersparnis.
In Abhängigkeit der eingesetzten Werkzeuge bestehen Möglichkeiten zur Einbindung bestehender eigener und kommerzieller Produkte. Im Fall des im Beispiel genannten Werkzeuges COMPOSER kann über die Schnittstellen DDE, OLE und CORBA (ab Version 4) auf bestehende Software zugegriffen werden. Weiterhin können bestehende COMPOSER-Anwendungen eingebunden werden.

1.2.2.2 Kompositionelle Wiederverwendung

Bei der kompositionellen Wiederverwendung werden neue Anwendungen aus bestehenden Bauteilen entwickelt. Dabei wird, wie in Abbildung 1.5 dargestellt eine Anwendung aus einem Pool verfügbarer Bausteine zusammengestellt.

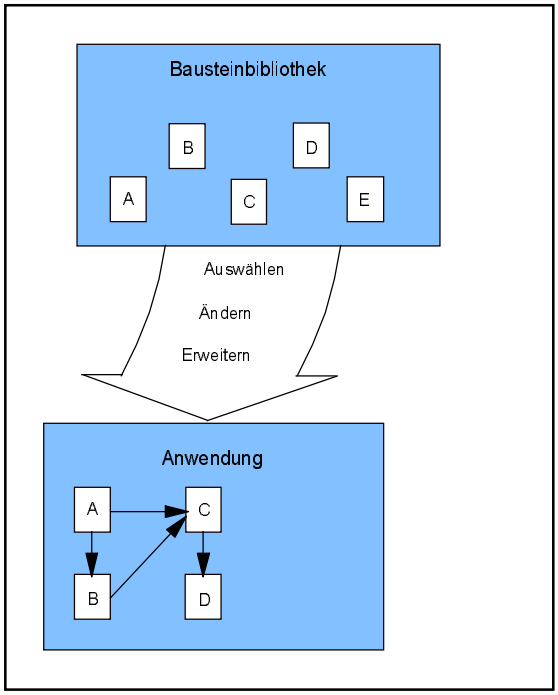

Abb. 1.5 Kompositionelle Wiederverwendung (nach Biggerstaff in [Ni96])

Es gibt zwei Arten der kompositionellen Wiederverwendung, die eng miteinander verknüpft sind, Wiederverwendung von Frameworks und komponentenbasierte Anwendungsentwicklung.

a) Frameworks

Unter Frameworks versteht man generische Architekturen, die aus steuernden Rahmen bestehen, in die die Komponenten des Frameworks eingebettet werden [SO97], [Li96]. Ein Framework ist das Gerüst oder Skelett einer Anwendung, man spricht auch von halbfertigen Bausteinen [St91]. Das Ziel von Frameworks ist es, soviel wie möglich Code, der mehreren Anwendungen gemeinsam ist, zu extrahieren und zu vordefinierten und wiederverwendbaren Bausteinen zusammenzufassen [St91], [Ap96]. Frameworks stellen sich als Sammlung von Teilen dar, die als Gruppe wiederverwendbar sind [Sh97].

Man kann zwischen White-Box-Frameworks und Black-Box-Frameworks unterscheiden. White-Box-Frameworks stellen die mächtigste Art der Wiederverwendung dar. Soll ein White-Box-Framework wiederverwendet werden, so ist Programmieraufwand nötig. Das bestehende Framework wird durch überschreiben erweitert, dazu

müssen Implementierungsdetails bekannt sein, das Zusammenspiel der Teile des Frameworks muß vom Benutzer verstanden werden. Grundprinzip der Wiederverwendung von White-Box-Frameworks ist die Spezialisierung durch Konkretisierung [St91], d.h. Anpassungen erfolgen durch Ersetzen allgemeiner durch anwendungsspezifische Teile des Frameworks.

Das Grundprinzip der Wiederverwendung von Black-Box-Frameworks ist die Spezialisierung durch Auswahl und Parametrisierung, d.h. Anpassungen erfolgen durch Auswahl vorgegebener Parameter. Im Gegensatz zum White-Box-Framework ist der Aufwand zur Nutzung geringer und die Verwendung ist einfacher, da die inneren Mechanismen verborgen bleiben. Um ein Black-Box-Framework wiederzuverwenden ist nur das Verständnis für das "Was tut das Framework?" nötig, das "Wie?" bleibt verborgen. Diese Einschränkung hinsichtlich der Anpaßbarkeit führt zu einer eingeschränkten Nutzbarkeit.

Beispiel:

Ein System von Fenstern der GUI-Oberfläche einer Anwendung ohne die hinterlegte fachliche Logik kann als Framework aufgefaßt werden. Zu den enthaltenen Objekten, wie etwa Schaltflächen oder Menüpunkten sind nur mögliche Ereignisse, wie zum Beispiel Anklicken durch eine Maus oder die Bewegung des Mauszeigers über die Fläche definiert. Die bei Auftreten eines solchen Ereignisses auszuführenden Schritte entsprechen den noch in das Framework einzusetzenden Bausteinen. Ein solches Framework im White-Box-Ansatz kann vom Benutzer noch um neue Elemente erweitert werden (neue Schaltflächen), die vorgegebenen Elemente können verändert werden (z.B. Form und Größe). Im Gegensatz dazu ist die Oberfläche beim Black-Box-Verfahren starr, lediglich durch Parametrisierung können unterschiedliche vorgegebene Ausprägungen gewählt werden, zum Beispiel Oberflächen für Nutzer mit Änderungsberechtigung der zugrundeliegenden Daten und Oberflächen für Nutzer mit Nur-Lese-Berechtigung.

Ein DBMS mit dazugehöriger Anfragesprache kann ebenso als Framework aufgefaßt werden. Das einzubettende Bestandteil zur Erfüllung der gestellten Aufgabe stellt dabei die Datenbank dar [Sh97]. Für verschiedene Aufgaben können

verschiedene Datenbanken in ein und dasselbe DBMS eingebunden werden, ohne daß dieses verändert werden muß (Black- Box).

Ein Beispiel für ein Framework in der Hardware-Herstellung stellen Leiterplatten dar. In die auf der Leiterplatte vorgesehenen Steckplätze können die für die Erfüllung der Aufgabe nötigen Bausteine (integrierte Schaltkreise) eingesetzt werden (Beispiel: Hauptspeicher unterschiedlicher Größe in Abhängigkeit der gestellten Anforderungen im Motherboard eines Computers).

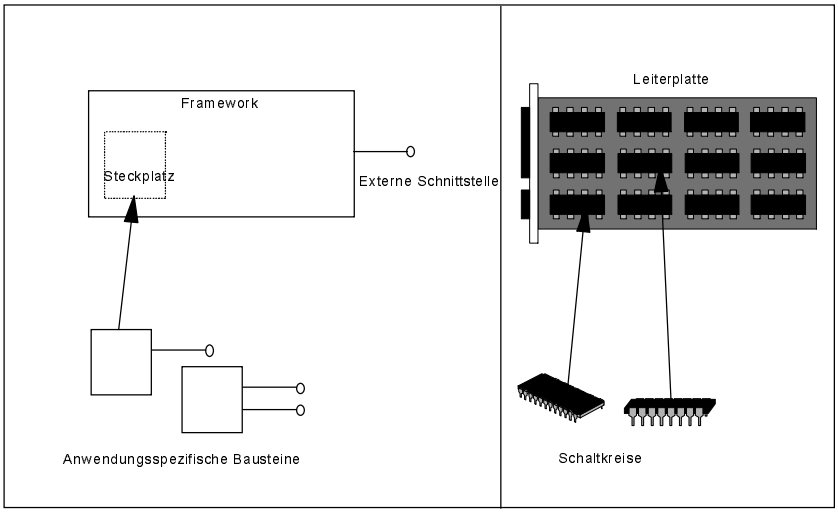

Abb. 1.6: Analogie Framework in Software und Hardware (nach [Sh97])

Die Vorteile der Wiederverwendung von Frameworks liegen in den hohen Wiederverwendungsraten, der Interoperabilität, der einfachen Erweiterbarkeit, Änderbarkeit (White Box) und guten Lesbarkeit. Es sind positive Auswirkungen auf Qualität (Korrektheit, Wartbarkeit) sowie auf Kosten, Risiken und Zeitbedarf (durch Mehrfachnutzung) zu erwarten. Die Einbindung kommerzieller Produkte ist abhängig von zur Verfügung stehenden Schnittstellen. Die Weiternutzung von Altanwendungen wird durch Frameworks nicht unterstützt.

Nachteile von Frameworks sind lange Einarbeitungszeiten, hohe Anforderungen an das Software-technische Wissen sowie die Notwendigkeit umfangreicher Erfahrungen und tiefer Kenntnisse der Anwendungsdomäne zur Extraktion der generischen Architektur [St91]. Die Verwendung mehrerer Frameworks innerhalb einer

Anwendung ist oft schwer [Bi95]. Außerdem führt die Umsetzung unterschiedlichster Anforderungen innerhalb eines Frameworks zu Ineffizienzen hinsichtlich Speicherbedarf und Laufzeit [St91]. Frameworks sind meist sehr groß und starr, sie beeinflussen oft die Struktur einer Anwendung [Bi95].
Frameworks werden hauptsächlich bei Applikationen mit hoher Interaktion, wie z.B. bei Benutzeroberflächen, genutzt [St91], da diese gut zu generalisieren sind [Ap95]. Dagegen sind Anwendungen aus Betriebswirtschaft und Technik schwer generalisierbar, deshalb sind die Möglichkeiten von Frameworks hier sehr gering [Ap96].

b) Komponentenbasierte Anwendungsentwicklung

Ziel der Wiederverwendung von Komponenten ist es, neue Anwendungen zu entwikkeln, indem die entsprechenden vorgefertigten und getesteten Komponenten mit Hilfe eines steuernden Rahmens zu kooperierenden Systemen zusammengesetzt werden [Li96]. Komponenten sind speziell für die Wiederverwendung entwickelte Software-Bausteine, die eine Gruppe von öffentlichen Diensten bieten [SO97]. Die Komponenten werden durch Parametrisierung an die Anforderungen angepaßt.
Auf eine Komponente kann nur über die von ihr gebotenen Dienste, die in einer Schnittstelle definiert werden, zugegriffen werden. Komponenten unterstützen das Prinzip des Information Hiding, indem eine Trennung zwischen Spezifikation ("Was tut die Komponente?") und Implementierung ("Wie tut sie es?") erfolgt.
Beispiel:
Ein Beispiel für eine Software-Komponente ist eine Verwaltung von Personen (Kunden, Angestellte, Geschäftspartner). Aufgrund der ähnlichen Anforderungen an eine solche Verwaltung kann die Komponente in vielfältigen Kontexten (als Kundenverwaltung, Personalverwaltung) in unterschiedlichsten Wirtschaftszweigen (Handel, Versicherungswirtschaft) eingebunden werden. Die zu einer Person gehörigen Daten können kontextunabhängig festgelegt werden, da die zu einer Person gehörenden Angaben vorhersehbar sind. Darauf werden die benötigten Dienste, wie "Person erfassen", "Personendaten ändern", "Person suchen", "Korrespondensanschrift erfassen" usw. definiert.

Auf die gespeicherten Personendaten kann ausschließlich über die verfügbaren Dienste zugegriffen werden. Die Personenverwaltung deckt alle Aufgaben zur Personenverwaltung ab.

Ein Beispiel einer Hardware-Komponente ist eine Festplatte. Sie bietet klar definierte Schnittstellen und Dienste, über die Daten auf dem Speichermedium verwaltet und verändert werden können. Auf die Daten kann nur über die definierte Schnittstelle zugegriffen werden.
In beiden Fällen spielt die tatsächliche Umsetzung der Dienste keine Rolle für den Benutzer, dieser benötigt lediglich Kenntnisse über Dienste und mögliche Parameter.

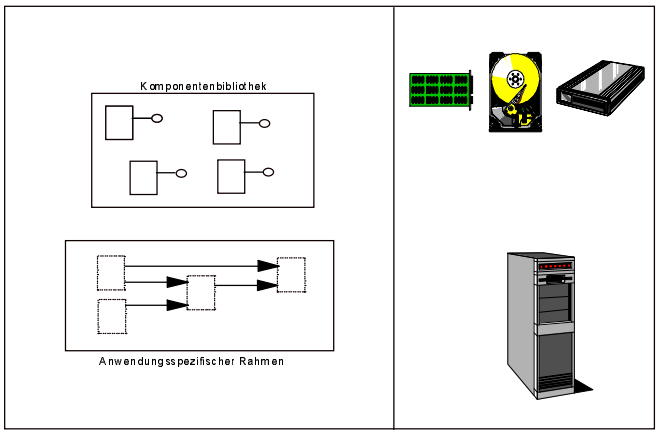

Abb. 1.7: Analogie Komponenten in Software und Hardware

Die Wiederverwendung von wiederverwendbar entwickelten Komponenten verspricht entscheidende Verbesserungen der Software-Qualität, insbesondere der Wartbarkeit. Die Verwendung bewährter Software-Bauteile verbessert die Korrektheit, Risiken werden gemindert. Bei mehrfachem Einsatz werden Kosten gespart. Durch geeignete Vorgehensweisen kann eine Parallelisierung der Entwicklung der Komponenten und des Rahmens zu Zeiteinsparung führen. Durch Schnittstellenstandards (CORBA, OLE, DDE) können kommerzielle Produkte eingebunden werden, soweit die Entwicklungsumgebung dies unterstützt. Durch "wrapper" (siehe Abschnitt 2.2.3) kann auf Altanwendungen zugegriffen werden.

1.2.3 Übersicht der Wiederverwendungsarten

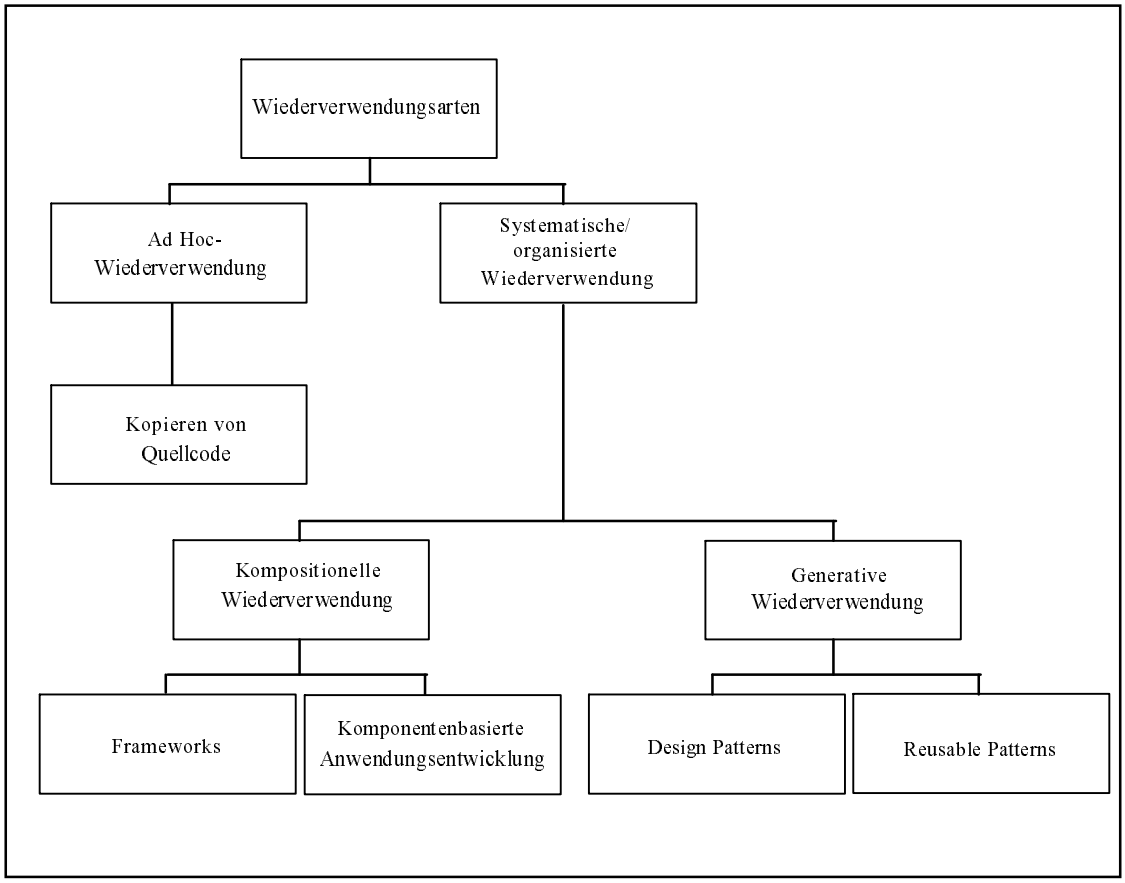

Abb. 1.8: Übersicht der Wiederverwendungsarten

1.3 Hindernisse beim Übergang zur gezielten Wiederverwendung

Bei der Einführung einer gezielten Wiederverwendung treten eine Reihe von Problemen auf, die es zu meistern gilt. Diese sind überwiegend nichttechnischer Natur, aber auch technische Schwierigkeiten müssen überwunden werden. Die Kenntnis dieser Gefahren und Probleme stellt eine wichtige Voraussetzung für deren Überwindung dar. Indiz dafür ist die Tatsache, daß Wiederverwendung in der Software-Erstellung heute eine geringe Rolle spielt, Ursache ist häufig das Scheitern von Wiederverwendungsprojekten an unerwarteten Hindernissen. Aus diesem Grund soll hier ein Überblick gegeben werden.

1.3.1 Technische Hindernisse

Die Veränderung eines Vorgehens und die Einführung neuer Methoden und Werkzeuge bringen immer technische Schwierigkeiten mit sich. So muß auch bei der Einführung der Wiederverwendung mit einigen technischen Problemen gerechnet werden.

Das grundlegende Problem bei der Einführung von Wiederverwendung ist, daß *keine allgemeingültigen Vorgehensweisen* und *keine Patentrezepte* für die Umsetzung existieren [BFB95]. Vielmehr muß die Methode der Wiederverwendung in die bestehende Welt integriert werden [Bö89]. Somit muß jedes Unternehmen eine spezifische Strategie zur Einführung entwickeln [Re95]. Da gerade die Einführung neuer Verfahren besonders kritisch ist, liegt hierin ein erhebliches Gefahrenpotential. Eine genaue Analyse der Software-Entwicklung bildet zusammen mit der Kenntnis möglicher Strategien die Grundlage für die Entscheidung.

Neben einer Strategie für die Einführung der Wiederverwendung muß auch die *strategische Planung von Art und Weise der Wiederverwendung* erfolgen. Dabei spielen Faktoren, wie Struktur der Anwendungen und bestehende Voraussetzungen eine entscheidende Rolle. Eine Hauptproblematik liegt in der Frage:

Was kann wiederverwendet werden ?

Die Entscheidungsgrundlage bilden die Analyse der Anwendungsarchitektur und Untersuchungen der zu erwartenden Erfolge bei Umsetzung der verschiedenen Wiederverwendungsstrategien.

Als besonders problematisch erweist sich die Frage der Granularität der wiederverwendbaren Bausteine.
Zum einen ist der Nutzen bei der Wiederverwendung großer Bauteile im Vergleich zu kleinen höher, andererseits können kleinere Bausteine öfter und mit geringerem Aufwand wiederverwendet werden [Pr96], [Bö89]. Diesen Effekt verdeutlicht Abbildung 1.9:

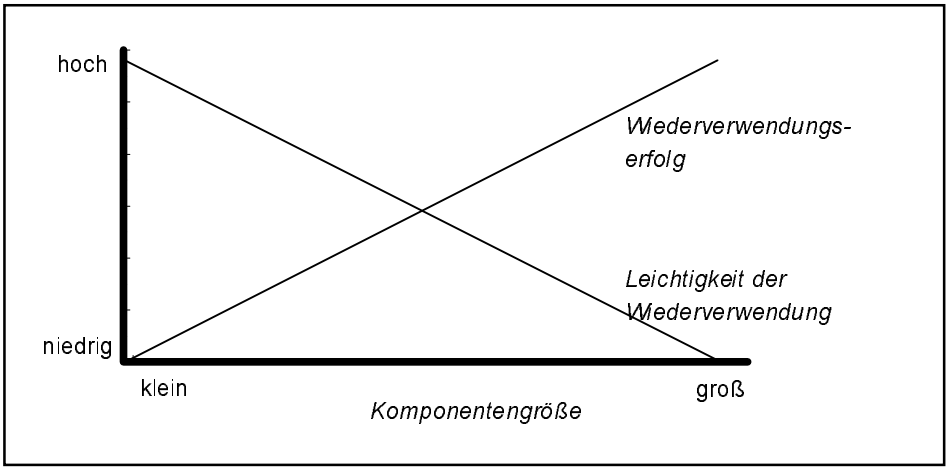

Abb. 1.9: Verhältnis zwischen Granularität von Elementen und deren Einsatz [Pr96]

Das Finden einer optimalen Lösung gestaltet sich vor allem wegen der fehlenden Erfahrung mit Wiederverwendung im jeweiligen Umfeld als schwierig.

Weiterhin ist die Frage zu klären:
Wie kann wiederverwendet werden ?
Entscheidungsgrundlagen sind hier das bestehende Vorgehen und Möglichkeiten und Grenzen der Entwicklungswerkzeuge.

Die Einbindung der Wiederverwendung in eine bestehende Software-Produktionsumgebung stellt *neue Anforderungen an Werkzeuge*. Entweder müssen *zusätzliche Werkzeuge* in die Landschaft eingepaßt werden oder die bestehenden Werkzeuge müssen geeignet erweitert werden. Ursache dafür ist die oft unzureichende Unterstützung der Wiederverwendungaktivitäten, wie zum Beispiel Verwaltung und Wiederverwendung der Bauteile - [Ka97]. Die erforderlichen Werkzeuge stehen oft nicht zur Verfügung, sie müssen also entweder aufwendig erstellt oder gekauft werden.

Ein weiteres technisches Problem ist der anfängliche *Mangel an verfügbaren wiederverwendbaren Bausteinen* [GM93]. In der Anfangsphase der Wiederverwendung existieren nur wenige wiederverwendbare Elemente, somit sind die Aufwände hoch, der Nutzen jedoch vorerst gering. Da gerade in der Einführungsphase die größten Widerstände gegen die Wiederverwendung zu erwarten sind, stellt dieser Mangel einen sehr kritischen Faktor für Erfolg oder Mißerfolg dar.

Ein wesentliches Hindernis im Hinblick auf breite Wiederverwendung ist das *Fehlen anerkannter Standards*. Dies gilt zum einen für Schnittstellen, wo momentan eine Reihe konkurrierender Vorgaben (CORBA, ACTIVE-X, DCOM) die Entscheidung behindern, zum anderen fehlen in vielen Bereichen Branchenstandards zu Daten- und Funktionsmodellen, die einen Austausch von Software-Teilen über Unternehmensgrenzen hinweg ermöglichen.
Bestehende veraltete, aber trotzdem etablierte de-facto-Standards behindern schnelle Veränderungen [Po93]. So konkurrieren bestehende Firmenstandards mit neuen Vorgaben. Auch hier müssen Lösungsansätze entwickelt werden.

Das breite Spektrum möglicher Anwendungen, in denen ein Bauteil wiederverwendet werden könnte, führt sehr schnell zu sehr komplexen Anforderungen. Eine zu große Komplexität behindert die Wiederverwendung jedoch schon allein wegen der schlechten Verständlichkeit des Bauteils. Auf der anderen Seite sind häufig bei der Erstellung nicht alle Kontexte der späteren Verwendung bekannt, d.h. es müssen alle eventuell auftretenden Probleme ohne Kenntnis des Umfeldes ausgeschlossen werden. Weiterhin bestehen oft gegensätzliche Anforderungen, wie zum Beispiel Speicher- vs. Zeitbedarf, die berücksichtigt werden sollen. Es muß also von vornherein geklärt werden, was ein wiederverwendbares Bauteil leisten soll und was nicht.

1.3.2 Nichttechnische Hindernisse

Sehr kritisch für Erfolg oder Scheitern von Wiederverwendung sind Faktoren nichttechnischer Natur. Während technische Hindernisse meist erwartet werden und geeignete Maßnahmen dagegen getroffen werden, fehlt das Bewußtsein für nichttechnische Problematiken oft ganz. Dies gilt insbesondere für soziologische Problematiken.

1.3.2.1 Soziologische Faktoren

Im Mittelpunkt der Softwareentwicklung steht der Mensch, Software wird von Menschen für Menschen entwickelt. Auf Änderungen der Umwelt reagieren sowohl der einzelne als auch die Gruppe meist mißtrauisch und abgeneigt. Diese individuellen und kollektiven Widerstände liegen in verschiedenen Eigenschaften begründet :

a) Individuelle Faktoren (nach [Ju96])

Künstler-Syndrom

Programmierer sehen sich häufig mehr als Künstler denn als Arbeiter. Deshalb wird in der Aufforderung zur Wiederverwendung oft eine Einschränkung der eigenen Kreativität gesehen. Die Auffassung, etwas besser machen zu können, dominiert über das Bewußtsein, daß es sehr oft zweckmäßiger ist, etwas bestehendes wiederzuverwenden. Oft fehlt Entwicklern ein Kostenbewußtsein, das den betriebswirtschaftlichen Nutzen einer neuentwickelten Ideallösung gegen die dadurch entstehenden Kosten abschätzt. Im allgemeinen sind Neuentwicklungen beliebter als das Anpassen bestehenden Codes, da das Produkt ein eigenes ist und nicht "nur etwas Verändertes".

Angst vor Standards

Auch die Einführungen von Standards und Vorgaben zu Art und Weise der Programmierung schränkt die Kreativität ein. Oft ist schon bei Namenskonventionen eine Reihe von Verstößen festzustellen. Standards haben meist Schwächen, die durch Kompromisse zwischen unterschiedlichen Anforderungen zustande kommen, es ist

jedoch sicher, daß ein nicht perfekter Standard immer noch weitaus besser ist als eine fehlende Festlegung.

The Eggheads [1]*-Syndrom*

Das Wissen des Entwicklers stellt für diesen eine gewisse Machtposition dar, daß heißt, andere sind zum Teil von diesem Wissen abhängig. Der öffentliche Zugang zu seinem Wissen führt somit in den Augen des Entwicklers zu Machtverlust. Deshalb werden oft nicht alle Informationen weitergegeben, dies kann zu nicht unerheblichen Behinderungen bei der Wiederverwendung führen.

The Feudal Lords [2]*Syndrom*

Führungskräfte messen ihre Bedeutung im Unternehmen häufig an Faktoren wie Anzahl der Mitarbeiter, Anzahl der Projekte, Höhe ihres Budgets. Die Einführung von Wiederverwendung führt zu kleineren organisatorischen Einheiten, dies führt zum Verlust von "Wichtigkeit", dies kann vom einzelnen als Gefahr für die Karriere gesehen werden.

Auch der einzelne Mitarbeiter sieht in neuen Verfahren oft ein Gefahr für die eigene Karriere, wenn er befürchten muß, daß andere mit den neuen Verfahren schneller besser vertraut werden, die ihm bei den herkömmlichen Methoden unterlegen sind. Damit dieser Mitarbeiter nicht auf der Karriereleiter überholt wird, wird er versuchen, die neuen Methoden zu be- oder sogar verhindern.

b) Kollektive Faktoren (nach [Ju96])

"Not Invented Here" [3]*-Syndrom*

Während jeder Entwickler und jedes Team eigene Programmteile und Erfahrungen ohne große Probleme versucht wiederzuverwenden, bereitet die Verwendung fremder Bausteine oft Unbehagen. Ein Vertrauen, wie es der eigene Arbeit entgegengebracht wird, kommt der Arbeit anderer nicht zu. Es besteht häufig die Auffassung: "Das können wir besser - das machen wir besser".

---

1 Egghead - Eierkopf, hier in der Bedeutung der Starrköpfigkeit zu verstehen

2 Feudal Lords - Feudalherren

3 Not Invented Here - Nicht hier entwickelt

Technologie-Syndrom

Neue Technologien bringen immer auch neue Probleme mit sich. Im Gegensatz zu bewährten Methoden wird neuen mit großer Skepsis entgegengetreten. Oft wird nur auf Fehler und Probleme der neuen Technologie gewartet und in diesen Schwierigkeiten wird dann die Bestätigung für die von vornherein gezeigte Abneigung gesehen. Auch die Tatsache, daß Änderungen auch immer höhere Aufwände in der Anfangsphase mit sich bringen und der Erfolg sich meist erst nach einer gewissen Zeit einstellt, führt zu Widerständen, es besteht ein deutlicher Hang zu alten, bewährten Methoden.

"Revenue-Mania" [1]

Der gestiegene Kostendruck in der Mehrzahl der Unternehmen wirkt sich zum Teil negativ auf die Software-Entwicklung aus. Software-Entwicklungsabteilungen sind, wenn das Unternehmen die Software nicht primär für den Verkauf entwickelt, sogenannte "Cost-Only"- Abteilungen. Das heißt, es sind lediglich die Kosten bekannt, der Nutzen der erstellten Software ist meist nicht in Zahlen ausweisbar. Bei wirtschaftlichen Problemen finden Einsparungen zuerst immer in den reinen Kostenabteilungen statt, also auch in der Software-Entwicklung. In diesem Zusammenhang muß man mit erheblichen Widerständen rechnen, da Wiederverwendung zuerst Kosten verursacht und der Erfolg erst später eintritt und dazu noch schwer meßbar ist. Ein weiteres Problem stellt die Kostenabrechnung dar, die durch die Entwicklung für Wiederverwendung entsteht. Dem Ersteller wiederverwendbarer Software-Bauteile entstehen höhere Aufwände, der Nutzen kommt jedoch hauptsächlich Folgeprojekten zugute.

Teilsicht vor Gesamtsicht - Syndrom

Die Notwendigkeit, die Entwicklung großer Anwendungen auf mehrere Projekte zu verteilen, hat oft zur Folge, daß die Gesamtsicht auf die zu erfüllende Aufgabe verlorengeht. Der kurzfristige Erfolg auf Projektebene rückt in den Mittelpunkt. Dies ist

[1] Revenue-Mania - Einnahmen-Wahn

insbesondere für die Entwicklung wiederverwendbarer Software hinderlich, da hier die Kenntnis unternehmensweiter Anforderungen nötig ist.

Neben den Widerständen des einzelnen und der Gruppe existieren noch eine Reihe weiterer Hemmnisse. So fehlt es im allgemeinen an einer *fundierten Ausbildung im Hinblick auf Wiederverwendung* [He93], so daß ein *erheblicher Lernprozeß erforderlich* ist. Häufig fehlt neben dem Willen auch die Fähigkeit, global zu denken oder sich in fremde Denkweisen hineinzuversetzen.
Auch vom Management wird der Einführung von organisierter Wiederverwendung nicht immer die notwendige Unterstützung entgegengebracht [Ka97]. Ursache dafür ist die Unsicherheit, die dieser Vorgang mit sich bringt, denn es gibt keine Erfolgsgarantien für Wiederverwendung [Ka96], [He93].

Der Überwindung der soziologischen Hemmnisse kommt eine entscheidende Bedeutung zu. Erst durch motivierte Wiederverwendung können Erfolge erzielt werden. Maßnahmen, wie Belohnungssysteme, einzuhaltende Vorgaben im Bezug auf Wiederverwendbarkeit und umfangreiche Weiterbildung und Training im Hinblick auf Wiederverwendung stellen Ansätze zur Überwindung der soziologischen Hemmnisse dar.

1.3.2.2 Ökonomische Faktoren

Unter dem Kostendruck, der auf der Software-Erstellung lastet, spielen auch ökonomische Faktoren eine entscheidende Rolle bei der Umsetzung von Wiederverwendung [CE95]. Den Erwartungen hinsichtlich der Reduzierung der Kosten stehen insbesondere in der Anfangsphase *hohe Investitionen* entgegen [Ka96], [Ka97], [Bö89], [He93]. Diese entstehen durch die *Notwendigkeit neuer Werkzeuge*, durch Aufwände zur *Mitarbeiterschulung* [He93] und durch nötige *Veränderungsprozesse in Organisation und Kultur* [Ka96], [BFB95]. Die erwarteten wirtschaftlichen Erfolge der Wiederverwendung sind jedoch erst über längere Zeiträume zu erreichen [Ka96], da zuerst die nötigen Investitionen erarbeitet werden müssen.
Die Entwicklungszeiten und -kosten für wiederverwendbare Teile sind anfänglich im Vergleich zur Entwicklung einmal einzusetzender Programme höher, zum einen wegen des Mangels an bereits bestehenden Bauteilen, zum anderen wegen des erhöhten

Aufwandes bei der Erstellung wiederverwendbarer Bauteile. Die Kosten, Software wiederverwendbar zu entwickeln, sind durchschnittlich etwa 25% höher, als die bei der Entwicklung für den einmaligen Ansatz, aber auch um 30 - 200% höhere Aufwände können entstehen [Bö89]. Eine wiederverwendbare Komponente muß dreimal[1] genutzt werden, damit sich die höheren Kosten auszahlen [McC93].

1.3.2.3 Organisatorische Faktoren

Organisierte Wiederverwendung erfordert Änderungen im Vorgehensmodell der Software-Erstellung. Herkömmliche Prozesse sehen keinerlei Maßnahmen zur Wiederverwendung vor. Es müssen zusätzliche Aktivitäten, wie Zertifizierung, Klassifizierung und Speicherung neuer wiederverwendbarer Elemente, sowie Suche, Bereitstellung und Einbindung bestehender wiederverwendbarer Elemente in das Vorgehensmodell eingefügt werden. Dazu müssen eine Reihe neuer Strukturen in der Organisation des Unternehmens geschaffen werden, die Zuständigkeiten und Kompetenzen müssen geklärt werden. Das Fehlen einer Wiederverwendungs-Infrastruktur [Ka97] behindert die schnelle Umsetzung. Weiterhin muß festgelegt werden, wie, wann, von wem und in welchem Umfang die Bauteile gewartet werden.
Der kurzfristige Erfolg wird häufig in den Vordergrund gestellt, die langfristige Planung stellt jedoch bei Wiederverwendung eine wichtige Voraussetzung dar.

1.3.2.4 Rechtliche Faktoren

Nutzungs- und Verwertungsrechte sowie Gewährleistungsverpflichtungen sind juristische Aspekte, die bei der Wiederverwendung zu prüfen bzw. vertraglich zu regeln sind. [Ka96]
Die Globalität des Marktes führt in vielfältiger Hinsicht zu Unklarheiten, was rechtliche Aspekte angeht. Es müssen Bedingungen geschaffen werden, die Rechte und Pflichten bei der Wiederverwendung von Softwareprodukten regeln.

---

[1] in [Ka96] als "Biggerstaffs Formel 3" bezeichnet

1.4 Zusammenfassung

Wiederverwendung ist ein erfolgversprechender Ansatz, die Software-Entwicklung zu verbessern. Wesentlich für den Erfolg der Wiederverwendung sind sowohl soziologische als auch technische Aspekte. Einer Einführung von Wiederverwendung im Unternehmen muß eine Analyse der Erwartungen und der bestehenden Situation vorausgehen, um eine erfolgversprechende Strategie zu entwickeln. Die genaue Kenntnis der Möglichkeiten von Wiederverwendung stellt hierbei eine unverzichtbare Grundlage dar.

Die Einführung der Wiederverwendung muß schrittweise und geplant in Abhängigkeit der bestehenden Voraussetzungen erfolgen. Die Erwartungen müssen realistisch sein, Ziele müssen genau definiert werden, um eine spätere Analyse des Erfolgs zu ermöglichen.

Im Hinblick auf die zu untersuchende Landschaft bei der R+V-Versicherung soll im weiteren die komponentenbasierte Anwendungsentwicklung betrachtet werden.

2. Komponentenbasierte Anwendungsentwicklung

Grundprinzip der komponentenbasierten Anwendungsentwicklung ist es, Anwendungen durch Zusammensetzen von vorgefertigten Bauteilen, den Komponenten, zu erstellen.
Die Idee ist nicht neu, bereits 1968 präsentierte McIlroy von den Bell Laboratorien auf der NATO-Konferenz in Garmisch seine Idee der "Massenproduktion wiederverwendbarer Software-Komponenten" [McI68].
Die Verbindung der Komponenten erfolgt über einen steuernden Rahmen, der anwendungsspezifisch erstellt werden muß. Die Komponenten sind so zu entwickeln, daß sie in vielfältigen Kontexten einsetzbar sind [McC93].
In diesem Kapitel sollen nach der Klärung des Komponentenbegriffs die Prinzipien der komponentenbasierten Anwendungsentwicklung dargestellt werden. Weiterhin werden zu schaffende Voraussetzungen genannt und Wege zur Einführung komponentenbasierter Anwendungsentwicklung aufgezeigt.

2.1 Was ist eine Komponente

Zur Verdeutlichung, was unter einer Komponente zu verstehen ist, sollen nach der Definition die Anforderungen an Komponenten aufgelistet werden. Danach sollen mögliche Ausprägungen von Komponenten aufgezeigt werden.

2.1.1 Begriffsklärung

Die Klärung des Verständnisses von Komponenten muß in jedem Unternehmen individuell vorgenommen werden, da unternehmensspezifische Faktoren, wie bestehende Entwicklungsumgebung, Vorgehensmodell, Anwendungsgebiet u.s.w. Ausprägung und Eigenschaften einer Komponente beeinflussen. Trotzdem sind einige Merkmale allgemeingültig, wie folgende Definitionen von Komponenten zeigen:
"Eine Software-Komponente ist ein Code-Modul, daß ein oder mehrere klar definierte Dienste implementiert, die durch einen Satz von öffentlichen Schnittstellen festgelegt sind."[1] [TI95]

[1] "A software component is a unit of code that implements one or more clearly defined services as defined by a set of published interfaces."

Die Schnittstelle einer Komponente bietet wohldefinierte Möglichkeiten zur Interaktion und Kommunikation [NT95].
"Ein Software-Fragment ist dann eine Komponente, wenn es speziell für Wiederverwendung entwickelt wurde und Teil eines Systems ist " [1](nach [NT95]).

"Eine Komponente ist ein unabhängig auslieferbares Paket von Software-Operationen. Das heißt, die Komponente ist ein in sich geschlossenes Software-Paket, das durch die von der Komponente gebotenen Operationen genutzt werden kann." [2][TI96]
Eine Komponente besteht aus drei Teilen, der Spezifikation, der Implementierung und aus dem ausführbaren Programm. Die Spezifikation beinhaltet die für die Benutzung notwendigen Datensichten, die darauf ausführbaren Operationen (Services) und deren Beschreibungen. Die Implementierung umfaßt die tatsächliche Umsetzung der spezifizierten Eigenschaften, sie sollte im allgemeinen für den Nutzer nicht sichtbar sein (Information Hiding). Das ausführbare Programm wird entweder direkt ausgeliefert oder es muß vom Benutzer in der Zielumgebung aus der Implementierung erzeugt werden. Weiterhin gilt: zu einer Spezifikation können mehrere Implementierungen und zu einer Implementierung wiederum mehrere ausführbare Programme existieren [TI96].

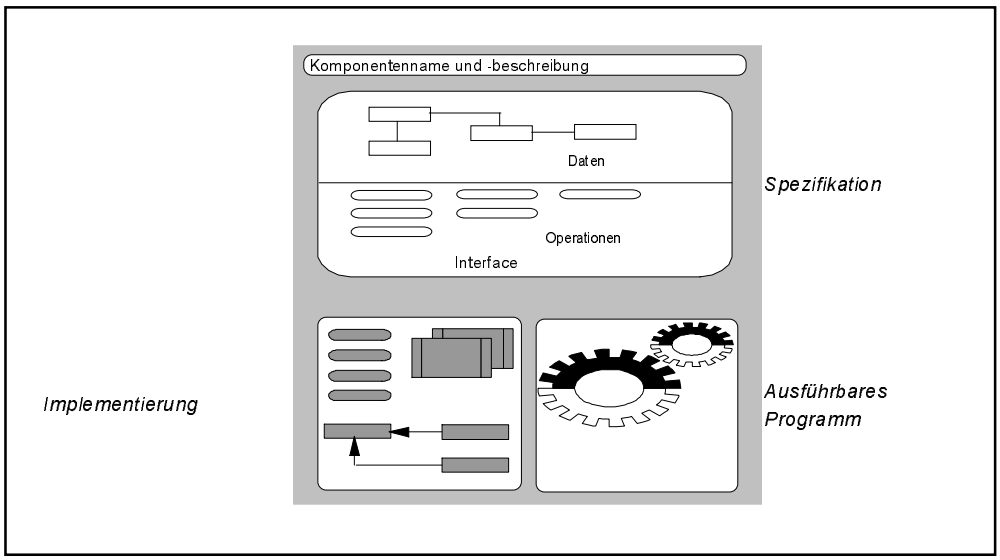

Abb.2.1: Bestandteile einer Komponente (nach [TI96])

[1] "A software fragment is a component, when it is designed for reuse and is part of a framework."
[2] "A component is an indipendently deliverable package of software operations. This means the component is a self-contained package of software that can be used through the operations offered by that component."

In dieser Arbeit wird der Begriff der Komponente wie folgt festgelegt:

Definition Komponente:
Eine Komponente ist ein speziell für die Wiederverwendung entwickelter Software-Baustein, der über eindeutig definierte Schnittstellen Dienste anbietet. Die Nutzung der Komponente erfolgt ausschließlich über diese Dienste, dabei wird das Prinzip des Information Hiding unterstützt. Die Komponente ist weitgehend unabhängig von dem Verwendungskontext, sie genügt definierten Qualitätsanforderungen und erfüllt dokumentierte Testfälle.

2.1.2 Anforderungen an eine Komponente

Eine Komponente muß eine Reihe von Anforderungen erfüllen, damit ihre Wiederverwendbarkeit gewährleistet bzw. verbessert wird (nach [Bö89], [Bi95], [Ap96], [McC93], [GM93], [Ap96]).

Verständlichkeit	Die Komponente ist leicht zu verstehen, der Nutzer kann ihre Funktionalität schnell interpretieren, es existieren geeignete Notationen für die Beschreibung der Komponente, ihrer Wirkungen und ihrer Wiederverwendung. Die Beschreibungsformen müssen in der betrachteten Domäne verbreitet, akzeptiert und als Verfahren geeignet sein [Kü94].
Definiertheit	Das Verhalten der Komponente muß für alle Situationen vorhersehbar sein.
Übertragbarkeit	Die Komponente ist auch in anderen, als den Entwicklungskontexten, anwendbar.
Zugreifbarkeit	Die Komponente ist mit Hilfe einer Beschreibung oder eines Browsing-Mechanismus wiederfindbar.
Hohe Kohäsion (Zusammenhalt)	Die Komponente umfaßt alle zusammengehörigen Operationen, d.h. die Komponente sollte vollständig sein, also verschiedene Aspekte der Nutzung berücksichtigen [Gö93].

Lose Kopplung	Die Komponenten sind weitgehend unabhängig voneinander (mit Ausnahme des Aufrufs), eine hohe Kopplung beeinträchtigt die Verwendbarkeit, da eine komplette Umgebung übernommen werden muß.
Additivität	Die Komponente kann mit minimalen Nebenwirkungen und ohne zerstörende Interaktion mit anderen Komponenten verbunden werden.
Austauschbarkeit	Eine Komponente ist durch eine andere problemlos ersetzbar, die mindestens die gleichen Dienste anbietet und mindestens die gleichen Datensichten bietet.
Konfigurierbarkeit	Anpaßbarkeit einer Komponente an konträre oder ähnliche Anforderungen, wie zum Beispiel hohe Performanz vs. geringer Speicherbedarf oder unterschiedlich große verfügbare Ressourcen.
Information Hiding	Der Nutzer kennt von der Komponente lediglich die expliziten Schnittstellen, das heißt die importierten und exportierten Daten und die nutzbaren Operationen. Die internen Daten und Funktionen bleiben für den Benutzer verborgen.
Kapselung	Unabhängigkeit der Daten einer Komponente von externen Datensichten zum Beispiel durch Ersetzung von Sichten auf globale Daten durch adäquate Lese- oder Schreiboperationen.
Portierbarkeit	Es existieren keine Einschränkungen hinsichtlich Hard- oder Softwareplattform.
Kompatibilität zu Firmenstandards	Die Komponente muß Festlegungen innerhalb des Unternehmens zum Beispiel hinsichtlich Dialoggestaltung und Namenskonventionen genügen.
Korrektheit	Die Komponente erfüllt eine wohldefinierte, leicht verständliche Aufgabe vollständig und richtig.
Zuverlässigkeit	Die Komponente ist fehlerfrei und robust gegenüber unerwarteten Situationen.

Hierarchische Strukturierbarkeit	Komponenten können selbst andere Komponenten nutzen [Gö93].

Tab. 2.1: Anforderungen an Komponenten

Während Zugreifbarkeit, Verständlichkeit und die hohe Kohäsion das schnelle Finden und Verstehen der Komponenten durch den Nutzer ermöglichen, sollen durch die anderen Eigenschaften Akzeptanz und Flexibilität erhöht werden.

2.1.3 Arten/ Ausprägungen von Komponenten

Welche Software-Bausteine zu Komponenten umgebaut werden, hängt stark von der Entwicklungsumgebung des Unternehmens ab. Garnett und Mariani definieren in [GM93] verschiedene Kandidaten für die Erstellung von Komponenten:

a) Funktionen

Funktionen sind Abstraktionen über Ausdrücken, die unter Verwendung von Parametern einen Rückgabewert liefern. Funktionen müssen durch Ersetzen der globalen Variablen und Seiteneffekte durch Parameter umgebungsunabhängig gemacht werden.

b) Prozeduren

Prozeduren sind Abstraktionen über Anweisungen, sie sind abhängig von ihrer Umgebung und ihre Ausführung bewirkt Änderungen der Umgebung. Die Abhängigkeit von der Umgebung widerspricht der Forderung nach Additivität, da Nebeneffekte auftreten. Erst durch Ersetzen globaler Referenzen durch Übergabeparameter können die Nebeneffekte eliminiert und somit die Komponentenbedingungen erfüllt werden.

c) Objekte

Objekte sind Abstraktionen über Variablen. Sie setzen sich aus einem Daten- und einem Funktionsteil zusammen.

d) Abstrakte Datentypen

Abstrakte Datentypen sind Abstraktionen über Typen. Sie bestehen aus Repräsentanten und einem Satz von auf Objekte des Typs anwendbaren Prozeduren.

f) Subsysteme

Subsysteme sind Abstraktionen über Programmen, sie können selbst wieder aus Komponenten gefertigt sein.

Die genannten Elemente an sich stellen keine Komponenten im Sinne dieser Arbeit dar, vielmehr sind es mögliche Kandidaten, die geeignet erweitert (siehe Anforderungen in Abschnitt 2.1.2) als Komponenten angesehen werden können.

2.2 Prinzipien der Anwendungsentwicklung aus Komponenten

Anwendungen werden komponentenbasiert entwickelt, indem das zu lösende Problem in zusammenhängende Teilprobleme zerlegt wird. Diese Teilprobleme sollten abgeschlossen sein, d.h. alle gleichartigen Teilaufgaben beinhalten. Die Lösung eines Teilproblems wird dem Entwickler des Gesamtsystems in Form einer Komponente zur Verfügung gestellt. Das Entwickeln des Gesamtsystems gestaltet sich lediglich in der Spezifikation der Anforderungen und im anschließenden Zusammenfügen der Komponenten durch einen steuernden Rahmen. In Abbildung 2.2 ist zu erkennen, daß die Komponenten in unterschiedlichen Anwendungen verwendet werden.

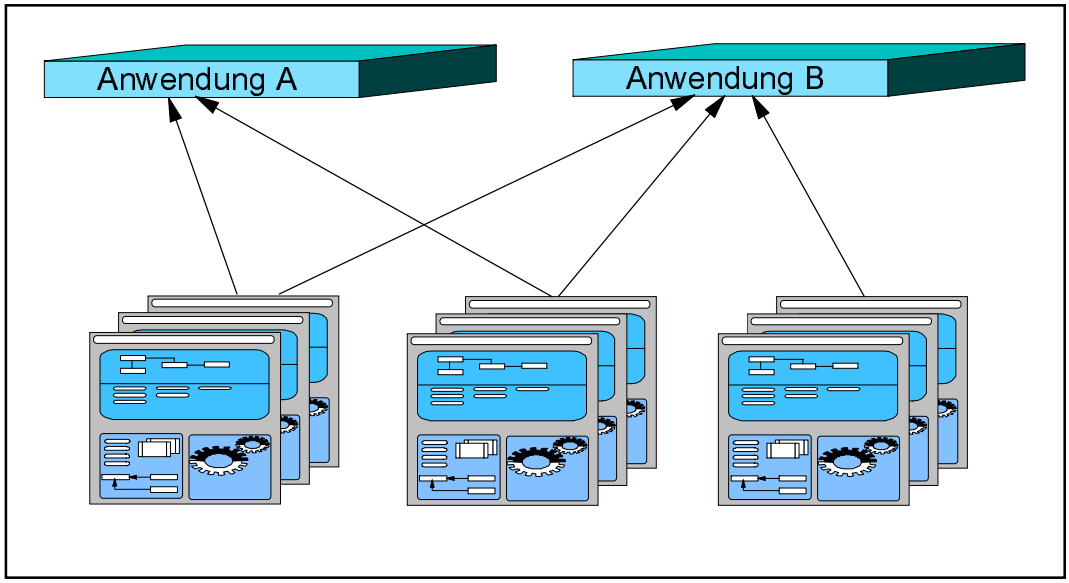

Abb. 2.2: Komponentenbasierte Anwendungsentwicklung

2.2.1 Vergleich mit Objektorientierung

Der Übergang von der prozeßorientierten zur objektorientierten Programmierung verspricht die Software-Entwicklung in vielfacher Hinsicht zu verbessern. Das Fehlen betriebswirtschaftlich nutzbarer Werkzeuge und Umgebungen behindert die Durchsetzung der Objektorientierung jedoch erheblich. Aus diesem Grund wurden Möglichkeiten gesucht, objektorientierte Prinzipien auch mit nicht objektorientierten Entwicklungswerkzeugen umzusetzen. Wie in der Einleitung angedeutet stellt die komponentenbasierte Anwendungsentwicklung einen Versuch dar, die momentan betriebswirtschaftlich erfolgversprechendsten Aspekte der Objektorientierung zu nutzen, ohne auf die Verfügbarkeit von im kommerziellen Umfeld einsetzbaren Entwicklungs- und Laufzeitumgebungen warten zu müssen. Nachfolgend sollen die Möglichkeiten der Objektorientierung und ihre Umsetzung in Komponenten dargestellt werden.

a) Datenabstraktion

Die zentrale Veränderung beim Übergang zu objektorientierter Software-Entwicklung stellt die Datenabstraktion dar. Das Zusammenspiel der Objekte der realen Welt bildet die Grundlage der Anwendungsarchitektur.

Zu den Objekten werden Methoden bereitgestellt, über die exklusiv jeglicher Datenzugriff gewährleistet wird. Dabei kann eine Methode beliebig oft und aus beliebigem Kontext aufgerufen werden [MO94].

Analog werden bei der komponentenbasierten Entwicklung von Software Elemente der realen Welt als Komponenten dargestellt. Auf eine Komponente kann ausschließlich über in einer Schnittstelle bereitgestellte Dienste zugegriffen werden. Der Aufruf dieser Dienste soll kontextunabhängig sein.

Für datenintensive Anwendungen birgt die Datenabstraktion das größere Potential der Wiederverwendung als funktionale Abstraktion [Qu94].

b) Information Hiding

Die Verwendung von Komponenten erfolgt wie der Methodenaufruf bei Objekten ohne Kenntnis interner Strukturen allein auf Basis ihrer Schnittstelle. Damit werden Strukturen und Zusammenhänge einer Anwendung übersichtlicher.

c) Kapselung

Die Komponenten werden wie die Objekte gekapselt, d.h. es existieren keinerlei Verbindungen über die Grenzen der Komponente hinweg mit Ausnahme des Aufrufes von Diensten anderer Komponenten.

d) Polymorphismus

Das Prinzip des Polymorphismus, d.h. die Möglichkeit, die gleiche Methode in unterschiedlichen Objekten unterschiedlich zu implementieren wird nur teilweise auf Komponenten übertragen. Erst die Verwendung von Object Request Brokern ermöglicht die Umsetzung von Polymorphismus.

e) Vererbung

Der generelle Unterschied zur Objektorientierung besteht im Fehlen von Vererbungsmechanismen. Ein Grund dafür sind die hohen Kosten der Umsetzung. Weiterhin erfordert die Nutzung der Vererbung von Methoden und deren Überschreiben erhebliche Disziplin seitens der Entwickler [MO94]. Vererbung spielt für Wiederverwendung eine untergeordnete Rolle, vielmehr steht Wiederverwendung durch Komposition im Vordergrund [Ga96]. Bei der komponentenbasierten Entwicklung steht mit Aggregation eine Möglichkeit zur Verfügung, vorhandene Komponenten zu "erben" (Abbildung 2.3).

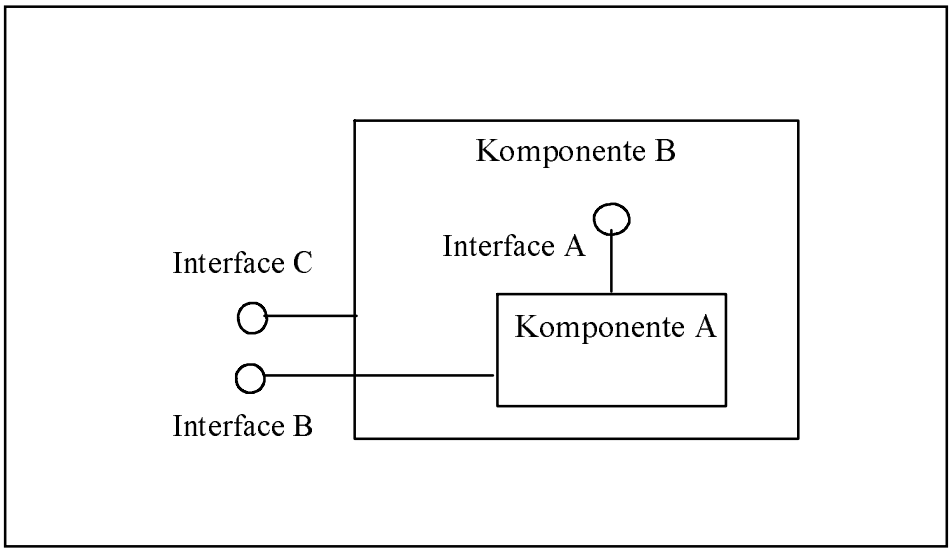

Abb. 2.3: Aggregation von Komponenten (nach [MO94])

Wie Abbildung 2.3 zeigt, können bei der Aggregation sowohl Teile der Schnittstelle der übernommenen Komponente verdeckt werden (Interface A), sie können unverändert nach außen gegeben werden (Interface B) und es können neue Methoden hinzugefügt werden (Interface C). Dies schließt auch die Veränderung (Überschreiben) eines Dienstes ein (z.B. Ersetzen von Interface A durch identisches Interface C mit anderer Implementierung) [MO94].

Ein Vorteil der Anwendungsentwicklung aus Komponenten gegenüber rein objektorientierten Ansätzen besteht in der Möglichkeit eines schrittweisen Überganges von der bestehenden Software-Entwicklung zum Objektparadigma. Ausgehend von den zentralen Elementen des Geschäftsbereiches können nach und nach mehr Komponenten erzeugt und in die Anwendungen integriert werden. Desweiteren entsteht kein gravierender Bruch in der Vorgehensweise bei der Entwicklung, wie bei einem Übergang zur Objektorientierung in einem Schritt. Damit wird das mit Veränderungen verbundene Risiko verringert.
Da Komponenten mehrere Schnittstellen haben können, in denen die Dienste der Komponente für die Nutzung zur Verfügung gestellt werden, wird eine weitere Kapselung erreicht. Die Dienste werden zu semantischen Einheiten zusammengefaßt, Zugriffsberechtigungen werden auf die Dienste einer Schnittstelle eingeschränkt (siehe Abb. 2.4).

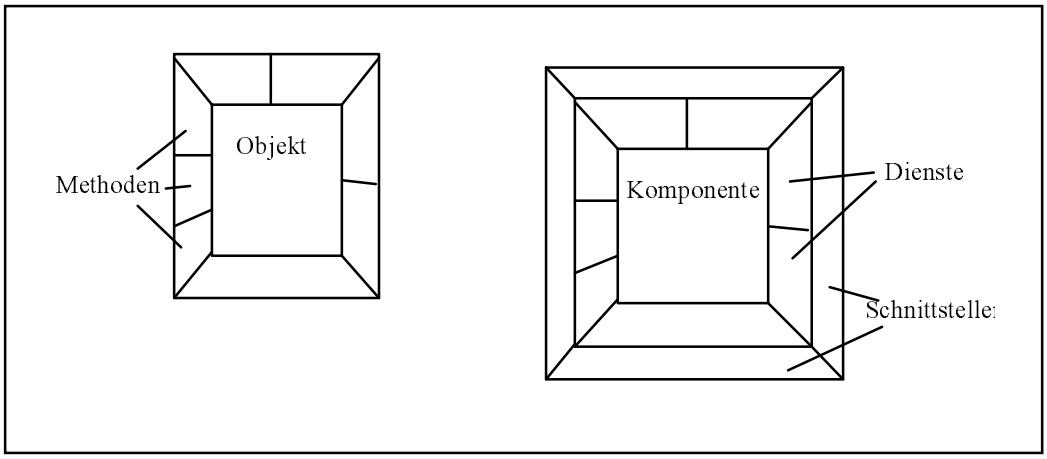

Abb.: 2.4: Kapselung von Komponente vs. Objekt

Eine ähnliche Methodik wird mit dem Component Object Model (COM) eingeführt [MO94].

2.2.2 Repository - Zentrales Element der komponentenbasierten Entwicklung

Die Anwendungsentwicklung aus Komponenten basiert auf der Verwendung vorhandener Bausteine in neuen Anwendungen. Grundvoraussetzung dafür ist ein System zur Verwaltung vorhandener Bausteine, in das neue Komponenten einfach eingebracht und in dem vorhandene Komponenten mit wenig Aufwand gesucht und wiederverwendet werden können. Ein solches System wird als Repository bezeichnet. Dem Repository kommt entscheidende Bedeutung bei Wiederverwendung zu, eine Komponente muß mit geringerem Aufwand gefunden und verwendet werden, als die erneute Erstellung in Anspruch nehmen würde.

2.2.2.1 Anforderungen an ein Repository

Ein Repository muß eine Reihe von Anforderungen erfüllen, um Wiederverwendung effektiv zu unterstützen. Neben Speicherung, Suche, Bewertung und Einbindung der Komponenten [Ka96] müssen Möglichkeiten der Konfigurations- und Änderungsverwaltung eingebunden sein [Li96]. Die Existenz unterschiedlicher Versionen einer Komponente muß abbildbar sein. Die Möglichkeit der Historisierung der Verwendung der einzelnen Komponenten ist nötig, um bei Änderungen betroffene Anwendungen zu informieren oder vor Änderungen die Änderungsauswirkungen zu untersuchen [TI95].
Das Repository sollte unternehmensweit zugänglich sein, um Wiederverwendung auch über Abteilungsgrenzen hinweg zu unterstützen [Ka96]. Weiterhin sollten Komponenten unterschiedlicher Entwicklungswerkzeuge verwaltet werden, Möglichkeiten zum Austausch über Werkzeuggrenzen verbessern die Möglichkeiten der Wiederverwendung. Notwendig dafür sind gemeinsame Austauschformate, Speicherformate und gemeinsame Zugriffs- und Beschreibungsformen [TI95].

2.2.2.2 Ablegen von Komponenten im Repository

Die in einer Qualitätskontrolle zertifizierten Komponenten müssen im Repository so gespeichert werden, daß sie mit minimalem Aufwand und maximaler Genauigkeit wiedergefunden werden. Dazu ist im Hinblick auf eine große Anzahl verfügbarer Komponenten ein Klassifikationsschema notwendig.

Generell kann man zwischen Klassifikation mit kontrolliertem und unkontrolliertem Vokabular unterscheiden [AF93]. Bei der Verwendung von kontrolliertem Vokabular dienen vorgegebene Schlagworte zur Beschreibung der Komponenten, durch geeignete Kombination der Schlagworte wird die Komponente eingeordnet. Vorteile dieses Verfahrens liegen im einfachen Finden der Komponenten anhand von Schlagworten, Vermeidung von Synonymen und in der Einfachheit der Suche, Nachteile sind im hohen Aufwand der Klassifikation und in der Einschränkung der Beschreibungsformen zu sehen [AF93].

Bei der Klassifikation mittels unkontrolliertem Vokabular können die Schlagworte frei vergeben werden. Meist erfolgt die Beschreibung im Text. Der Aufwand zum Klassifizieren ist gering, dafür wird das Finden relevanter Komponenten durch Probleme wie Homonyme, Synonyme und unklare Bedeutungen erschwert. Das Suchen mittels Volltextrecherche erfordert sowohl höheren Aufwand als auch genauere Kenntnis von Suchstrategien.

Abbildung 2.5 stellt das prinzipielle Vorgehen schematisch dar.

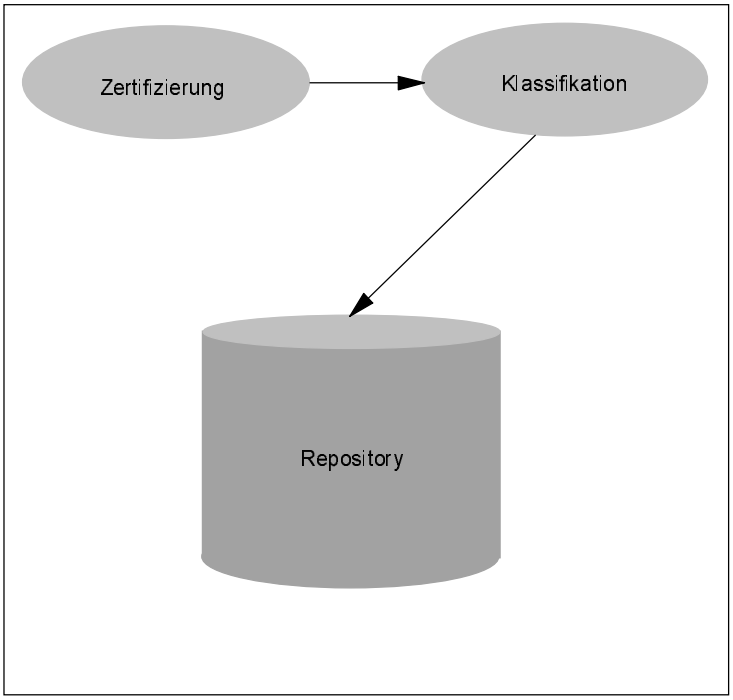

Abb. 2.5: Klassifikation von Komponenten (nach [AF93])

2.2.2.3 Einbinden von Komponenten aus dem Repository in eine Anwendung

Die Übernahme von Komponenten in eine zu erstellende Anwendung erfolgt in drei Schritten: Suche der Komponenten, Bewertung der gefundenen Kandidaten und Auslieferung der benötigten Quellen.

a) Suche relevanter Komponenten

Die in 2.2.2.2 eingeführte Klassifikation dient zur Vorauswahl möglicher Kandidaten von Komponenten, die die zu lösende Aufgabe erfüllen. Je nach Art der Klassifikation wird mittels Volltextsuche (unkontrolliertes Vokabular) oder navigierender Suche (kontrolliertes Vokabular) gesucht. Werden beide Möglichkeiten angeboten, erhöhen sich die Wahrscheinlichkeit, entsprechende Komponenten zu finden und die Akzeptanz der Nutzer (wegen individueller Vorlieben für die eine oder andere Variante).

b) Bewertung der gefundenen Komponenten hinsichtlich der Eignung

Zu den gefundenen Komponenten müssen dann weitere Informationen angeboten werden, anhand derer entschieden werden kann, ob die gefundene Komponente eingesetzt werden kann. Diese zusätzlichen Informationen sind entweder im Repository selbst enthalten, oder es existieren Verweise zu entsprechenden Informationsquellen. Folgende Informationen, sind für die Wiederverwendung einer Komponente nötig und hilfreich:

Name	aussagekräftiger Bezeichner
Absicht	kurz (einSatz) welches Problem wird gelöst
Synonyme	alternative Namen
Motivation	wo einsetzbar (ausführliche Beschreibung)
Anwendbarkeit	wann anwendbar
Struktur	graphische Struktur mit allen beteiligten Komponenten
Teilnehmer	Kurzbeschreibung der einzelnen Elemente
Interaktion	Beschreibung des Zusammenwirkens, um das gewünschte Verhalten zu erzielen
Konsequenzen	erwartete Wirkungen, Einschränkungen der Verwendung
Referenzen	zu erfolgreichem Einsatz
Beziehungen zu anderen Komponenten	mit welchen anderen Komponenten kombinierbar
Spezifikation	vollständige fachliche Beschreibung der Elemente mit ihren Schnittstellen und Operationen
Synthese und Konfiguration	Informationen zu Art und Ausprägung der Verwendung und Konfiguration
Implementierung	Implementierungsspezifische Hinweise, wie zum Beispiel Speicher-/ Zeitkomplexität, Lieferformen, Hardware-Anforderungen

Testberichte	Testdaten und Testergebnisse [AF93]

Tab. 2.2: Informationen zu Komponenten (nach [Bi95])

Diese Fülle an Informationen muß vom Nutzer bei Bedarf abrufbar sein. Während Name und Synonyme zur Suche dienen, sollen Absicht und Motivation die Komponente beschreiben, so daß ersichtlich wird, was die Komponente leistet und was nicht. Soll die Komponente eingesetzt werden, so sind Informationen zu Konsequenzen, Anwendbarkeit, Implementierung und Teilnehmern nötig.
Testdaten und -berichte sowie Referenzen dienen der Motivation zur Wiederverwendung.
Entsprechend sollten die Informationen zu den jeweiligen Zeitpunkten abrufbar sein, um eine schnelle Information des Benutzers zu gewährleisten. Werden Informationen zu einer Komponente abgerufen, so sollen dem Nutzer ausschließlich benötigte Informationen zur Verfügung gestellt werden [Gö93].

Insbesondere der Komponentenbeschreibung kommt eine zentrale Bedeutung zu, da anhand der Beschreibung entschieden wird, ob die Komponente für die Anforderung relevant ist oder nicht.
"Da wiederverwendbare Module aufgrund ihrer Spezifikation erkannt und eingebaut werden, muß die Funktionalität und die Schnittstelle des Moduls exakt und granular definiert werden. So sollen neben der Definition der Schnittstelle auch die Definition der Einschränkungen (hinsichtlich des Gebrauchs und der Implementierung) enthalten sein." [Kü94]

c) Einbindung der Komponente in eine Anwendung

Ist die zu verwendende Komponente gefunden, so sollte sie mit minimalem Aufwand aus dem Repository in die Anwendungsumgebung eingebunden werden können. Das "Wie?" hängt dabei von der Art der Komponente und der Einbindung ab (siehe Abschnitt 2.2.4).

2.2.3 Erstellen von Komponenten

Grundvoraussetzung der Anwendungsentwicklung aus Komponenten ist das Vorhandensein einer Bibliothek mit zahlreichen verfügbaren Komponenten. Um Komponenten zu erhalten gibt es die drei in Abbildung 2.6 dargestellten Wege [AF93]:

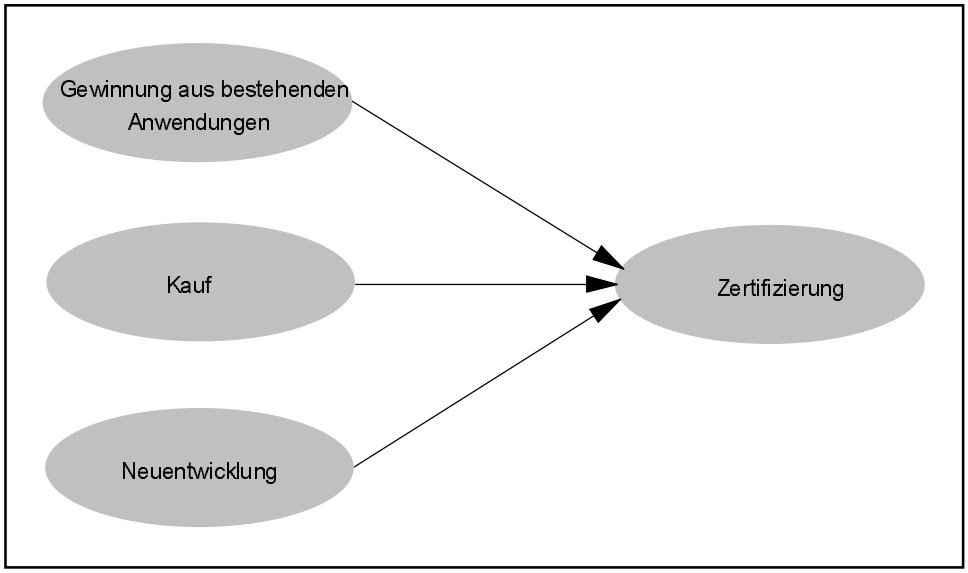

Abb. 2.6: Drei Wege zum Erhalten von Komponenten (nach [AF93])

Um eine effektive Wiederverwendung zu gewährleisten, sind unabhängig von der Art der Entstehung der Komponente Korrektheit und Fehlerfreiheit zu überprüfen, eine Zertifizierung der geprüften Komponenten fördert deren Akzeptanz. Die Überprüfung muß kontextunabhängig erfolgen, da die Einsetzbarkeit nicht auf bestimmte Kontexte beschränkt sein soll.

a) Re-Engineering für Wiederverwendung

Ein hohes Potential wiederverwendbarer Bausteine beinhalten die bestehenden Anwendungen [Ar93]. Aufgrund der Komplexität der Anwendungen ist eine sofortige komplette Ablösung unrealistisch [You93], es müssen Möglichkeiten zur Weiterverwendung und Überarbeitung der bestehenden Software erarbeitet werden. Der anfängliche Mangel an wiederverwendbaren Komponenten kann durch Re-Engineering vorhandener Software

behoben werden, die gleichzeitige Nutzung vergangener Investitionen und bestehenden Wissens senkt das Entwicklungsrisiko [Ar93], [AF93].

Wegen der vielfach falschen Benutzung der Begriffe Re-Engineering, Reverse-Engineering und Re-Strukturierung soll hier eine kurze Unterscheidung (nach [Ba96]) gegeben werden.
Unter *Re-Strukturierung* versteht man Maßnahmen zur Reorganisation von Programmen, so daß sie den Regeln der strukturierten Programmierung entsprechen [You93]. Beispiel für Re-Strukturierung ist das Entfernen von GOTO-Anweisungen im Quelltext [Ba96].
Das *Reverse-Engineering* führt zur Identifikation der einzelnen Komponenten eines Software-Systems und ihrer Beziehungen durch eine methodische Analyse. Ziele des Reverse-Engineering sind Förderung des Verständnisses des Systems, Identifizieren und Extrahieren wiederverwendbarer Komponenten oder auch die Wiedergewinnung des vollständigen Designs der Anwendung (Design-Recovery).
Unter *Re-Design* versteht man Veränderungen eines Software-Dokuments, die die Funktionalität der Applikation nicht verändern. Ein Beispiel für Re-Design ist die Modularisierung von Programmen.
Das *Software-Reengineering* beinhaltet die Anwendung von Reverse-Engineering oder Design-Recovery, gegebenenfalls gefolgt von Re-Strukturierung und Re-Design. Software-Reengineering umfaßt alle Aktivitäten, die auf die ursprünglich nicht geplante Wiederverwendung eines vorhandenen Software-Systems oder von Teilen dieses Systems gerichtet sind.

Bei der Wiederverwendung von Teilen bestehender Altanwendungen kommt dem Re-Engineering enorme Bedeutung zu [You93].
Die bestehende Software wird analysiert, um Bestandteile zu finden, die für neue Anwendungen verwendet werden können (siehe Abbildung 2.7). Eine unkontrollierte Übernahme alter Programmteile in neue Anwendungen ist problematisch, als wiederverwendbar befundene Teile müssen "transformiert" werden, um besser wiederverwendbar zu sein [AF93]. Im Ergebnis müssen sie dem Verständnis von Komponenten im jeweiligen

Unternehmen entsprechen, d.h. sie müssen getestet, mit den notwendigen Zusatzinformationen erweitert und zertifiziert werden.

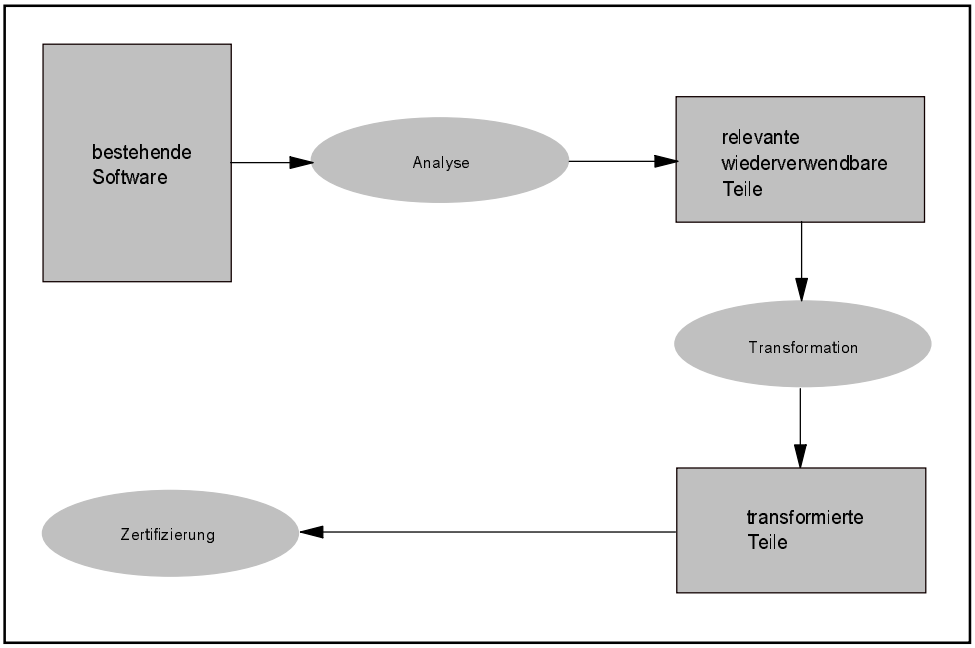

Abb. 2.7: Re-Engineering für Wiederverwendung (nach [AF93])

Die Wiederverwendung von bestehender, nicht für Wiederverwendung entwickelter Software beschränkt sich nicht auf Systemteile, auch ganze Anwendungen können weiter genutzt werden. Hierzu werden sogenannte "wrapper" eingesetzt, mit deren Hilfe die Altanwendung ("legacy system") lokal oder mittels eines Remote-Systems aufgerufen werden. Ein "wrapper" stellt eine saubere (standardisierte) Schnittstelle zu einer Altanwendung zur Verfügung, über die neue Anwendungen auf die bestehende Anwendung zugreifen. Der "wrapper" gewährleistet dabei Datenkonvertierung, das Generieren benötigter Nachrichten und Meldungen, implementiert Sicherheitsmechanismen und bei Bedarf Maßnahmen zur Transaktionskontrolle.

Ein Beispiel für den Einsatz eines "wrapper" findet man bei der Nutzung von Terminal-Anwendungen aus dem Großrechnerumfeld in GUI-Applikationen im PC-Umfeld. Dabei simuliert der "wrapper" gegenüber der Host-Anwendung die Dateneingabe in einer 3270-Emulation, für den Nutzer wird ein Fenstersystem zur Dateneingabe bereitgestellt.

Somit kann beispielsweise eine Großrechneranwendung zur Kundenverwaltung von einer PC-Applikation genutzt werden.

Folgende Abbildung zeigt das "Verpacken" eines Systems mit Hilfe eines "wrapper":

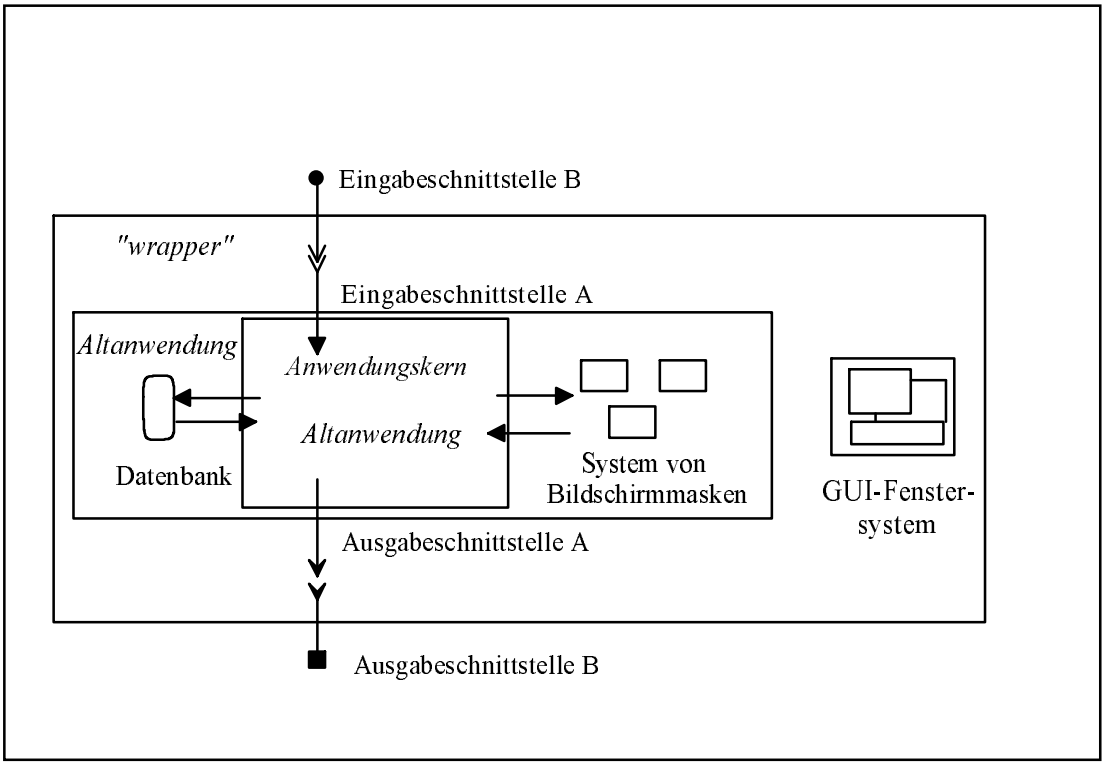

Abb. 2.8: "wrapping" einer Terminal-basierten Anwendung zu einer GUI-Anwendung

Die Eingaben der Eingabeschnittstelle B des "wrapper" werden so konvertiert, daß sie den erwarteten Eingaben der Eingabeschnittstelle A der Anwendungen entsprechen. Analog werden die Ausgaben der Anwendung konvertiert, so daß sie dem Ausgabeformat des "wrapper" entsprechen. Das System von Bildschirmmasken wird gegenüber dem Nutzer verborgen, stattdessen übernimmt ein System von Fenstern die Interaktion. Die in den Fenstern eingegebenen Werte oder ausgelösten Aktionen werden durch Funktionen des "wrapper" auf die Datenstrukturen und die Funktionalität der Altanwendung abgebildet.

b) Neuentwicklung

Bevor eine Komponente neu entwickelt wird, sollte zum einen das Vorhandensein innerhalb des Unternehmens untersucht werden. Häufig würde die Anpassung oder Erweiterung bestehender ähnlicher Programme weniger Aufwand erfordern als eine vollständige Neuentwicklung. Weiterhin empfiehlt es sich zu überprüfen, ob nicht Komponenten außerhalb des Unternehmens verfügbar sind, die die gewünschten Aufgaben erfüllen.

Erst wenn man zu dem Schluß gekommen ist, daß die Neuentwicklung die effektivste Möglichkeit darstellt, sollte man die Komponente neu entwickeln. Die Entwicklung sollte sich nicht auf die spezifizierte Anforderung beschränken, sondern auch mögliche spätere Anforderungen berücksichtigen (Abgeschlossenheit), insbesondere ist die Komponente kontextunabhängig zu erstellen, um einen späteren Einsatz in anderen Kontexten zu ermöglichen. Dazu muß die Komponente insbesondere die Anforderungen aus Abschnitt 2.1.2 Definiertheit, lose Kopplung, Kapselung, Zuverlässigkeit und Portierbarkeit erfüllen.

2.2.4 Zusammenfügen von Komponenten

Das Zusammenfügen der Komponenten zu einer Anwendung kann zu verschiedenen Zeitpunkten erfolgen. Dabei kann man nach der Art und Weise statisches und dynamisches Einbinden der Komponenten unterscheiden.

a) Statisches Zusammenfügen der Komponenten während der Entwicklung

Die statische Einbindung der Komponenten in die Anwendung erfolgt entweder durch Einfügen der Implementierungen in die Anwendung vor der Übersetzung des Quellcodes in die ausführbaren Dateien oder durch Einbinden während der Übersetzung [TI95]. Dabei wird durch letztere Möglichkeit das Prinzip des "Information Hiding" vollständig unterstützt. Die Komponenten sind fest mit der Anwendung verbunden, eine Änderung der Implementierung der Komponente oder die Ersetzung einer Komponente durch eine andere erfordern die erneute Übersetzung der Anwendung.

b) Dynamisches Zusammenfügen der Komponenten zur Laufzeit

Eine flexibleres Verfahren stellt die dynamische Einbindung der Komponenten zur Laufzeit dar. Dabei werden verfügbare Dienste der Komponenten zur Zeit der Ausführung der Anwendung bei Bedarf aufgerufen und verwaltet [TI95]. Vorteil dieses Verfahrens ist die einfache Ersetzbarkeit der Komponenten ohne Auswirkungen auf die eigentliche Anwendung. Lediglich die Verfügbarkeit einer Komponente, die den benötigten Dienst anbietet, ist zu gewährleisten.

Wird der Dienst der Komponente direkt aufgerufen (analog dem einfachen Methodenaufruf in der Objektorientierung), so wird ein konkreter Dienst einer speziellen Komponente eingebunden. Diese kann folglich nur durch eine gleichnamige Komponente mit gleichnamigem Dienst ersetzt werden.

Durch Standards, wie zum Beispiel OLE ("Object Linking and Embedding") oder DDE ("Dynamic Data Exchange"), werden Möglichkeiten zur Interaktion von Komponenten unterschiedlicher Entwicklungswerkzeuge bereitgestellt.

Der Einsatz eines "Object Request Broker" (ORB) ermöglicht darüber hinaus die Trennung der Kommunikationsmechanismen von der eigentlichen Anwendung [MO94]. Der ORB gewährleistet den transparenten Aufruf des benötigten Dienstes einschließlich der Kommunikation und eventueller Konvertierungen (siehe Abb. 2.9).

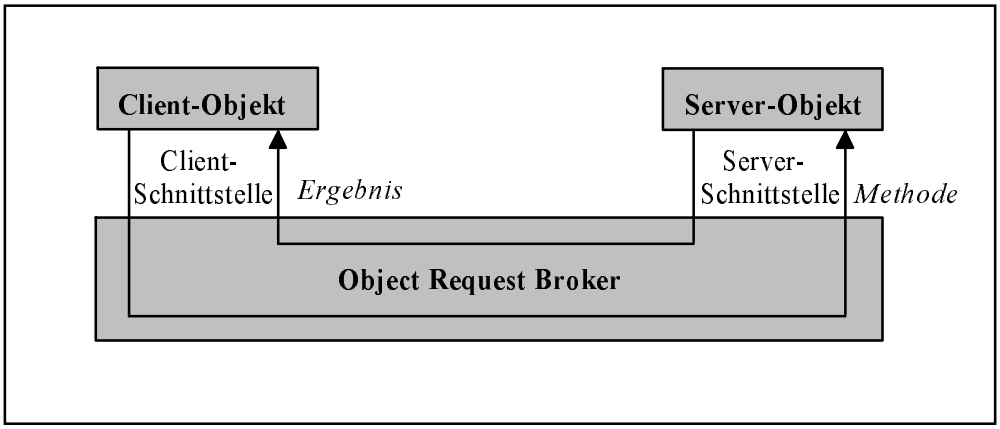

Abb. 2.9: Funktionsweise eines ORB (nach [MO94])

Das Architekturmodell der OMG weist einem ORB folgende Aufgaben zu (nach [MO94]):

- Name Service - gewährleistet die Abbildung der Objektnamen aus der Domäne des Nutzers (Client) in die Domäne des Dienstanbieters (Server) und umgekehrt,
- Request Dispatch - bestimmt, welcher konkrete Aufruf auf eine Dienstanforderung erfolgen soll,
- Parameter-Codierung,
- Ablieferung von Anfrage und Ergebnissen,
- Synchronisation
- Aktivierung persistenter Objekte
- Ausnahmebehandlung
- Sicherheitsmechanismen

Durch einen ORB können zur Laufzeit unterschiedliche Komponenten eingebunden werden, die einen angeforderten Dienst anbieten. Es existieren inzwischen verschiedene Standards für ORB, die nur zum Teil kompatibel sind. Dabei kommt der "Common Object Request Broker Architecture" (CORBA) der OMG die wohl größte Bedeutung zu. Deshalb soll diese hier kurz betrachtet werden.

Der zentrale Bestandteil von CORBA ist eine "Interface Definition Language" (IDL), also eine Schnittstellendefinitionssprache [MO94]. In der IDL werden die Schnittstellen zum Methodenaufruf sprachneutral spezifiziert. Mit Hilfe von IDL-Compilern werden dann sogenannte STUBS für die Client- und SKELETONS für die Server-Seite generiert (Abb. 2.10). Über diesen Code und die Laufzeitumgebung der ORB erfolgt dann der eigentliche Methodenaufruf [MO94] (siehe Abbildung 2.11).

Weiterhin werden aus der IDL-Spezifikation Informationen sowohl für das Interface Repository (für Abfragen von Informationen über die Schnittstellen) als auch Informationen für das Implementation Repository generiert (Informationen zum dynamischen Aufruf ohne Stubs und Skeletons).

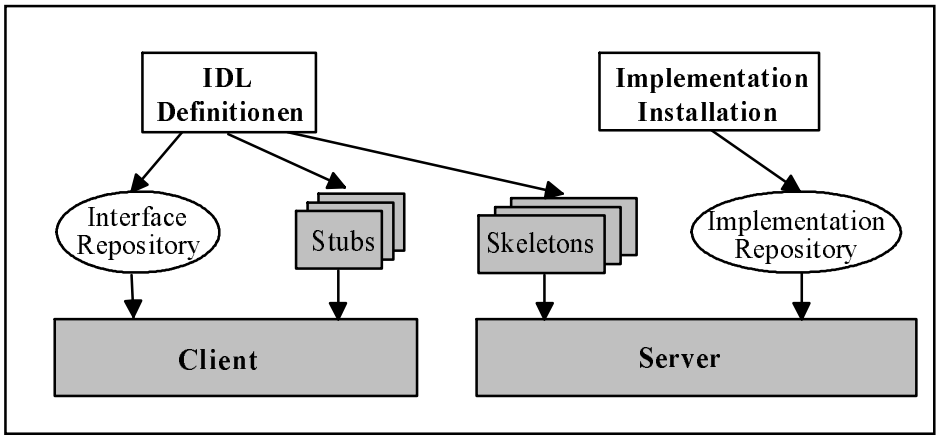

Abb. 2.10: Rolle der CORBA-IDL (nach [MO94])

Der Methodenaufruf kann entweder über die generierten Stubs und Skeletons oder dynamisch nach Abfrage eines ORB Service Interface erfolgen (siehe Abbildung 2.11):

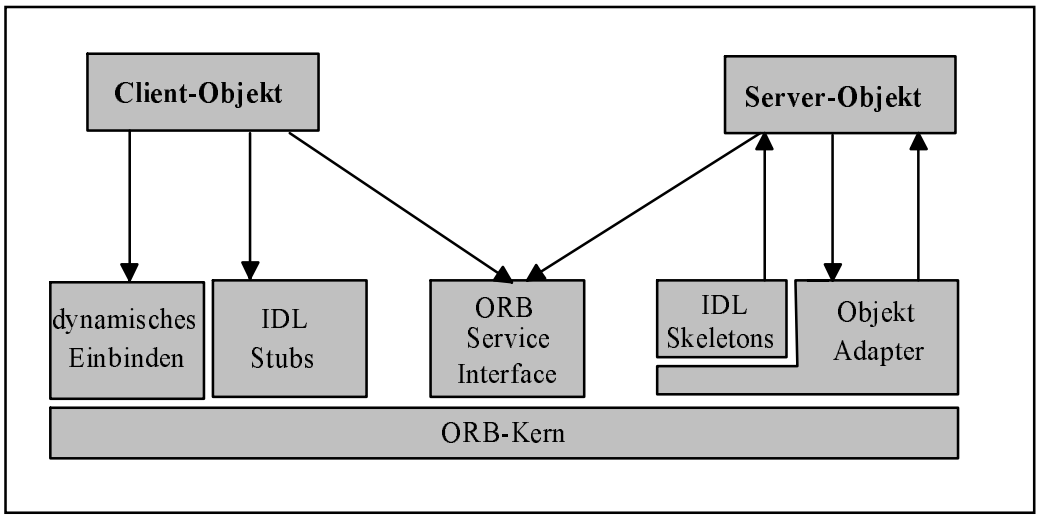

Abb. 2.11: Methodenaufruf bei CORBA (nach [MO94])

Zur CORBA-Spezifikation existieren inzwischen einige Implementierungen, wie DSOM ("Distributed System Object Model") von IBM [MO94].
Art und Weise der Einbindung der Komponenten hängen von den Möglichkeiten der Software-Produktionsumgebung (SPU) ab. Wird mit der Einführung komponentenbasierter Entwicklung auch eine neue Entwicklungsumgebung eingeführt, so ist darauf zu achten, daß ein möglichst breites Spektrum an Standards berücksichtigt ist.

Insbesondere die mögliche Einbindung von Bausteinen anderer Entwicklungswerkzeuge z.B. über CORBA, OLE oder DDE stellt eine wichtige Anforderung an eine SPU dar.

2.2.5 Schlußfolgerungen für das Life-Cycle-Modell

Die Entwicklung von Anwendungen aus Komponenten erfordert neue Modelle für den Lebenszyklus einer Anwendung.
In Life-Cycle-Modellen, wie dem einfachen Software-Life-Cycle oder dem Wasserfallmodell werden keinerlei Aussagen zur Einbindung von Wiederverwendungsaktivitäten getroffen. Geplante Wiederverwendung muß jedoch in das Modell der Software-Erstellung integriert werden.
a) Life-Cycle-Modell

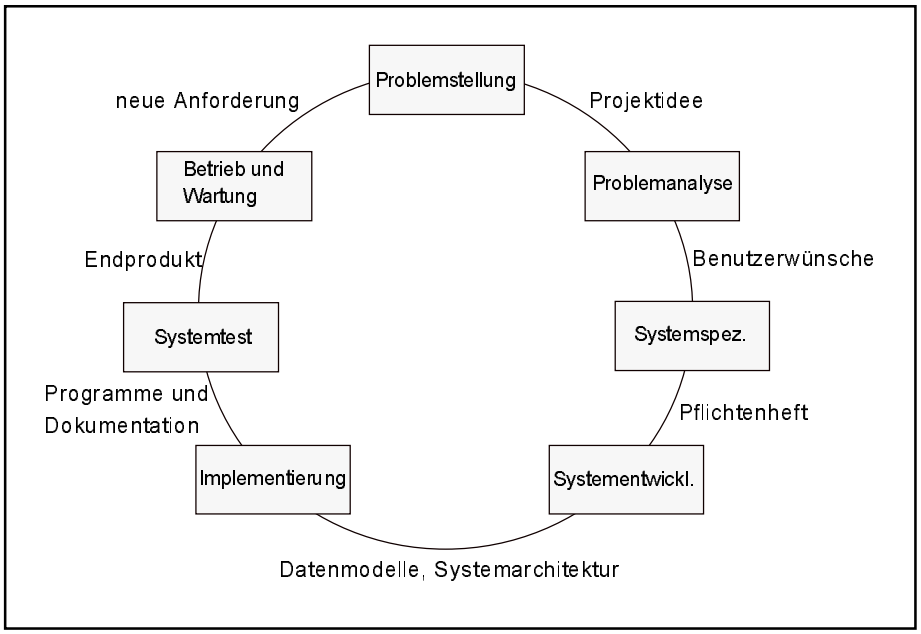

Abb. 2.12: Software-Life-Cycle (nach [PB93])

Das Life-Cycle-Modell (Abbildung 2.12) zeigt Unzulänglichkeiten in Bezug auf die streng sequentielle Phasenfolge (kein iteratives Vorgehen abbildbar) sowie das Fehlen von Wiederverwendungsaspekten (keine Einbindung bestehender Elemente) [PB93]. Aus diesem Grund erfolgt eine Erweiterung, in deren Ergebnis das Wasserfallmodell entsteht.

b) Wasserfallmodell

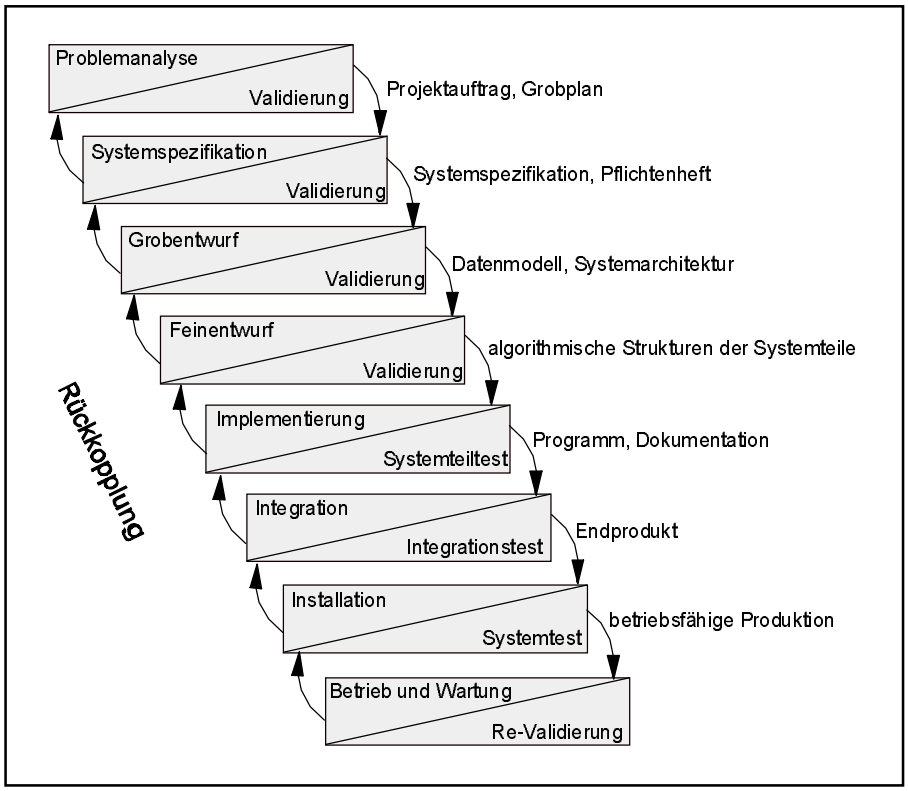

Abb. 2.13: Wasserfallmodell (nach [PB93])

Das in Abbildung 2.13 dargestellte Wasserfallmodell berücksichtigt die Überschneidung der Entwicklungsphasen, es wurde ein Wissenstransfer in vorgelagerte Phasen berücksichtigt. Somit ist ein iteratives Vorgehen bei der Software-Entwicklung möglich. Zur Wiederverwendung werden jedoch weiterhin keine Aussagen gemacht [PB93]. Deshalb ist eine weitere Erweiterung nötig. Diese soll speziell für die komponentenbasierte Anwendungsentwicklung gemacht werden.

c) Komponentenmodell

Das Komponentenmodell soll die Struktur der Anwendungsentwicklung wiedergeben, bei der eine Trennung der Entwicklung des Anwendungsrahmens und der eingebetteten Komponenten erfolgt (siehe Abbildung 2.14).

Dabei erfolgt die Komponentenentwicklung in zwei getrennten Schritten. Die Phase Verkauf/ Organisation übernimmt Koordination und strategische Planung der Komponentenentwicklung, die Komponenten werden vollständig spezifiziert. Dieser Vorgang erfolgt

synchron zu den Anforderungen der Anwendungsentwicklung. Zu den erstellten Spezifikationen werden in der Produktionsphase die Implementierungen erzeugt. Dies geschieht ausschließlich im Auftrag von Verkauf/ Organisation und somit unabhängig von der Anwendungsentwicklung. Insbesondere werden strategisch geplante Komponenten ohne spezielle Anforderung einer Anwendungsentwicklung erstellt (asynchron) [CB93].

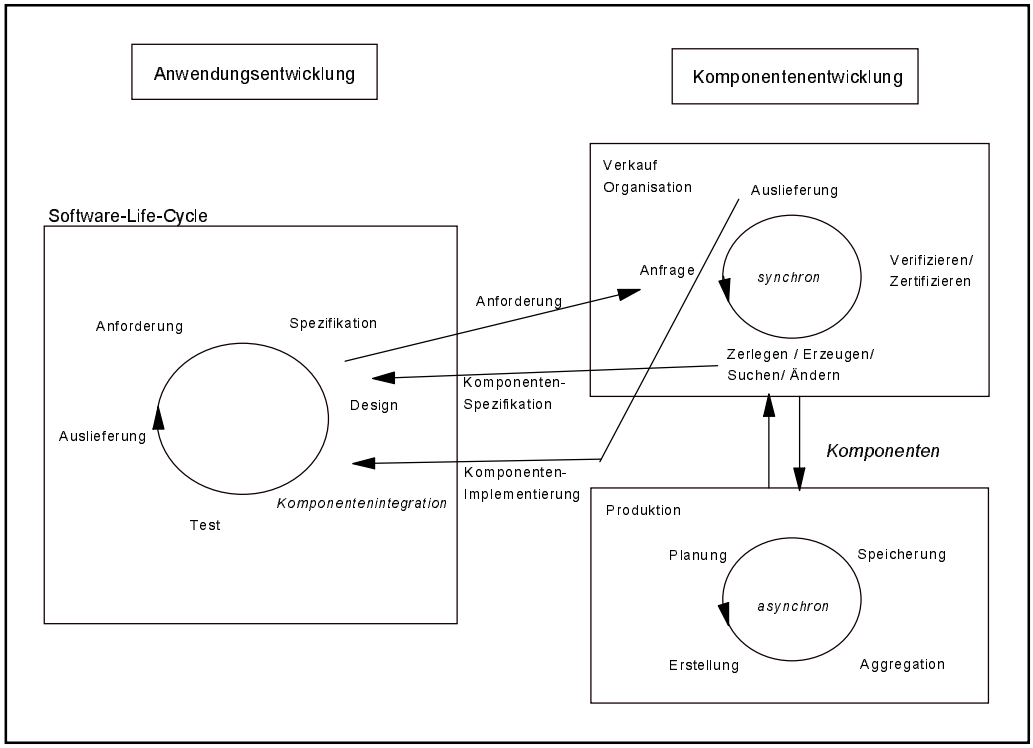

Abb. 2.14: Komponentenmodell (nach [CB93])

Das Modell schließt die Erstellung, Suche, Änderung und Wiederverwendung der Komponenten in den Software-Entwicklungsprozeß ein.

Im in Abbildung 2.14 dargestellten Modell sind die einzelnen Entwicklungsmodelle aus Übersichtsgründen vereinfacht als Life-Cycle dargestellt. Diese Teilmodelle sollen nun weiter spezialisiert werden.
Die Anwendungsentwicklung erfolgt nach einem abgewandelten Wasserfallmodell (Abbildung 2.15), in dem die Verbindung zur Komponentenentwicklung zum Ausdruck kommt.

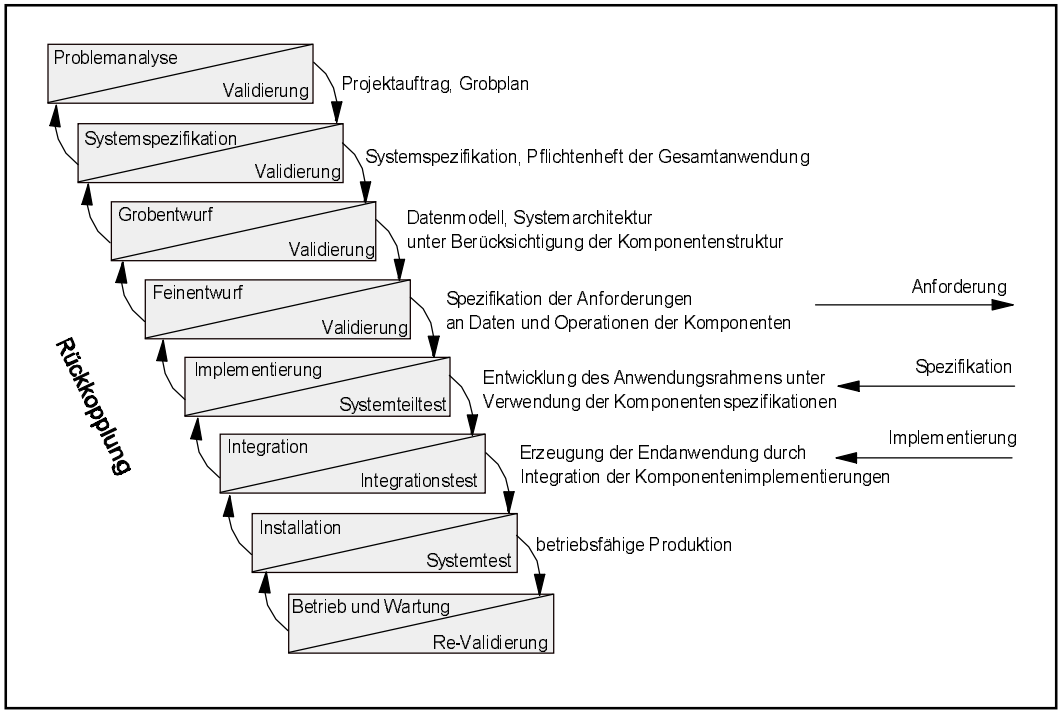

Abb. 2.15: Abgewandeltes Wasserfallmodell

In das Wasserfallmodell (Abbildung 2.11) wurden Übergänge zur Komponentenentwicklung eingefügt, über die die Anforderungen sowie die resultierenden Spezifikationen und Implementierungen ausgetauscht werden (Abbildung 2.15).

Die Erstellung der Komponenten erfolgt nach dem sogenannten abgewandelten Fontainen-Modell.

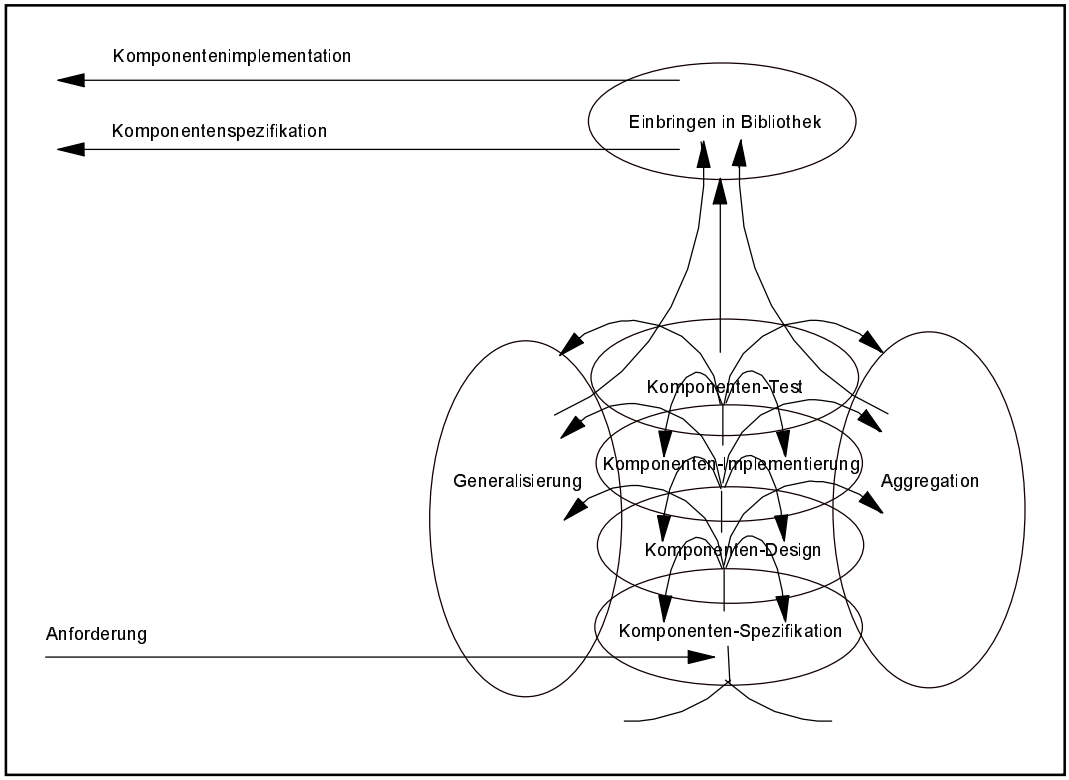

Abb. 2.16: Abgewandeltes Fontainen-Modell

Aus Abbildung 2.16 ist ersichtlich, daß sowohl ein Informationsfluß in frühere Entwicklungsphasen als auch die Speicherung von Entwicklungsergebnissen und -erfahrungen im Hinblick auf erneute Anwendung berücksichtigt wird. Dabei werden Ergebnisse aller Entwicklungsphasen über Aggregation und Generalisierung aufgearbeitet und für spätere Wiederverwendung bereitgestellt.

2.3 Voraussetzungen

Um Wiederverwendung erfolgreich umsetzen zu können, müssen eine Reihe von Voraussetzungen geschaffen werden. Insbesondere müssen Maßnahmen getroffen werden, um die in Abschnitt 1.3 genannten Hindernisse zu überwinden.

2.3.1 Technische Voraussetzungen

Grundvoraussetzung für Wiederverwendung ist eine klare Strategie, "Was?" und "Wie?" wiederverwendet werden soll. Dazu sollte eine *Analyse des Geschäftsbereiches* durchgeführt werden, in deren Ergebnis die Kernelemente der *Anwendungsarchitektur* stehen. Diese stellen erfolgversprechende Kandidaten zur Ausarbeitung als Komponenten dar.
Weiterhin müssen die *Möglichkeiten und Grenzen vorhandener Entwicklungswerkzeuge* untersucht werden. Herkömmliche Software-Produktionsumgebungen unterstützen meist ausschließlich die Entwicklung und Wartung von Anwendungen. Wiederverwendungsorientierte Software-Entwicklung erfordert jedoch Werkzeuge, die es gestatten, Systeme wiederverwendbarer Bausteine durch Import aus anderen Software-Produktionsumgebungen und durch Eigenentwicklungen aufzubauen und zu warten. Ebenso müssen sie die Montage von Anwendungen aus Bausteinen unterstützen [Li96]. Grundvoraussetzung für eine erfolgreiche Wiederverwendung ist ein leistungsfähiges *Repository* (siehe Abschnitt 2.2.2). Die nahtlose Einbindung der Wiederverwendung in die bestehende Software-Entwicklungsumgebung, insbesondere die Integration neuer Werkzeuge zur optimalen Unterstützung der Wiederverwendungsaktivitäten, ist unabdingbar. Andererseits sollen verbreitete *Standards*, wie COM, CORBA oder ACTIVE-X unterstützt werden, um Wiederverwendung fremder Komponenten im eigenen Unternehmen sowie Wiederverwendung eigener Komponenten in anderen Unternehmen zu ermöglichen.
Dem Mangel an verfügbaren Komponenten steht das große Potential bestehender Anwendungen entgegen, eine Analyse der Software mit anschließendem *Re-Engineering* kann wichtige Bausteine liefern. Außerdem müssen externe Quellen von wiederverwendbaren Komponenten erschlossen werden. Dazu leisten Partnerschaften zu anderen Unternehmen mit ähnlicher Anwendungsarchitektur einen wichtigen Beitrag.

Mit der Berücksichtigung bzw. Initiierung *branchenweiter Vereinbarungen* sowie mit der Umsetzung bestehender Standards werden die nötigen Voraussetzungen für eine Wiederverwendung über Unternehmensgrenzen hinweg geschaffen.

2.3.2 Nichttechnische Voraussetzungen

Die Überwindung der nichttechnischen Hindernisse, insbesondere der soziologische Hemmnisse, ist entscheidend für den Erfolg der Wiederverwendung.
Um die Akzeptanz der Mitarbeiter im Hinblick auf die Wiederverwendung von Ergebnissen Dritter zu erhöhen, müssen geeignete Maßnahmen getroffen werden. Beispielsweise kann über *Belohnungssysteme* Wiederverwendung bestehender Software und Erstellung wiederverwendbarer Software honoriert werden. Das *Kostenbewußtsein* der Entwickler muß geschult werden, der *Nutzen von Standards* und Kompromissen muß verdeutlicht werden.
Zur Bekämpfung des "Not Invented Here"-Syndroms ist die Einhaltung definierter *Qualitätsmaßstäbe* für Komponenten wichtig. Das Verfahren der komponentenbasierten Anwendungsentwicklung darf weiterhin nicht als Allheilmittel propagiert werden, vielmehr müssen *Vor- und Nachteile* von vornherein deutlich gemacht werden. Die *schrittweise Einbindung* in den bestehenden Entwicklungsprozeß vermeidet einen zu großen Bruch, so daß sich das neue Verfahren allmählich etablieren kann.
Die Klärung der ökonomischen Aspekte der Wiederverwendung von Komponenten muß vor der Einführung dieses Vorgehens erfolgen. Mit Anwendungsentwicklung aus Komponenten wird von arbeitsintensiver Projektkultur zu *kapitalintensiver Komponentenkultur* übergegangen [Qu94]. Die damit verbundenen Kosten müssen bei der Entscheidung für Wiederverwendung berücksichtigt werden, um spätere Diskussionen über *Kosten/ Nutzen-Aspekte* zu vermeiden.
Die Umstellung der Organisation auf Anwendungsentwicklung aus Komponenten erfordert einige *Veränderungen im Vorgehensmodell*, in der *Struktur der Mitarbeiterorganisation* und in der *Planung* der Anwendungsentwicklung in Abhängigkeit der bestehenden Voraussetzungen.
Wiederverwendung erfordert ein neues Vorgehen bei der Software-Erstellung. Dieses neue Vorgehen muß sich auch in den organisatorischen Strukturen wiederfinden, insbesondere sind neue Stellen zu schaffen, die Wiederverwendungsaktivitäten planen, leiten,

koordinieren und durchführen. Die Besetzung dieser Stelle sollte mit Mitarbeitern erfolgen, die über die Fähigkeit verfügen, langfristig zu planen und global zu denken, um den Erfolg der Wiederverwendung zu gewährleisten.

Entsprechend dem Vorgehensmodell in 2.2.5 und der allgemeinen Prinzipien in 2.2 zerfallen die herkömmlichen Projektstrukturen in Anwendungsentwicklung und Komponentenerstellung. Eine zentrale Rolle kommt der Schnittstelle dieser beiden Strukturen zu. Der Bereitstellung existierender Komponenten zum einen und der Planung neuer Komponenten zum anderen kommt eine entscheidende Bedeutung für den Erfolg der Wiederverwendung zu.

Im Hinblick auf rechtliche Problematiken müssen die Grundlagen zu vertraglichen Vereinbarungen bei Wiederverwendung von Komponenten Dritter geschaffen werden.

Weiterhin ist festzulegen, zu welchen Bedingungen welche Informationen zu Komponenten von Dritten abgerufen werden können, ohne daß der Schutz der in die Komponenten geleisteten Investition verlorengeht.

2.4 Einführung von komponentenbasierter Anwendungsentwicklung

Die Einführung der Wiederverwendung muß geplant erfolgen. Dabei sind insbesondere der Stand der Software-Entwicklung und die Möglichkeiten der bestehenden Software-Produktionsumgebung zu berücksichtigen. Ein Übergang in einem Schritt ist unrealistisch, da eine Reihe von organisatorischen und strukturellen Veränderungen durchgeführt werden müssen, die auch aufgrund der Hindernisse aus Abschnitt 1.3 Risiken bergen. Für den schrittweisen Übergang hin zu einer unternehmensweiten Wiederverwendungskultur gibt es verschiedene Wege.

[Re95] schlägt einen Übergang in drei Schritten vor. Ausgehend von Ad Hoc-Wiederverwendung wird im ersten Schritt die Wiederverwendung bestehender Software systematisiert.

2.4.1 1. Schritt: Systematisierung der Wiederverwendung verfügbarer Software

Eine zu schaffende Voraussetzung im ersten Schritt ist die *modulare Software-Erstellung*. Dies schließt die Erstellung modularer Anwendung sowie das Auswählen, Verstehen und Anpassen vorhandener Module ein [Re95].

Rezagholi versteht unter Modulen Software-Einheiten mit definierten Schnittstellen und bestimmten Funktionalitäten. Somit stellen Komponenten spezielle Module dar. Unterstützende Techniken sind strukturierte Methoden für Analyse und Design, Schnittstellendefinitionen, Datenabstraktionen und Modularisierungsrichtlinien [Re95]. Im ersten Schritt werden neben der Einführung marktüblicher *Standards* (zum Beispiel VAA im Versicherungsbereich) auch unternehmensspezifische Festlegungen, z.B. hinsichtlich Namenskonventionen, Strukturnormen, getroffen, die zur Vereinheitlichung des Vorgehens führen.

Als weitere Voraussetzung wird eine Katalogisierung wiederverwendbarer Komponenten eingeführt [Re95]. Darin werden alle Komponenten erfaßt und dokumentiert.

Der ersten Schritt betrifft nur die einzelnen Projekte, deren Vorgehen bei der Entwicklung wird auf Modularisierung der Strukturen und Publikation der Ergebnisse ausgerichtet. Infolge der Modularisierung werden Ergebnisse innerhalb des Projektes wiederverwendet. Da die erzeugten Ergebnisse projektorientiert sind, ist eine

Wiederverwendbarkeit in anderen Projekten unwahrscheinlich. Deshalb wird in Schritt 2 die Sichtweise von einzelnen Projekten auf Anwendungsfelder ausgeweitet.

2.4.2 2. Schritt: Ausrichtung der Software-Erstellung auf Anwendungsdomänen

Ziel von Schritt 2 ist es, Software-Komponenten nicht auf die Verwendung in einer Anwendung, sondern im Hinblick auf ihren künftigen Einsatz in allen Produkten innerhalb einer *Anwendungsdomäne* zu entwickeln [Re95]. Unter einer Anwendungsdomäne versteht man einen Unternehmensbereich oder eine Sparte [BFB95], sie ist gekennzeichnet durch ähnliche Anwendungsstrukturen. Im zweiten Schritt werden eine Reihe von organisatorischen Maßnahmen durchgeführt, um eine *Infrastruktur* für die wiederverwendungsorientierte Software-Erstellung zu schaffen [Re95]. Notwendig für die erfolgreiche Umsetzung ist der Übergang von der Einzelprojektsichtweise hin zu einer projektübergreifenden Sichtweise. Dafür ist eine projektübergreifende *Controlling- und Support*-Stelle einzurichten, die Planung, Koordination, Überwachung und Unterstützung des wiederverwendungsorientierten Erstellungsprozesses übernimmt [Re95].
Als Voraussetzung muß eine *Domänenanalyse* durchgeführt werden [BFB95], um ähnliche Teilprodukte zu extrahieren, die dann als Komponenten realisiert werden können [Re95]. Als Ergebnis entsteht ein Domänenmodell, das die generelle Architektur der Domäne festlegt und damit die Grundlage der Wiederverwendbarkeit schafft.
Die in Schritt 1 entwickelten Standards und Richtlinien werden um die Aspekte der projektübergreifenden Wiederverwendung erweitert. Es werden *Qualitätsanforderungen* für wiederverwendbare Bausteine festgelegt.
Um die Akzeptanz der Mitarbeiter gegenüber dem neuen Vorgehen zu Verbessern, werden *Schulungen* zum Schwerpunktthema Wiederverwendung durchgeführt [Re95]. Die erstellten Komponenten werden in einer projektübergreifenden Bibliothek zusammengefaßt und verfügbar gemacht.

2.4.3 3. Schritt: Kennzahlenbasiertes Management der Wiederverwendung

Ziel des 3. Schrittes ist es, die Wiederverwendung kennzahlenbasiert zu steuern [Re95]. Dazu werden *Metriken* und *Kosten/ Nutzen-Modelle* erstellt, die eine *Bewertung* der einzelnen Komponenten ermöglichen [Re95]. Aufgrund dieser Einschätzungen können

Entscheidungen zu Weiterentwicklung, Ablösung oder Kauf externen Komponenten getroffen werden.

2.5 Zusammenfassung

Die Einführung objektorientierter Software-Entwicklung gestaltet sich im betriebswirtschaftlichen Umfeld oft als zu schwierig. In diesem Zusammenhang bietet komponentenbasierte Entwicklung einen sinnvollen Zwischenschritt, der zum einen die Risiken des Übergangs von prozeßorientierter zu objektorientierter Software-Produktion minimiert und zum anderen Vorteile der Objektorientierung mit bestehenden Entwicklungswerkzeugen nutzbar macht.

Die Wiederverwendbarkeit der Komponenten wird durch Erfüllung der gestellten Anforderungen unterstützt, die Erwartungen an Wiederverwendung (Abschnitt 1.1) können durch komponentenbasierte Entwicklung in hohem Maße erfüllt werden, wenn strukturiert und organisiert vorgegangen wird.

Durch das "Zerlegen" einer Anwendung in kooperierende Komponenten kann der Entwicklungsprozeß effizienter gestaltet werden (Prinzip "Teile und Herrsche") [Gö93]. Die Komponenten können unabhängig entwickelt, getestet und verändert werden. Dadurch kann eine entscheidende Einsparung an Kommunikationsaufwand erreicht werden [Gö93]. Die teilweise Parallelisierbarkeit der Entwicklung führt zu Zeitersparnis und der Verminderung des Risikos von Fehlentwicklungen.

Die Umsetzbarkeit komponentenbasierter Entwicklung soll im folgenden Kapitel am Beispiel der Software-Entwicklung in einem Versicherungsunternehmens untersucht werden.

3. Wiederverwendung in der R+V-Versicherung

Die in den ersten beiden Abschnitten gewonnenen Erkenntnisse zu Wiederverwendung und komponentenbasierter Anwendungsentwicklung sollen im folgenden auf das spezielle Umfeld der R+V-Versicherung angewendet werden. Dazu wird ausgehend von der derzeitigen Situation im Unternehmen ein möglicher Weg aufgezeigt, Wiederverwendung bei der Software-Entwicklung geplant umzusetzen. Die für das vorgeschlagene Vorgehensmodell notwendigen Voraussetzungen werden benannt. Abschließend wird die Wiederverwendung hinsichtlich der Erwartungen und Ziele bewertet.
Mit Rücksicht auf den Umfang dieser Arbeit ist es nicht möglich, alle Aspekte der Wiederverwendung zu betrachten. Deshalb wird der Schwerpunkt auf den Prozeß der Entwicklung von Anwendungen aus Komponenten gelegt und ergänzende andere Wiederverwendungsmöglichkeiten werden nur am Rande betrachtet. Die prototypische Implementation dient der Darstellung der Prinzipien und des Vorgehens, der versicherungstechnische Aspekt wird vereinfacht dargestellt. Die Implementierung soll die Machbarkeit des vorgeschlagenen Vorgehens demonstrieren und prinzipiell darstellen, wie eine praktische Umsetzung aussieht.

3.1 Analyse der Ausgangssituation

Eine genaue Analyse des heutigen Ablaufs der Software-Entwicklung in der R+V-Versicherung hinsichtlich Strukturiertheit und Methodik stellt eine wesentliche Grundlage dar, um Wiederverwendung in den Prozeß der Software-Erstellung einzubinden.
Weiterhin müssen Möglichkeiten und Grenzen der eingesetzten Werkzeuge untersucht werden, um deren Tauglichkeit für Wiederverwendung festzustellen. Daraus sind die weiteren Schritte zur Umsetzung der Wiederverwendung abzuleiten.

3.1.1 Analyse der Systemlandschaft

In informationsverarbeitenden Unternehmen, wie sie Versicherungen darstellen, kommt der Datenverarbeitung elementare Bedeutung zu. Leistungsfähigkeit und Qualität der Anwendungssysteme beeinflussen in hohem Maße Leistungsfähigkeit und Qualität der Produkte des Unternehmens.

Als Ergebnis einer umfassenden Analyse der bestehenden Systeme im Jahr 1992 ergaben sich teilweise erschreckende Werte. Es existierten zu diesem Zeitpunkt:

- ca. 72000 technische Datenelemente
- ca. 16000 Quellcodemodule
- ca. 2000 Testdatenbanken
- ca. 1200 produktiv eingesetzte Datenbanken
- ca. 133.000 Dateien (für Test und produktiven Einsatz)

(Quelle: Abschlußbericht aus [UWM92])

Dabei fanden sich für identische Elemente der realen Welt häufig redundante Entsprechungen in den Datenmodellen. Infolgedessen waren Anforderungen, wie anwendungsübergreifende Auswertungen oder spartenübergreifende Versicherungsprodukte, nicht oder nur schwer umsetzbar.

Die Ursachen der Vielzahl redundanter Datenelemente und Funktionen lagen in der parallelen Entwicklung von mehr als 250 Mitarbeitern in der R+V-Versicherung sowie in der vom Gesetzgeber vorgeschriebenen Spartentrennung innerhalb der Versicherungsunternehmen.

Die Erkenntnisse der Analyse brachten einige Veränderungen im Vorgehen bei der Software-Entwicklung mit sich. Die momentane Vorgehensweise soll in folgenden Betrachtet werden.

3.1.2 Analyse der Vorgehensweisen bei der Software-Erstellung

In der R+V-Versicherung kann man heute bereits von einem Wiederverwendungsbewußtsein in der strategischen Planung sprechen.

Insbesondere das im Jahr 1992 verabschiedete Unternehmensmodell zeugt vom Bewußtsein, Grundlagen für Wiederverwendung schaffen zu müssen.

3.1.2.1 Das unternehmensweite Modell (UWM)

Das Unternehmensmodell entstand nach umfassender Analyse aller aktuellen und potentiellen Anforderungen an die zu verarbeitenden Informationen in der R+V-Versicherung. Im Ergebnis entstand ein abstraktes Datenmodell (genannt A-Level-Modell), in dem die Kerneinheiten des Datenmodells der R+V-Versicherung festgelegt wurden. Die dabei

getroffenen Festlegungen sind verbindlich für alle nachfolgenden Projekte. Ziel des Unternehmensmodells ist es, ein unternehmensweit einheitliches Datenmodell zu schaffen, das zum einen redundanzfrei ist und zum anderen Austauschbarkeit und Auswertbarkeit der Informationen im gesamten Unternehmen ermöglicht.

Die festgelegten Kerneinheiten *Dokument*, *Geschäftsvorfall*, *Markt*, *Objekt*, *Partner*, *Produkt*, *Vertrag* und *Wertbewegung* sollen kurz definiert werden (nach [UWM92]):

Kerneinheit	Kurzbeschreibung
Dokument	Medium der Äußerungen von Geschäftspartnern einschließlich der R+V selbst gegenüber anderen Geschäftspartnern
Geschäftsvorfall	Reihe fachlicher Aktivitäten, die durch einen Auslöser initiiert werden und zielgerichtet ein Ergebnis erzeugen
Markt	Umweltfaktoren für den Geschäftsbetrieb (volks- und versicherungswirtschaftliche Daten)
Objekt	materielle oder immaterielle Sachen oder Rechte
Partner	alle natürlichen Personen, juristischen Personen und nicht-rechtsfähigen Gruppierungen, die für R+V im Rahmen ihres Geschäftsbetriebes von Interesse sind
Produkt	alle materiellen und immateriellen Waren und Dienstleistungen, die von R+V am Markt angeboten oder von dort in Anspruch genommen werden
Vertrag	rechtlich bindende Vereinbarung zwischen Partnern, in dem die Rechte und Pflichten der Vertragsparteien festgelegt werden
Wertbewegung	monetäre und nichtmonetäre Wertflüsse

Tab. 3.1: Kerneinheiten der R+V-Versicherung (nach [UWM92])

Mit Hilfe dieser acht Kerneinheiten wird die gesamte Datenwelt der R+V-Versicherung erfaßt.

Dabei bestehen folgende Verbindungen zwischen den Kerneinheiten:

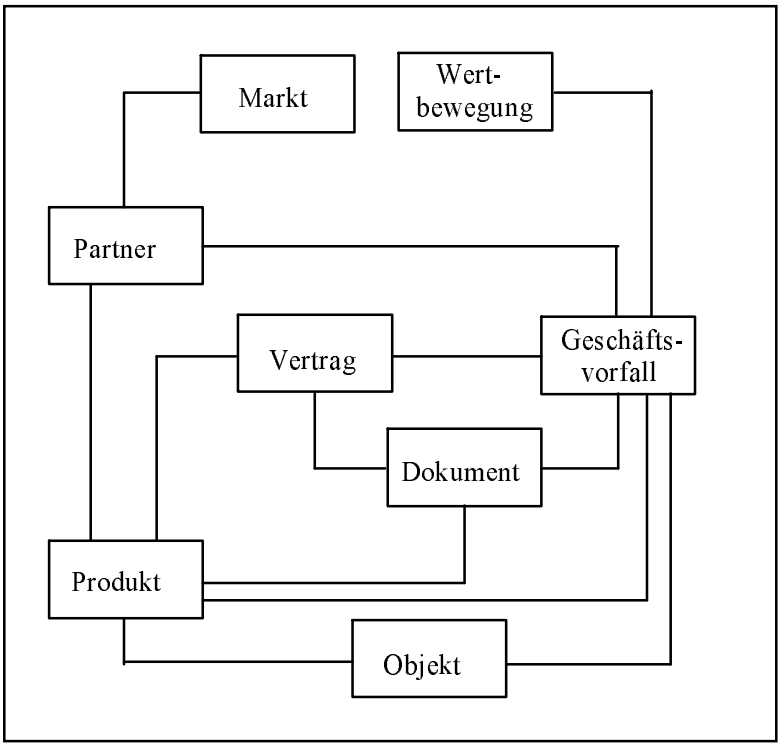

Abb. 3.1: Verbindungen zwischen den Kerneinheiten (nach [UWM92])

Die Verbindungen in Abbildung 3.1 sind wie folgt zu lesen:
Ein Produkt (z.B. Kfz-Haftpflichtversicherung) kann ein Objekt versichern (z.B. Kfz), zu dem Produkt existieren Dokumente (z.B. Produktbeschreibungen), das Produkt wird von Partnern (z.B. Versicherungsnehmer) in Anspruch genommen durch einen Vertrag (z.B. Kfz-Rundumschutz). Ein Geschäftsvorfall (z.B. Schadenmeldung) bewirkt Änderungen des Produktes (z.B. Änderung der Schadensfreiheitsrabatte).

Das [UWM92] spezifiziert den strukturellen Aufbau jeder Kerneinheit weiter, die Ergebnisse wurden als konzeptionelles Datenmodell (B-Level-Modell) festgelegt. Das B-Level-Modell legt zum einen Entities und deren Beziehungen fest, zum anderen wurden Attribute und deren Eigenschaften verbindlich vereinbart. Insbesondere muß jedes Projekt auf Grundlage der bereits bestehenden Datenwelt aus früheren Projekten

weitergeführt werden (siehe auch Abschnitt 3.1.2.3). Damit werden redundante Datenstrukturen vermieden.
Das beschriebene Unternehmensdatenmodell stellt eine sehr gute Basis für die Entwicklung von Komponenten unabhängig von deren Anwendung dar.
Das in der R+V-Versicherung entwickelte Unternehmensmodell ist konform zum Standard der deutschen Versicherungswirtschaft VAA [GDV96]. Durch aktive Mitarbeit bei der Entwicklung dieser Branchenstandards wurden die Voraussetzungen geschaffen, um Wiederverwendung zwischen den Versicherungsunternehmen zu ermöglichen.

3.1.2.2 Projektstruktur der R+V-Versicherung

Die Erkenntnisse der Untersuchung zum Unternehmensdatenmodell spiegeln sich auch in der Projektstruktur wider. Die Entwicklung der Software für den PC-Bereich beinhaltet zum Beispiel die Projekte *Produktkern* und *Adresse/ Partner*, die auf den Kernentitäten basieren. Ergebnisse dieser Projekte werden in Anwendungsprojekten, wie zum Beispiel *KFZ/ Rechtsschutz*, *Unfall*, *Finanzierung* und *Agentursystem* verwendet. Die Zuständigkeiten für Änderungen oder Erweiterungen von Aspekten des Bereiches Produktkern werden ausschließlich durch das Projekt Produktkern ausgeführt. Somit wurde eine teilweise Trennung der Anwendungsentwicklung von der Entwicklung einzelner Systemteile eingeführt.
Die Ergebnisse der Projekte zu den Kerneinheiten werden mehrfach (in unterschiedlichen Anwendungen) genutzt.

3.1.2.3 Wiederverwendung in der R+V-Versicherung

a) Wiederverwendung von Funktionen
Die Möglichkeiten der Wiederverwendung werden in der R+V-Versicherung nicht voll genutzt. Es existieren Ansätze, wie das "Zentrale Funktionenmodell". Darin sind häufig benutzte Funktionalitäten abgelegt, die von den verschiedenen Projekten wiederverwendet werden. Momentan werden darüber hinaus 155 fachliche Funktionen auf Basis der oben beschriebenen Projektstruktur in unterschiedlichen Anwendungen genutzt.
Dabei treten eine Reihe von Schwierigkeiten auf. Zum einen existieren keine vollständigen Richtlinien und Vorgaben zur Wiederverwendung und die Mitarbeiter sind nicht in

ausreichendem Maße im Hinblick auf Wiederverwendung geschult. Zum anderen treten fast alle in Abschnitt 1.3 genannten Hindernisse zutage. Die Wiederverwendung der Funktionen erfolgt durch Kopieren des Quellcodes in das Anwendungsprojekt. Dabei werden Information Hiding und Kapselung nicht vollständig umgesetzt. Somit können Funktionalitäten verändert oder hinzugefügt werden.
Die Unterstützung durch Werkzeuge zum Finden vorhandener Funktionen wird nur zum Teil gewährleistet.
Negative Auswirkungen gehen auch von festgelegten Terminen aus, der kurzfristige Erfolg steht im Vordergrund, es fehlt die Ausrichtung auf langfristigen Erfolg durch breite Wiederverwendung. Das fehlende Bewußtsein für die Auswirkungen von Software-Entwicklung mit redundanten Strukturen auf die Wartung ist eine weitere Ursache für mangelnde Umsetzung der Wiederverwendung.

b) Wiederverwendung von Daten
Aufgrund der fachlichen Anforderungen wird von neuen Anwendungen auch weiterhin auf bestehende Datenbanken über festgelegte Schnittstellen zugegriffen. Dabei werden Maßnahmen zum Information Hiding nur unzureichend getroffen. Die nutzende Anwendung greift explizit auf die Schnittstelle der bestehenden Datenbank zu. Eine Erweiterung der Unabhängigkeit und Austauschbarkeit wäre wünschenswert.
Innerhalb der mit COMPOSER erstellten Anwendungen werden Datenmodelle vollständig wiederverwendet. Es existiert ein Gesamtdatenmodell, aus dem bestehende Datenstrukturen, die in neuen Anwendungen benötigt werden, in das Anwendungsmodell *migriert* werden. In der Gegenrichtung werden neue Datenstrukturen konsistent in das Gesamtmodell eingebunden (mit sehr guter Unterstützung des Werkzeuges COMPOSER). Das Werkzeug ermöglicht dabei parallele Bearbeitung unterschiedlicher Ausschnitte des Gesamtdatenmodells. Im Ergebnis fügen sich alle mit COMPOSER entwickelten Anwendungen in das Gesamtdatenmodell ein.
Das Vorgehensmodell zur Entwicklung der Datenstrukturen legt fest, daß neue Datenstrukturen, die während eines Projektes entstehen, in die bestehende Datenwelt integriert werden müssen. Insbesondere sind redundante Strukturen, wie zum Beispiel das Anlegen einer Entität Kunde neben der bestehenden Partnerstruktur, unzulässig.

Die bestehenden Grundlagen (Unternehmensmodell) bieten eine gute Basis, um mit Hilfe eines neuen, auf Wiederverwendung ausgerichteten Vorgehensmodells Wiederverwendung in der R+V-Versicherung in breiter Form umzusetzen. Im folgenden soll die Werkzeuglandschaft auf ihre Möglichkeiten im Hinblick auf Wiederverwendung untersucht werden.

3.1.3 Analyse der Werkzeuge

Wiederverwendung muß in die bestehende Software-Entwicklungsumgebung eingebunden werden. In der R+V-Versicherung ist die Software-Entwicklung durch den Einsatz des Anwendungsgenerators IEF/ COMPOSER gekennzeichnet.
Dieses Werkzeug bietet umfassende Möglichkeiten zur Software-Erstellung über alle Entwicklungsphasen hinweg. Ausgehend von der Daten- und Funktionsanalyse in der Analysephase über Daten- und Funktionsmodellierung in der Designphase bis hin zur Implementierung der fachlichen Logik und der Entwicklung der Benutzerschnittstellen wird die Software-Entwicklung durch COMPOSER unterstützt. Die Programmierung erfolgt in einer abstrakten Beschreibungssprache aus der die ausführbaren Programme in Abhängigkeit von der Zielumgebung generiert werden. So können gleiche Programme auf verschiedenen Plattformen eingesetzt werden, was insbesondere unter dem Aspekt der heterogenen Hardware-Plattformen im Unternehmen Vorteile bringt.
COMPOSER unterstützt strukturierte Anwendungsentwicklung durch verschiedene Möglichkeiten der Modularisierung, somit existieren Kandidaten zur Erstellung von Komponenten (siehe Abschnitt 3.2.1).
Durch COMPOSER werden bisher jedoch weder Information Hiding noch umfassende Möglichkeiten zum Suchen oder Speichern von wiederverwendbaren Bauteilen unterstützt.
Diese unzureichende Unterstützung seitens des Werkzeugs kann durch organisatorische Maßnahmen kompensiert werden (siehe Abschnitt 3.3), so daß festgestellt werden kann, daß die Werkzeuglandschaft bei der R+V-Versicherung zur komponentenbasierten Anwendungsentwicklung geeignet ist. Da der eingesetzte Anwendungsgenerator leistungsfähige Möglichkeiten zur Oberflächenerstellung und Ablauforganisation bietet, ergeben sich unter Berücksichtigung der Erkenntnisse zu Framework's nur sehr geringe Vorteile durch Wiederverwendung von Frameworks. Design Patterns in Form des [UWM92]

oder von Unternehmensstandards werden bereits in hohem Maße angewendet, somit empfiehlt sich eine Konzentration der Weiterentwicklung der Wiederverwendungsaktivitäten im Hinblick auf komponentenbasierte Anwendungsentwicklung, wobei Action Blocks und Procedure Steps erfolgversprechende Kandidaten zur Entwicklung von Komponenten sind.

3.1.4 Schlußfolgerungen

Ausgehend von den Erkenntnissen über die derzeitige Situation der Software-Entwicklung können folgende Aussagen getroffen werden:

- Die Grundvoraussetzungen für die Einführung von Wiederverwendung sind gegeben.

- Aufgrund der Struktur der Systemlandschaft und der Entwicklungswerkzeuge erscheint die komponentenbasierte Anwendungsentwicklung als erfolgversprechendster Ansatz zur Wiederverwendung in der R+V-Versicherung.

- Es müssen organisatorische Strukturen geschaffen werden, um den Prozeß der Wiederverwendung optimal zu unterstützen.

Die zu treffenden Maßnahmen sollen nun anhand der Prozeßbeschreibung entwickelt werden.

3.2 Komponentenbasierte Entwicklung bei R+V-Versicherung

Um eine mögliche Vorgehensweise der komponentenbasierten Anwendungsentwicklung im Umfeld der R+V-Versicherung zu beschreiben, soll die prototypische Implementierung einer Kfz-Versicherungs-Anwendung dienen.
Zu Beginn ist zu klären, was Komponenten im COMPOSER-Umfeld sind und welche Anforderungen sie erfüllen. Entsprechend der zentralen Bedeutung wird danach die Verwaltung der vorhandenen Komponenten betrachtet. Die Entwicklungsschritte zur komponentenbasierten Anwendungsentwicklung werden formal beschrieben, im Anschluß wird jeweils das Vorgehen anhand des Beispieles Kfz-Versicherung verdeutlicht.
Dabei spiegelt sich die Trennung von Komponenten- und Anwendungsentwicklung in der getrennten Betrachtung der beiden Schritte wider.

3.2.1 Komponenten im COMPOSER-Umfeld

Um festzulegen, was unter Komponenten im Umfeld von COMPOSER zu verstehen ist, sollen zuerst Kandidaten dargestellt werden, die zu Komponenten erweitert werden können. Nachfolgend werden die Beschreibungselemente dieser Komponenten benannt.

3.2.1.1 Kandidaten für Komponenten in COMPOSER

COMPOSER bietet leistungsfähige Möglichkeiten zur Modularisierung der Anwendung wie *Action Blocks* und *Procedure Steps*, deren Wiederverwendung einfach zu handhaben ist. Diese Bausteine sollen kurz beschrieben werden:
Action Block:

Ein Action Block (AB) besteht aus einem Deklarationsteil, in dem vier Arten von Datensichten definiert werden (Importsichten, Exportsichten, lokale Sichten und Entity-Sichten). Die definierten Sichten werden im Implementierungsteil genutzt, um die Funktionalität des Action Block zu implementieren.

Beispiel: AB zum Lesen eines Datensatzes aus einer Datenbank

```
+- IPA_PARTNER_LESEN_I
| IMPORTS:                                                    :importierte Daten
| Work View imp_partner kfz_partner (optional,transient,import only)    :Name der Datenicht
```

```
|   uid (optional)                                          :Name des Attributs
| EXPORTS:                                                  :exportierte Daten
|  Work View  exp_partner kfz_partner (transient,export only)  :Name der Datenicht
|   telefonnummer                                           :Namen der Attribute
|   hausnummer
|   straáe
|   grosskundennummer
|   staatszugeh"rigkeit
|   zusatz
|   name
|   wohnort
|   anrede
|   nummer
|   typ
|   status
|   uid
| LOCALS:                                                   : lokal nötige Daten (leer)
| ENTITY ACTIONS:                                           :Datensichten für DB-Zugriffe
|  Entity View ent_partner partner
|   uid
|   letzte_aenderung
|   status
|   typ
|   nummer
|   matchcode
|   anrede
|   name
|   zusatz
|   staatszugehoerigkeit
|   grosskundennummer
|   jahr_ersterfassung
|   art_letzte_aenderung
|   userid_letzte_aenderung
|  Entity View ent_adresse adresse
|   strasse
|   ort
|   uid
|
| +- READ ent_adresse adresse                                  :lesender DB-Zugriff
| |   ent_partner partner                                      : über Relationen
| |   WHERE DESIRED ent_partner partner wohnt_in DESIRED ent_adresse adresse
| |   AND DESIRED ent_partner partner uid IS EQUAL TO imp_partner kfz_partner
| |     uid
| +- WHEN successful
| | SET exp_partner kfz_partner anrede TO ent_partner partner anrede   :Wertzuweisungen
| | SET exp_partner kfz_partner uid TO ent_partner partner uid
| | SET exp_partner kfz_partner name TO ent_partner partner name
| | SET exp_partner kfz_partner staatszugeh"rigkeit TO ent_partner partner
| |     staatszugehoerigkeit
```

```
| | SET exp_partner kfz_partner status TO ent_partner partner status
| | SET exp_partner kfz_partner zusatz TO ent_partner partner zusatz
| | SET exp_partner kfz_partner typ TO ent_partner partner typ
| | SET exp_partner kfz_partner grosskundennummer TO ent_partner partner
| |     grosskundennummer
| | SET exp_partner kfz_partner nummer TO ent_partner partner nummer
| | SET exp_partner kfz_partner wohnort TO ent_adresse adresse ort
| | SET exp_partner kfz_partner straáe TO ent_adresse adresse strasse
| +- WHEN not found
| +--
+--
```

Ein Action Block wird mit Hilfe von Parametern aufgerufen, mit deren Hilfe den definierten Sichten Werte zugewiesen werden.

Beispiel: Aufruf des Beispiel-AB

```
USE ipa_partner_lesen_i                          : Aufruf des Action Block
 WHICH IMPORTS: Work View  imp_partner kfz_partner   : übergebene Sicht
 WHICH EXPORTS: Work View  exp_partner kfz_partner   : erhaltene Sicht
```

Procedure Step

Ein Procedure Step (PS) stellt einen Teilprozeß dar, der aus einer Abfolge von Aktivitäten besteht. Der gesamte Teilprozeß kann wiederverwendet werden. Der Übergang zu einem Procedure Step erfolgt durch sogenannte *Flows* (nur eine Richtung) oder *Links* (mit anschließender Rückkehr), bei denen Datenstrukturen übergeben werden.

Die genannten Bestandteile von COMPOSER-Anwendungen können so erweitert werden, daß die Anforderungen an Komponenten aus Abschnitt 2.1.2 erfüllt werden (siehe Bewertung in Abschnitt 3.2.6). Dabei können Action Blocks als Services der Komponente fungieren. Procedure Steps können zu Komponenten erweitert werden. Die dazu zu treffenden Maßnahmen werden in den folgenden Abschnitten aufgeführt.

Im Mittelpunkt des Überganges zu Komponenten steht die Kapselung semantisch zusammengehöriger Systemteile. Dabei sind die Grenzen so festzulegen, daß Kosten und Nutzen der Kapselung in einem günstigen Verhältnis stehen. Erfahrungen anderer

Unternehmen zeigen, daß Datenstrukturen von Komponenten zwischen 10 und 100 Entitäten enthalten sollten [Jo95].

Generell muß man zwischen GUI- und Nicht-GUI-Komponenten unterscheiden. Während GUI-Komponenten die grafische Benutzerschnittstelle bereits implementieren, muß diese bei Nicht-GUI-Komponenten anwendungsspezifisch erstellt werden. Vorteil der Nicht-GUI-Komponenten ist die höhere Portabilität (PC, Großrechner), da sie nicht an ein bestimmtes graphisches System gebunden sind.
Action Blocks implementieren keine GUI Schnittstellen, Procedure Steps können mit oder ohne graphische Benutzerschnittstelle erstellt werden.

3.2.1.2 Beschreibungselemente von Komponenten in COMPOSER

Analog zu Abschnitt 2.1.1 wird die Spezifikation der Komponente in Datenstrukturen und darauf definierte Funktionen unterteilt. In COMPOSER steht im Mittelpunkt eines jeden Anwendungsmodells ein relationales Datenmodell. Für Komponenten wird dem Nutzer ein unimplementiertes Datenmodell zur Verfügung gestellt, d.h. die Datenstrukturen dienen lediglich dem Datenaustausch und entsprechen nicht den tatsächlich in einer Datenbank umgesetzten Strukturen.
Die Funktionen werden lediglich durch Übergabeparameter und Pre- und Postconditions definiert.

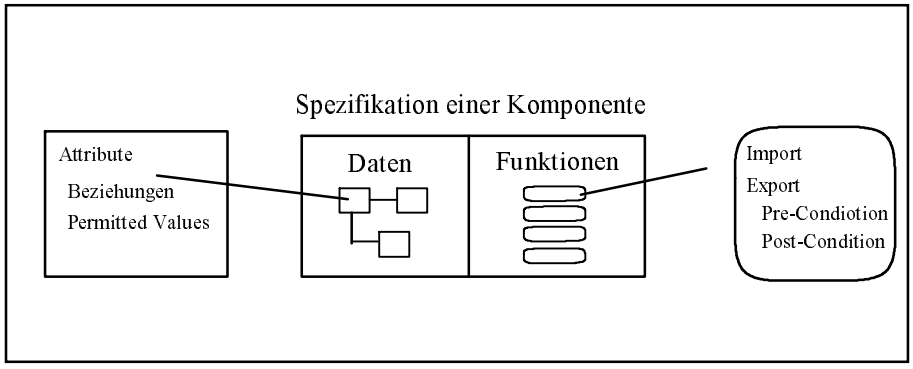

Abb. 3.2: Beschreibungselemente von COMPOSER-Komponenten

Zum besseren Verständnis werden sowohl Daten als auch Funktionen mit Hilfe der in COMPOSER zur Verfügung stehenden "*Description Fields*" ausführlich beschrieben.

3.2.2 Verwaltung der Komponenten in COMPOSER

Wie bei der Betrachtung der Werkzeuge in Abschnitt 3.1.3 festgestellt, existiert in der R+V-Versicherung kein Werkzeug zur Verwaltung der Komponenten. Dies ist jedoch eine unabdingbare Voraussetzung für erfolgreiche Wiederverwendung. Auf Grundlage einer Kooperation der Firmen MICROSOFT und TEXAS INSTRUMENTS ist die Einbindung des *Microsoft Repository* geplant, dessen Spezifikation alle Anforderungen aus Abschnitt 2.2.2.1 berücksichtigt. Bis zur Verfügbarkeit dieses Werkzeuges müssen jedoch andere Maßnahmen zur Verwaltung der Komponenten getroffen werden.

Das Werkzeug COMPOSER bietet mit seiner zentralen Enzyklopädie eine Vielzahl von Funktionen, mit deren Hilfe COMPOSER-Komponenten verwaltet werden können.

Dabei wird jeder Anwendung ein *Modell* zugewiesen. Der Austausch von Datenstrukturen und Funktionen erfolgt über den Mechanismus der *Migration*. Damit wird ein Kopieren unter Beibehaltung der Informationen zur Quelle bezeichnet. Die Migrationsmechanismen beinhalten Vorkehrungen zur Konsistenz der Datenmodelle, abhängige Strukturen werden automatisch mit überführt (z.B. Datenmodell eines migrierten Action Block).

Die COMPOSER-Enzyklopädie enthält Suchmechanismen, z.B. können über eine "*where used*"-Funktion alle Verwendungen einer Funktion oder eines Datenfeldes festgestellt werden.

Um Wiederverwendung von Komponenten, insbesondere deren Suche, zu ermöglichen, ist eine neue Organisation der Modelle in der Enzyklopädie einzuführen (siehe Abschnitt 3.2.7.2).

In diesen Strukturen können mittels des in COMPOSER enthaltenen *Data Model Browser* leistungsfähig Suche und Bewertung der Komponenten durchgeführt werden. Der Data Model Browser ermöglicht neben der Anzeige der Datenstrukturen und der darauf definierten Funktionen auch Anzeige von Beschreibungen zu Funktionen, Attributen und Verweise zu Analyse-Funktionen.

3.2.3 Vorgehensweise bei der Anwendungsentwicklung (Komponentennutzung)

Analog zu den Prinzipien aus Abschnitt 2.2 erfolgt die Projektabwicklung für die Entwicklung einer Anwendung getrennt von der Entwicklung der Komponenten (Abbildung 3.3).

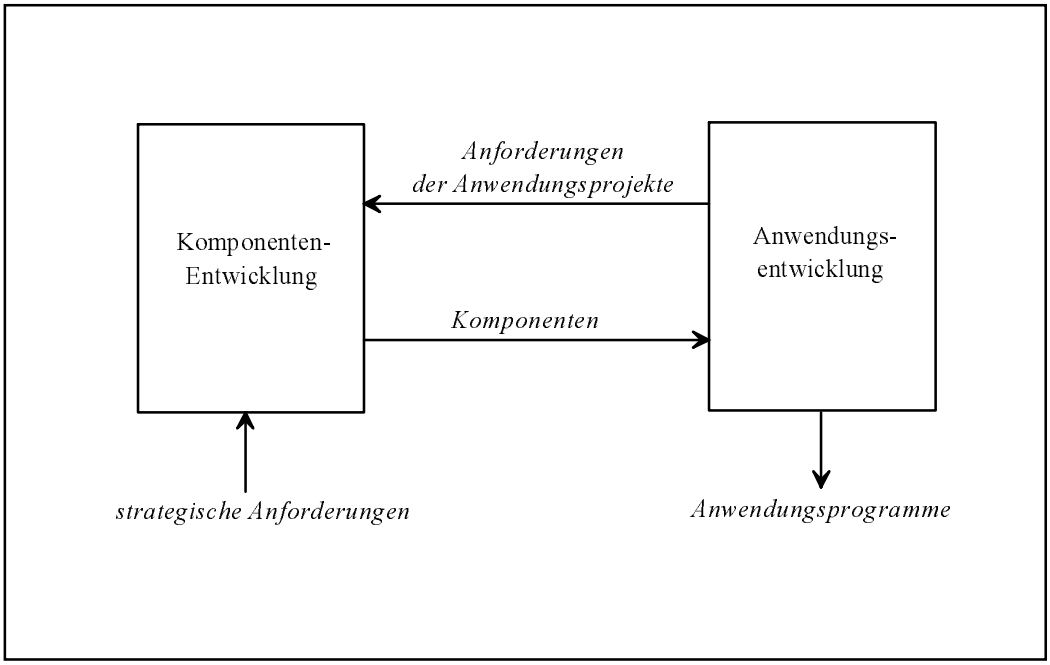

Abb. 3.3: Trennung der Entwicklung von Komponenten und Anwendungen

Die Anwendungsentwicklung erfolgt in den Phasen:

1. Anforderungsanalyse
2. Abgleich der Anforderungen mit Komponentenmanagement
3. Erstellen des Anwendungsrahmens
4. Einbinden der Komponenten
5. Test der Anwendung

Die einzelnen Schritte sollen im weiteren genau beschrieben werden.

3.2.3.1 Anforderungsanalyse (Phase 1)

Analog zur herkömmlichen Projektabwicklung steht am Beginn der Anwendungsentwicklung die Anforderungsanalyse. Dabei werden sowohl die erforderlichen fachlichen Funktionen als auch die zugrundeliegenden Daten festgelegt.
Wie bereits heute umgesetzt werden die Datenstrukturen unter Berücksichtigung des Gesamtdatenmodells der R+V-Versicherung so entwickelt, daß eine weitestgehend homogene Datenstruktur innerhalb des Unternehmens entsteht. Dazu werden bereits bestehende Datenmodellausschnitte in das Projekt einbezogen, um eine Mehrfachentwicklung auszuschließen und sehr früh eine Abstimmung mit bestehenden Erkenntnissen zu ermöglichen. Mit der Mehrfachnutzung der zugrundeliegenden Datenstrukturen wird eine entscheidende Grundlage für mögliche Mehrfachnutzung von darauf basierenden Funktionen geschaffen.
Die Festlegung der Daten erfolgt auf einer abstrakten Stufe, d.h. es werden lediglich Entscheidungen getroffen, *welche* Daten benutzt werden, *wie* diese Daten implementiert sind, bleibt offen.
Weiterhin sind die benötigten Funktionen zu den festgelegten Daten festzulegen und zu beschreiben, wobei wiederum das *"Was leistet die Funktion ?"* im Vordergrund steht.

Beispiel:
Die Kfz-Anwendung soll zum einen zum Erstellen und Verwalten von Kfz-Versicherungsverträgen dienen, zum anderen soll eine schnelle Angebotserstellung enthalten sein.
Der Vertragsverwaltung liegen Vertragsdaten (im Sinne des [UWM92]), Produktdaten, Objektdaten (des Kfz als versichertes Objekt) sowie Partnerdaten zugrunde. Die Angebotserstellung benötigt Vertragsdaten, Produktdaten und Objektdaten .
Für jede dieser Datenstrukturen sind nun die Inhalte festzulegen, die für die Erfüllung der Anforderung benötigt werden. Im Fall der Vertragsdaten sind dies Vertragsbeginn, Vertragsende, Antragsdatum, Vertragsnummer und Vertragsstatus. Zu beachten ist die Existenz technischer Felder (UID) zur eindeutigen Identifikation der Datensätze, die nicht in der eigentlichen Anforderung spezifiziert wurden, zur technischen Abarbeitung von Funktionalitäten aber nötig sind.

Die benötigten Funktionen, denen die Vertragsdaten zugrunde liegen, sind Vertrag neu anlegen, Vertragsdaten ändern, Vertrag zu einem Partner lesen, Vertrag zu einem Kfz lesen, Vertrag zu einer Versicherungsscheinnummer lesen und Vertrag aufheben.
Analog sind die Anforderungen an die anderen Teilbereiche der Anwendung zu spezifizieren. Die Ergebnisse sind in der folgenden Tabelle zusammengefaßt:

Kerneinheit	Benötigte Attribute	Benötigte Operationen
Vertrag	Versicherungsscheinnummer Vertragsbeginn Vertragsende Antragsdatum Vertragsstatus	Vertrag lesen zu Partner-UID Vertrag lesen zu VSNR Vertrag lesen zu Kfz-Kennzeichen Vertrag neu anlegen Vertragsdaten ändern Vertrag beenden
Partner	Kundennummer Großkundennummer Anrede Name Zusatz Wohnort Straße Hausnummer Staatszugehörigkeit Telefonnummer Status	Partner suchen nach Kundennr. Partner suchen nach Großk.nr. Partner suchen nach Name Partner lesen zu UID Partner neu anlegen Partnerdaten ändern Partner löschen

Produkt	Selbstbeteiligung Vollkasko Selbstbeteiligung Teilkasko Schadenfreiheitsklasse HP Schadenfreiheitsklasse VK Zahlungsweise Prämie Haftpflichtversicherung Prämie Teilkasko Prämie Vollkasko	Prämie Vollkaslo berechnen Prämie Teilkasko berechnen Prämie Haftpflicht berechnen Abschlußsaldo berechnen Produktausprägung anlegen Prämien berechnen (HP, VK, TK)
Objekt	Pol. Kennzeichen Zulassungskreis Hersteller Typ Modell Hubraum Leistung Farbe Status	Fahrzeug wählen nach Mermalen Fahrzeug suchen zu Kennzeichen Fahrzeug lesen zu UID Fahrzeug neu anlegen Fahrzeug versichern Fahrzeug löschen

Tab. 3.2: Anforderungen an Kfz-Anwendung

Die Resultate der ersten Phase bilden die Grundlage für Phase 2.

3.2.3.2 Komponentenmanagement (Phase 2)

Die in 3.2.3.1 festgestellten Anforderungen werden in Phase 2 mit vorhandenen, geplanten und möglichen zukünftigen Komponenten abgeglichen. Dazu werden vorhandene Komponenten auf die vollständige Abdeckung der Anforderungen hin überprüft (siehe Abb. 3.4).

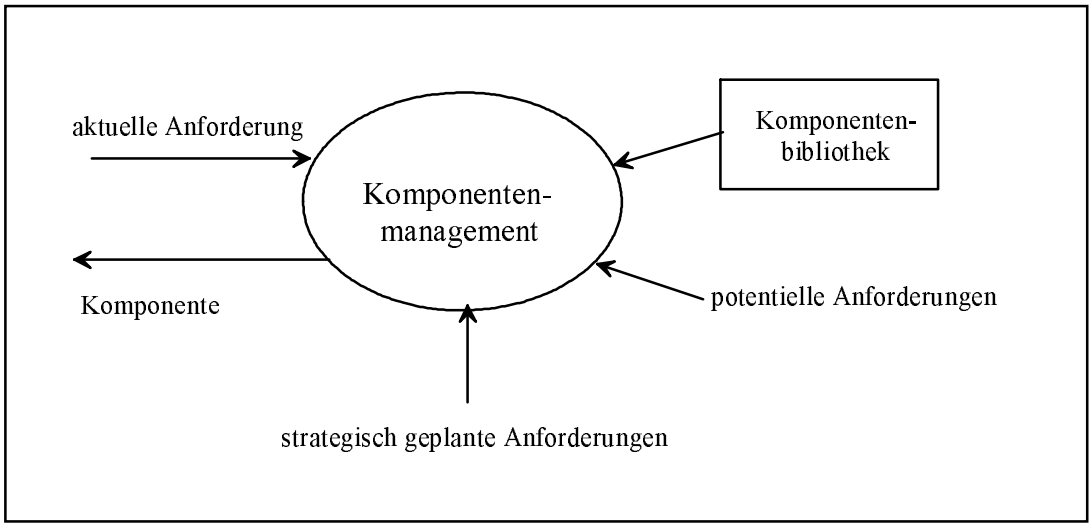

Abb. 3.4: Komponentenmanagement

Wird die Komponente neu erstellt, so sind für andere geplante Anwendungen nötige Datenstrukturen und Funktionen in die Anforderung einzubeziehen (siehe nachfolgende Beispiele). Die unterschiedlichen Kerneinheiten werden durch verschiedene Entwicklergruppen betreut. Es muß also zu jeder Kerneinheit eine Gruppe geben, die Neuentwicklungen durchführt, Änderungen vornimmt und Wartung gewährleistet. Da innerhalb überschaubarer Gruppen die Kommunikation in stärkerem Maße gewährleistet ist, empfiehlt es sich, die einzelnen Gruppen in Abhängigkeit der Struktur der Kerneinheit weiter zu unterteilen, so daß fachlich zusammengehörende Systemteile von einer Untergruppe betreut werden. Damit wird gewährleistet, daß die Entwickler einen besseren Überblick über das jeweilige Teilgebiet behalten. Dadurch kann schneller und genauer entschieden werden, wann was neu entwickelt und wann wiederverwendet werden kann.

Beispiel:

Fall a)

Die Anforderungen zur Verwaltung der Partnerdaten werden vollständig durch eine existierende Partnerverwaltung abgedeckt. Diese liegt als gekapselter Procedure Step mit vollständiger Benutzerschnittstelle vor.

Als Resultat des Abgleiches wird dem Anwendungsentwickler die Schnittstelle (Spezifikationsmodell ohne Implementierungsdetails) zur Erstellung des Anwendungsrahmens

zur Verfügung gestellt. Dies geschieht durch Migration eines leeren Procedure Steps und der für die Datenübergabe nötigen Worksets (Daten) in das Anwendungsmodell.
Die Worksets stellen die benötigten Datenstrukturen dar, im speziellen Fall des Partners sind das Kundennummer, Großkundennummer, Anrede, Name, Zusatz, Wohnort, Straße, Hausnummer, Staatszugehörigkeit, Telefonnummer und Status (wie in 3.2.1 angefordert) erweitert um einen eindeutigen Identifikator (UID). Es soll hier ausdrücklich darauf hingewiesen werden, daß die Struktur des Worksets in keiner Weise mit der Struktur des Datenmodells der Komponente übereinstimmt, vielmehr erfolgt eine Zuordnung der benötigten Daten zu den Datenstrukturen innerhalb der Komponente, die für den Nutzer verborgen bleibt. Im konkreten Beispiel sind Adreßdaten im Workset mit Angaben zur Person zusammengefaßt, da für Kfz-Verträge nur eine Anschrift je Person relevant ist. Tatsächlich kann in der Partnerverwaltung (Komponente) aufgrund der Trennung der Partner- und Adreßdaten in über eine Relation verbundene Entitäten jedoch eine Mehrfachzuordnung (Erstwohnsitz, Zweitwohnsitz, Korrespondenzanschrift) stattfinden.
Zur Behandlung von Ausnahmesituationen und zur Analyse der Resultate wird ein zusätzliches technisches Workset "irg_component" zur Verfügung gestellt, das Informationen zu aufgetretenen Fehlern und deren Ursachen enthält.
Die Auswertung dieser Werte muß im Anwendungsrahmen erfolgen.

Fall b)
Die Analyse der Anforderung an den Vertragsteil ergibt, daß nur ein Teil der benötigten Funktionalität vorhanden ist. Folglich müssen nicht bestehende Funktionen neu entwickelt werden. Dazu werden zuerst die Anforderungen aus Phase 1 um mögliche Anforderungen anderer aktueller oder zukünftiger Anwendungen (z.B. für Lebensversicherung - versicherte Person, begünstigte Person) erweitert und eine gemeinsame Anforderung an die gewünschte Funktionalität wird spezifiziert. Die entstandene Spezifikation wird zum einen den Anwendungsentwicklern zur Verfügung gestellt, zum anderen entwickelt parallel die zuständige Komponentenentwicklungsgruppe die Implementierung zur gemachten Spezifikation (siehe Abschnitt 3.2.4).

Fall c)

Es existiert keinerlei Funktionalität zu einem Systemteil, hier zum Beispiel zur Kfz-Verwaltung. Es wird wie mit den fehlenden Funktionen in Fall b) verfahren, zusätzlich sind die internen Datenstrukturen der Komponente zu entwickeln und die Schnittstellen darauf abzubilden (siehe Abschnitt 3.2.4).

Am Ende der zweiten Phase sind den Anwendungsentwicklern alle benötigten Datenstrukturen und alle angeforderten Funktionalitäten zur Verfügung gestellt, damit kann zu Phase 3 übergegangen werden.

3.2.3.3 Erstellung des Anwendungsrahmens (Phase 3)

Die Erstellung des Anwendungsrahmens beinhaltet zwei Schritte, zum einen muß die zugrundeliegende fachliche Funktionalität (Verknüpfung der Komponenten) abgebildet werden, zum anderen werden die nötigen Benutzeroberflächen der Anwendung erstellt. Dazu werden die zu unterstützenden fachlichen Abläufe mit Hilfe der zur Verfügung gestellten Operationen modelliert.

Beispiel:
Der prinzipielle Arbeitsablauf der Angebotserstellung soll wie folgt modelliert werden:

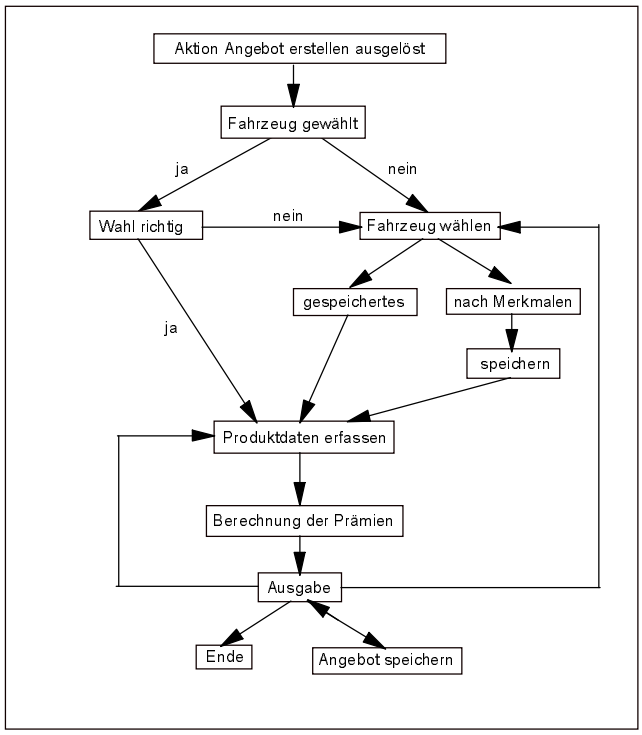

Abb. 3.5: Fachlicher Ablauf der Angebotserstellung

Entsprechend werden alle zu verwirklichenden Abläufe modelliert. Die Modellierung der Abläufe geschieht begleitend zur Erstellung und Bearbeitung der Anforderungen an die Komponenten.

Stehen die Spezifikationen zur Verfügung, kann mit der Erstellung der Oberfläche begonnen werden. Dazu werden zu den Abläufen die entsprechenden Darstellungen für den Bildschirm entwickelt. Im speziellen Fall handelt es sich um GUI-Fenster, die mit den auszuführenden Aktivitäten verbunden werden. Dieses Vorgehen gewährleistet eine frühe Abstimmung mit dem Auftraggeber der Anwendung, da die Benutzerschnittstelle erkennen läßt, ob die Anwendung das leisten wird, was sie leisten soll.

Beispiel:

Der Einstiegsbildschirm der Anwendung soll einen Kfz-Versicherungsvertrag darstellen. Dazu werden die Elemente der Worksets analog dem Äußeren des Vertrages auf der Oberfläche angeordnet. Für die Partnerangaben wird so eine View (Datensicht auf

Partnerdaten) für die Darstellung angelegt, die Angaben, wie Name, Kundennummer u.s.w. werden auf der Oberfläche plaziert.
In einer Menüstruktur werden die Verbindungen zu den einzelnen Aktivitäten hergestellt. Als ein Element findet sich dort ein Menüpunkt "Angebot erstellen". Durch das darauf definierte Click-Event wird die Aktivität "Angebot erstellen" aufgerufen.

Um Mehrfacheingaben zu vermeiden werden die aktuellen Fahrzeug- und Produktdaten an das Fenster "Angebot" übergeben. Die einzelnen Fenster werden erstellt und ihre Verknüpfungen werden hergestellt. Die fachliche Logik wird mit Hilfe der zur Verfügung gestellten Spezifikationen der Services abgebildet. Die Fensterfolge wird in folgender Abbildung dargestellt:

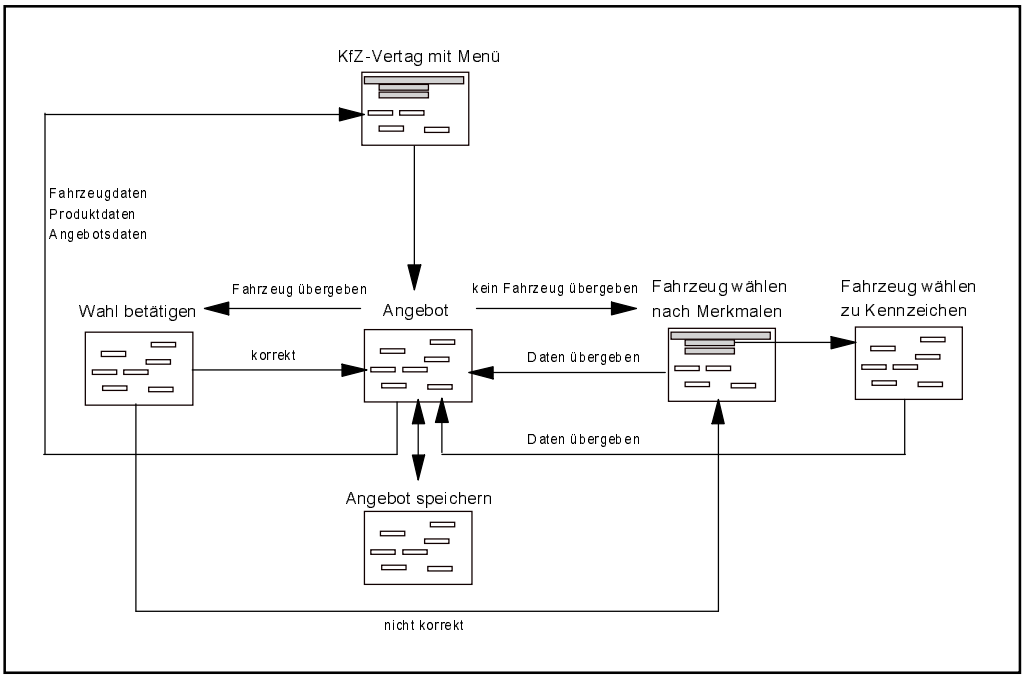

Abb. 3.6: Fensterfolge des Vorgangs "Angebot erstellen"

Die in den Fenstern enthaltenen Aktionsflächen wie Push-Buttons und Menüpunkte werden mit Aktionen verknüpft, die die fachliche Logik enthalten. Dies sind zum einen Verzweigungen zu anderen Fenstern und zum anderen Aufrufe der spezifizierten Komponenten.

3.2.3.4 Einbindung der implementierten Komponenten (Phase 4)

Um den Prozeß der Einbindung der Komponenten in die Anwendung darzustellen, soll zuerst die herkömmliche Verfahrensweise der Anwendungsentwicklung mit COMPOSER betrachtet werden.

a) Vorgehen bei der Anwendungserstellung ohne Wiederverwendung von Komponenten

Die Anwendungsentwicklung mit COMPOSER implementiert die generative Wiederverwendung (siehe Abbildung 1.4). Die Programmierung der Anwendungslogik erfolgt in einer abstrakten Beschreibungssprache. Die Benutzerschnittstelle wird graphisch erstellt und mit der Anwendungslogik verknüpft.
Aus den relationalen Datenbankschemata wird automatisch eine physische Datenbank erzeugt. Dabei kann zwischen verschiedenen DBMS gewählt werden. Die Codierung der Logik (inklusive der Datenbankzugriffe) erfolgt automatisch durch *Generierung*. Im anschließenden *Built* werden die erzeugten Quellcodes durch entsprechende Compiler in das ausführbare Programm übersetzt.

b) modifiziertes Vorgehen mit Wiederverwendung

Wurden zur Programmierung der Anwendung Spezifikationen von Komponenten verwendet, so müssen die dazugehörigen Implementierungen in die Anwendung eingefügt werden.

- Die Anwendung wird mit Ausnahme der Spezifikationen der Komponenten (leeren Action Blocks und Procedure Steps) generiert. Die Implementierungen der Action Blocks und Procedure Steps wird durch die Komponentenentwickler separat generiert und built wird ausgeführt. Die resultierenden Dateien der Action Blocks (*.obj, *.ddl, *.bnd) werden in das Anwendungsverzeichnis kopiert, danach wird das built für die Anwendung ausgeführt. Die Dateien der Procedure Steps (*.exe, *.dll) werden in die Laufzeitumgebung kopiert und erst zur Laufzeit eingebunden. Eine Besonderheit ergibt sich im Hinblick auf Datenbanken. Im untersuchten Umfeld konnte das verwendete DBMS DB2/2 nur mit einer Datenbank arbeiten. Deshalb mußte die Datenbankschemata der Komponenten zu

einer einzelnen Datenbank zusammengefaßt werden, indem die entsprechende *.ins-Datei um die notwendigen Create-Table-Statements erweitert wurde.

- Mit Verfügbarkeit der Version 4 des Werkzeugs COMPOSER ist es möglich, gekapselte Komponenten mit automatisch generierten CORBA- und OLE-Schnittstellen zu erzeugen. Diese können dann zur Laufzeit in einer CORBA-Laufzeitumgebung miteinander verknüpft werden.

3.2.3.5 Test der Gesamtanwendung (Phase 5)

Nach dem Erzeugen der Anwendung folgen die herkömmlichen Testschritte. Da die eingebundenen Komponenten separat getestet wurden, müssen lediglich das Zusammenspiel der Komponenten mit dem Anwendungsrahmen sowie die Richtigkeit des Rahmens überprüft werden.

3.2.4 Vorgehensweise bei der Komponentenerstellung

Die Erstellung der Komponenten kann synchron (bei Anforderung) oder asynchron (strategisch ohne Anforderung) erfolgen. Asynchrone Entwicklung sollte für solche Komponenten verwendet werden, deren Einsatz absehbar ist und die im Mittelpunkt der Anwendungslandschaft stehen. Im Fall der R+V-Versicherung sind solche Komponenten zum Beispiel Vertrag/ Rechtsgeschäft, Produkt oder Partner. Diese Systemteile werden in fast allen Anwendungen benötigt, sie stellen den Mittelpunkt des Versicherungsgeschäftes dar.

Dagegen kann die Kerneinheit Objekt schrittweise in Abhängigkeit der Anforderungen entwickelt werden, da aufgrund der stark unterschiedlichen Struktur der verschiedenen Objekte (Kfz, Verwertungsrechte, Tiere) eine umfassende Umsetzung erhebliche Aufwände erfordert. Dabei sollte jedoch die spätere Einbindung der anderen Objekte geplant werden. Die Austauschbarkeit der Komponenten ermöglicht es weiterhin, vorübergehende Umsetzungen in die Anwendung einzubinden, die später durch eine vollständige Komponente ersetzt werden, die eine äquivalente Schnittstelle (mit mindestens den gleichen Services) bietet.

In Abhängigkeit von bereits bestehenden Teilen müssen diese dem Entwickler der Komponente zur Verfügung gestellt werden.
Bei der Erstellung der Komponenten werden zu den Spezifikationen aus Phase 2 die Implementierungen entwickelt, nach der Erstellung der Komponenten werden diese in einer Testumgebung getestet. Nach erfolgreichem Test werden die für die Nutzung nötigen Dateien abrufbereit abgelegt.
Je nachdem, inwieweit bestehende Datenbanken, Altanwendungen oder Komponenten bei der Entwicklung neuer Komponenten genutzt werden, unterscheidet man vier Vorgehensweisen.

a) Vollständige Neuentwicklung

Bei vollständiger Neuentwicklung liegen keinerlei wiederverwendbare Elemente vor. Unter Berücksichtigung von Unternehmensdatenmodellen oder Branchenstandards wird die der Komponente zugrundeliegende Datenstruktur entwickelt.

Beispiel:
Das Datenmodell für die Speicherung der Fahrzeugdaten wird entwickelt. Da entschieden wurde, anstelle einer komplexen Objektdatenbank vorerst eine reine Kfz-Datenbank einzubinden, ergibt sich folgendes einfaches Schema:

```
Type     Name
Subject Area +-KFZ_CBD_DIPLOM
Subject Area ¦ +-CBD_KFZ_FAHRZEUG
Entity Type  ¦ ¦ +-FAHRZEUG
Attribute    ¦ ¦ ¦  FAHRGESTELLNUMMER (Text, 30, Optional, Basic)
Attribute    ¦ ¦ ¦  STATUS      (Text, 15, Optional, Basic)
Attribute    ¦ ¦ ¦ I UID        (Text, 16, Mandatory, Basic)
Attribute    ¦ ¦ ¦  ZULASSUNGSKREIS  (Text, 30, Optional, Basic)
Attribute    ¦ ¦ ¦  FK_PARTNER    (Text, 16, Optional, Basic)
Attribute    ¦ ¦ ¦  KENNZEICHEN    (Text, 11, Optional, Basic)
Relationship ¦ ¦ ¦  Always ZU_EINER One VARIANTE
       ¦ ¦ +-
Entity Type  ¦ ¦ +-HERSTELLER
Attribute    ¦ ¦ ¦  NAME (Text, 20, Optional, Basic)
Attribute    ¦ ¦ ¦ I UID (Text, 16, Mandatory, Basic)
Relationship ¦ ¦ ¦  Always PRODUZIERT One or More TYP
       ¦ ¦ +-
Entity Type  ¦ ¦ +-TYP
Attribute    ¦ ¦ ¦  BAUJAHR_VON (Date, 8, Optional, Basic)
Attribute    ¦ ¦ ¦  BEZEICHNUNG (Text, 20, Optional, Basic)
Attribute    ¦ ¦ ¦  BAUJAHR_BIS (Date, 8, Optional, Basic)
Attribute    ¦ ¦ ¦ I UID     (Text, 16, Mandatory, Basic)
```

```
Relationship ¦ ¦ ¦  Always PRODUZIERT_VON One HERSTELLER
Relationship ¦ ¦ ¦  Always HAT One or More VARIANTE
       ¦ ¦ +-

Entity Type  ¦ ¦ +-VARIANTE
Attribute    ¦ ¦ ¦  HP_KLASSE  (Number, 2, Mandatory, Basic)
Attribute    ¦ ¦ ¦  VK_KLASSE  (Number, 2, Mandatory, Basic)
Attribute    ¦ ¦ ¦  TK_KLASSE  (Number, 2, Optional, Basic)
Attribute    ¦ ¦ ¦  LSTG_PS    (Number, 3, Optional, Basic)
Attribute    ¦ ¦ ¦  LSTG_KW    (Number, 3, Optional, Basic)
Attribute    ¦ ¦ ¦  BEZEICHNUNG (Text, 20, Optional, Basic)
Attribute    ¦ ¦ ¦ I UID      (Text, 16, Mandatory, Basic)
Relationship ¦ ¦ ¦  Sometimes SCHLIEßT_EIN One or More FAHRZEUG
Relationship ¦ ¦ ¦  Always ZU One TYP
       ¦ ¦ +-
       ¦ +-
       +-
```

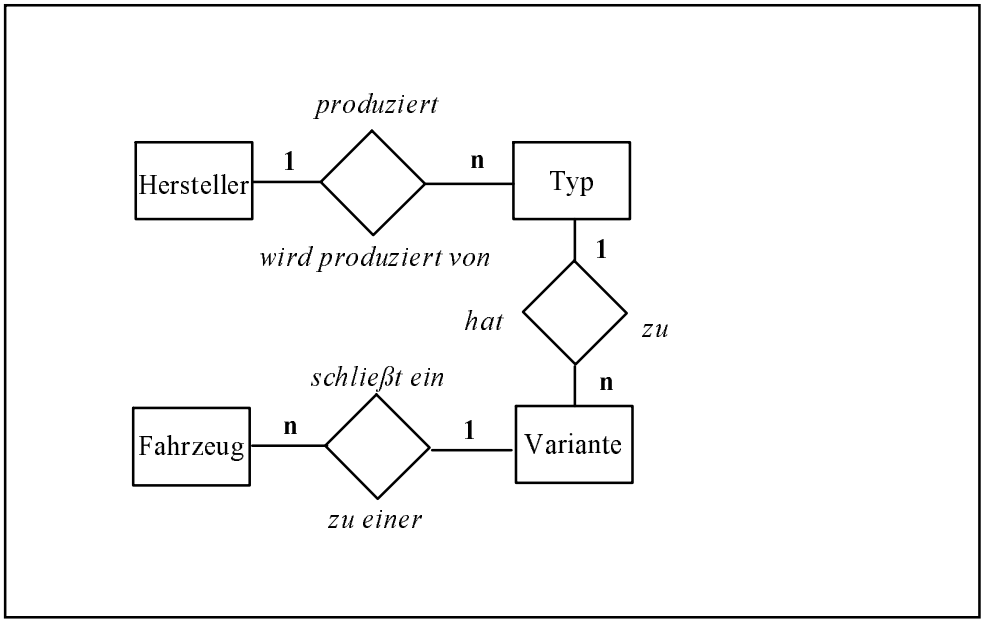

Abb. 3.7: ER-Diagramm der Kfz-Datenbank

Das angegebene Datenmodell enthält keine expliziten Beziehungen nach außen. Lediglich das Attribut "fk_partner" dient der Bindung eines Fahrzeuges an einen Partner (in der Rolle des Eigentümers), wobei der Wert von "fk_partner" der "UID" des jeweiligen Partners entspricht.

Die Komponente soll die in Abschnitt 3.2.3.1 festgelegten Anforderungen erfüllen. Dazu sind die entsprechenden Services zu entwickeln. Dazu müssen neben der Logik

des Dienstes auch die entsprechenden Wertzuweisungen zwischen Datenmodell der Komponente und Datenstruktur der Schnittstelle der Komponente erfolgen.

Beispiel:

Implementiert werden soll der Service Kfz_suchen_zu_Kennzeichen. Gegeben ist folgende Spezifikation:

```
BAA Action Block: IKF_KFZ_SUCHEN_S
_____________________________________________________________________

Action Block Description:
Sucht Fahrzeuge mit gegebenen Kennzeichen oder zu gegebenem Partner.
Die Ergebnisse werden als Trefferliste zurückgegeben.

 +- IKF_KFZ_SUCHEN_S
 |  IMPORTS:
 |   Work View  imp_kfz kfz (optional,transient,import only)
                                                        :Importierte Suchkriterien
 |    fk_partner (optional)
 |    kennzeichen (optional)
 |  EXPORTS:                                    :Gruppe für Ergebnisse
 |   Group View exp_lb_kfz (100,explicit,export only)
 |    Work View  exp_sel ief_supplied (transient)
 |     select_char                              :Auswahlfeld der Gruppe
 |    Work View  exp_le_kfz kfz (transient)     :Gruppenelement Kfz
 |     ausprägung
 |     hp_klasse
 |     tk_klasse
 |     vk_klasse
 |     zulassungskreis
 |     hubraum
 |     leistung
 |     fk_partner
 |     sfklasse
 |     kennzeichen
 |     typ
 |     hersteller
 |     uid
 |   Work View  exp_component irg_component (transient,export only)
                                          :record für Fehlercodes und -gründe
 |    reason_code
 |    return_code
 |  LOCALS:
 |  ENTITY ACTIONS:
 |
 | NOTE ********************************  : Liste garantierter Nachbedingungen
 |                                    : bei erfüllten Vorbedingungen
 |    Pre-Condition
 |        Kennzeichen ist nicht leer.
 |
 |    Post Condition
```

```
|        a) alle KFZ mit diesem Kennzeichen in der Liste
|            return code: +0001
|
|        b) keine KFZ mit diesem Kennzeichen
|            return-code: +0002
|
|    ************************************************************
|
|    Pre-Condition
|        Kennzeichen ist leer
|        Partner ist nicht leer
|
|    Post-Condition
|        a) Alle KFZ zu Partner in der Liste
|            return-code: +0001
|
|        b) Keine KFZ zu Partner
|            return-code: +0002
|
|    ************************************************************
| EXTERNAL                                  : extern implementierter Action Block
+--
```

In der Spezifikation wird das EXTERNAL-Statement ersetzt durch ein USE-Statement auf die Implementierung KFZ_SUCHEN_I:

```
| USE ikf_kfz_suchen_i
|   WHICH IMPORTS: Work View  imp_kfz kfz
|   WHICH EXPORTS: Group View exp_lb_kfz
|           Work View  exp_component irg_component
|           Work View  exp_kfz kfz
+--
```

Der Dienst soll zu einem gegebenen Fahrzeugkennzeichen die Liste der entsprechenden Fahrzeuge mit all ihren Merkmalen liefern (1. Pre- und Postcondition), wird kein Kennzeichen übergeben, so werden alle Fahrzeuge mit dem entsprechenden Eintrag im Feld fk_partner in der Liste zurückgegeben (2. Pre- und Postcondition). In Abhängigkeit des Suchverlaufs werden die entsprechenden return-codes (Fehler-Codes) und reasen-codes (Ursachen-Codes) gesetzt.
Es wird ein Datensatz mit zwei Feldern an den Dienst übergeben (Kennzeichen und fk_partner), der Dienst liefert eine Liste der entsprechenden Fahrzeuge und einen Datensatz mit Informationen zu aufgetretenen Fehlern und deren Ursachen (irg_component).
Die Implementierung dieses Dienstes ist in Anhang A enthalten.

b) Neuentwicklung mit Wiederverwendung einer Datenbank

Auf Basis der Unternehmensmodells wurden seit dem Jahr 1992 Datenstrukturen so entwickelt, daß sie in anderen Anwendungen wiederverwendet werden können. Somit entfällt die Entwicklung des Datenmodells der Komponente. In Abhängigkeit der Quelle der Datenbank kann bei COMPOSER-Datenbanken das Datenmodell durch Migration übernommen werden, darauf entwickelte Funktionen sind ebenfalls weiter nutzbar. Handelt es sich um eine relationale Datenbank, die nicht mit COMPOSER entwickelt wurde, so ist das entsprechende relationale Datenmodell in COMPOSER nachzubilden. Die bestehende (evtl. produktiv eingesetzte) Datenbank kann problemlos in der neuen Anwendung verwendet werden. Nichtrelationale Datenbanken können in COMPOSER mittels *Externer Action Blocks* eingebunden werden. Dazu werden von COMPOSER Schnittstellen generiert, in die in der jeweiligen Programmiersprache (C, COBOL) Zugriffsfunktionen auf die bestehende Datenbank eingebunden werden können.

Neue erforderlichen Dienste werden analog der Neuentwicklung erstellt.

c) "wrapping" von Altanwendungen

Analog dem dargestellten Vorgehen bei der Einbindung nicht-relationaler Datenbanken können auch Altanwendung über einen Externen Action Block (EAB) eingebunden werden. Der EAB stellt den "wrapper" dar, im eingefügten Code müssen Datenkonvertierung und Aufruf der Altanwendung implementiert werden.

d) Anpassung oder Erweiterung einer bestehenden Komponente

Die Anpassung einer vorhandenen Komponente entspricht einer Änderungsanforderung und wird im nächsten Abschnitt behandelt.

3.2.5 Vorgehensweise bei Änderungsanforderungen

Wesentliche Anforderungen, die mit der Verwendung von Komponenten verbunden werden, sind Änderbarkeit und Erweiterbarkeit. Diese Anforderungen beziehen sich sowohl auf Änderungen der Komponenten als auch auf Änderungen der nutzenden

Anwendungen. In beiden Fällen sollen die jeweils kooperierenden Systeme weitestgehend unbeeinflußt sein. Man muß dabei nach dem Grad der Änderung unterscheiden.

3.2.5.1 Änderung ohne Schnittstellenänderung

Änderungen ohne Einfluß auf die Schnittstelle sind sehr leicht auszuführen.

a) Änderungen der nutzenden Anwendung:

Da die nutzende Anwendung die Komponente weiterhin über die identische Schnittstelle aufruft, wird die Komponente durch die Änderung der Anwendung nicht berührt und muß nicht geändert werden. Mit der zu ändernden Anwendung wird wie bisher verfahren.

Beispiel:

Der Systemteil zur Angebotserstellung soll vereinfacht werden. Der Aufruf des Fensters "Wahl richtig" entfällt, statt dessen wird immer in das Fenster "Fahrzeug wählen" verzweigt, dort wird die Abfrage zur Wahl des Fahrzeugs hinzugefügt. Die Struktur der Anwendung ändert sich, ohne daß irgendeine Schnittstelle betroffen ist. Deshalb unterliegen keine Komponenten einer Änderungsanforderung.

b) Änderungen der Komponente

Müssen zu einer Komponente Implementierungsdetails geändert werden, so bleiben die gebotenen Schnittstellen unverändert. Dennoch sind die nutzenden Anwendungen betroffen. Es ist zwar nicht nötig, die nutzenden Anwendungen zu verändern, die Generierung der Anwendungen muß nach dem Ersetzen der alten Quelldateien der Komponente durch die veränderten jedoch wiederholt werden. Damit ist in den meisten Fällen auch ein erneuter Test verbunden.

Beispiel:

In der ersten Version der Kfz-Anwendung wurde das versicherte Objekt Kfz in einer eigenen Datenbank nur für Kraftfahrzeuge implementiert. Entsprechend der Maßgaben des [UWM92] muß jedoch eine Einordnung in die Datenwelt der R+V-Versicherung geschehen, um beispielsweise einfache Auswertungen mittels eines DSS (Desicion Support System) zu ermöglichen. Die Komponente "Versichertes Objekt", die die Verwaltung sämtlicher Objekte im Sinne des [UWM92] übernimmt, wurde erstellt, zur Vereinfachung der Verwendung wurden unterschiedliche Schnittstellen

für die Arbeit mit unterschiedlichen versicherten Objekten (Tier, Kfz, Verwertungsrecht, Gebäude, Person) bereitgestellt. Die Schnittstelle für Kfz ist identisch zur verwendeten Schnittstelle der bisherigen Komponente Kfz.

Technisch wird dies realisiert, indem die in der Anwendung eingebundene Spezifikation zur Implementierung genutzt wird, zum Beispiel durch Implementierung der Wertzuweisungen zwischen Schnittstelle "Kfz" und Schnittstelle "Versichertes Objekt - Kfz" und anschließenden Aufruf der entsprechenden Operationen der Komponente "Versichertes Objekt - Kfz" zur Erfüllung der spezifizierten Operationen der Komponente "Kfz".

Die neue Implementierung ist zu generieren und die entstandenen Quelldateien werden der Anwendung zur Verfügung gestellt. Es wird der Built-Prozeß erneut ausgeführt und die Anwendung ist aktualisiert.

3.2.5.2 Änderung mit Schnittstellenänderung

Nicht alle Änderungen sind soweit vorhersehbar, daß sie die bestehenden Schnittstellen nicht beeinflussen, das heißt, die Schnittstellen unterliegen teilweise auch Änderungsanforderungen.

Änderungen der Schnittstellen haben Auswirkungen für alle beteiligten Partner. Ursache sind Anforderungen, die von der bestehenden Komponente nicht erfüllt werden können. Als Folge wird eine geänderte Schnittstelle erzeugt. Die Verwendung der Operation mit der neuen Schnittstelle erfordert andere Aufrufe in der nutzenden Anwendung. Dazu wird die Schnittstelle in das Anwendungsmodell migriert, dort werden die Aufrufe der Komponente von der alten auf die neue Schnittstelle umgesetzt. Gegebenenfalls müssen die Auswertungsanweisungen der Fehler- und Ursachen-Codes geändert werden. Die Anwendung muß neu generiert werden und nach Hinzufügen der nötigen Komponentendateien kann der built-Schritt ausgeführt werden (Ablauf analog der Einbindung einer neuen Komponente).

Die geänderte Komponente sollte in einem neuen eigenen Modell abgelegt werden, da unter Umständen die ungeänderte Version der Komponente weiter in Benutzung bleibt.

Beispiel:
Eine denkbare Änderungsanforderung ist das Einbeziehen der Fahrzeugfarbe bei der Berechnung der Haftpflichtprämie eines Kfz. Der Einfluß der Farbe auf die Prämienberechnung, entspricht neueren Analysen, welche ergaben, daß Fahrzeuge bestimmter Farben (z.B. rot) höhere Schadenzahlen aufweisen.
Zur Einbindung dieser Anforderung muß entweder die Zuordnung der Typklasse die Fahrzeugfarbe berücksichtigen, dann ändert sich die Implementierung der Datenbank, in der die Typklassen den Fahrzeugen zugeordnet werden. Andernfalls muß zur Berechnung der Prämie das Attribut Farbe des Fahrzeugs mit übergeben werden. Damit ändert sich die Schnittstelle der Operation "HP_P_BERECHNEN_S".
Im zweiten Fall wird beispielsweise bei bestimmten Farben ein Risikozuschlag erteilt, dies ist in der Implementierung umzusetzen. Die neue Dienst "HP_BERECHNEN" der Komponente "Kfz-Prämienberechnung" wird generiert, gleichzeitig wird die neue Schnittstelle in die betroffenen Anwendungen eingebunden. Dazu werden die USE-Statements im Anwendungsmodell auf die alte Schnittstelle durch USE-Statements auf die neue Schnittstelle ersetzt (und der zusätzliche Wert Farbe wird übergeben). Anschließend wird ein erneutes built der betroffenen Anwendungen ausgeführt.

3.2.6 Bewertung der Komponenten in COMPOSER

Um die zu erwartenden Vorteile der komponentenbasierten Anwendungsentwicklung im vorliegenden Umfeld abschätzen zu können, sollen die Eigenschaften der Komponenten im COMPOSER-Umfeld betrachtet werden. Dabei soll der Vergleich mit den Anforderungen aus Teil I, Abschnitt 2.2.1 zur Bewertung der Leistungsfähigkeit dienen.

Komponenten können in COMPOSER mit Hilfe der gebotenen Description Fields (Beschreibungsfelder) ausführlich beschrieben werden. Die Festlegungen zu den anzugebenden Inhalten gewährleisten eine gute *Verständlichkeit*. Da die Beschreibungen zu einem Datenfeld, Action Block u.s.w. von jedem Auftreten aus abrufbar sind, ist eine umfassende Information des Nutzers gewährleistet. Voraussetzung ist lediglich eine ausreichende Beschreibung durch den Ersteller. Diese wird jedoch durch Qualitätssicherungsmaßnahmen kontrolliert.

Definiertheit und *Übertragbarkeit* werden durch die kontextunabhängige Erstellung der Komponenten und die anschließenden Tests garantiert.

Zugreifbarkeit wird nur bedingt gewährleistet, da kein Repository im Sinne der Anforderungen aus 2.2.2.1 vorhanden ist. Die Bindung der Komponenten an kleine Entwicklergruppen und die Suche durch das Komponentenmanagement sollen das Finden von Komponenten erleichtern. Eine gute Organisation der Ablage bestehender Komponenten kann eine weitere Verbesserung mit sich bringen. Auf Dauer wird jedoch eine Unterstützung durch ein Repository unabdingbar sein.

Hohe Kohäsion und *lose Kopplung* werden durch das vorgeschlagene Vorgehen auf Basis der Erkenntnisse des [UWM92] unterstützt, da zusammengehörige Dienste auf Grundlage der gemeinsamen Daten in gleichen und unterschiedliche Dienste in unterschiedlichen Komponenten zusammengefaßt werden. Unterstützt wird dies noch durch die Erstellung in getrennten Gruppen.

Additivität wird gewährleistet, die Verwendung von Diensten einer Komponente zur Erstellung einer anderen ist problemlos möglich.

Austauschbarkeit wird ebenfalls unterstützt. Wurden sowohl die zu ersetzende als auch die ersetzende Komponente aus der gleichen Spezifikation erzeugt, genügt die Ersetzung der Quelldateien in Phase 4, im anderen Fall müssen die in der Anwendung eingebundene Spezifikation und deren Aufrufe ausgetauscht werden, da die Composer-intern verwendeten Original Object ID (Feld zur eindeutigen Identifikation von Daten gleichen Ursprungs) unterschiedlich sind.

Konfigurierbarkeit kann durch die Komponente unterstützt werden, wenn die konträren Anforderungen unterstützt werden. Im Hinblick auf die Performance empfiehlt sich jedoch die Entwicklung unterschiedlicher Implementierungen zur Spezifikation in der Einsatz der einen oder anderen Version in Abhängigkeit der Anforderung.

Das beschriebene Verfahren unterstützt das Prinzip des *Information Hiding*. Die Benutzung der Komponenten erfolgt ohne Kenntnis der inneren Strukturen allein aufgrund der Spezifikation. Durch den alleinigen Zugriff auf die Daten über die zur Verfügung gestellten Services wird vollständige *Kapselung* erreicht.

Portierbarkeit wird durch die Möglichkeiten des Werkzeuges COMPOSER gewährleistet. Insbesondere Komponenten ohne GUI-Oberfläche sind auf unterschiedliche Hardware- (Großrechner, PC) und Systemplattformen (Betriebssysteme: MVS, Windows,

Windows NT, OS/2; DBMS: SYBASE, INFORMIX, DB2, ORACLE) übertragbar, aus identischem Pseudocode kann für diese Zielsysteme generiert werden.
Die *Kompatibilität zu Firmenstandards* bleibt erhalten, *Korrektheit* und *Zuverlässigkeit* werden durch Test und Qualitätskontrolle sichergestellt.

Die Datenorientierung der Anwendungsentwicklung mit COMPOSER und die vorgeschlagene Vorgehensweise unterstützen die Entwicklung von Komponenten im Sinne der Anforderungen. Wiederverwendung von Komponenten wird durch die Möglichkeiten von COMPOSER zur Modularisierung in Verbindung mit der Trennung der Komponenten von speziellen Anwendungen ermöglicht.

Zusammenfassend kann festgestellt werden, daß komponentenbasierte Anwendungsentwicklung mit den vorhandenen Werkzeugen unter Weiterentwicklung des Vorgehensmodelles erfolgversprechend umgesetzt werden kann.

3.2.7 Verwalten der Komponenten

Wie in 2.3 festgestellt, müssen durch organisatorische Maßnahmen Bedingungen geschaffen werden, die das Speichern, Suchen und Wiederverwenden der bestehenden Komponenten ermöglichen.

3.2.7.1 Organisation der Projektstruktur

Die vorgeschlagene Struktur der Anwendungsentwicklung befreit den *Anwendungsentwickler* von der Suche nach Komponenten, er erhält auf Grundlage seiner Anforderung Datenstrukturen und Funktionen im Anwendungsmodell zur Verfügung gestellt. Dies gewährleistet die Konzentration des Anwendungsentwicklers auf die Abbildung der fachlichen Zusammenhänge und der Benutzerschnittstellen in Zusammenarbeit mit dem Fachbereich, er wird von technischen Problematiken, wie Datenmodellierung, Programmierung von Datenbankzugriffen u.s.w. befreit.
Die Trennung von Anwendungs- und Komponentenentwicklung muß sich in der Projektstruktur widerspiegeln:

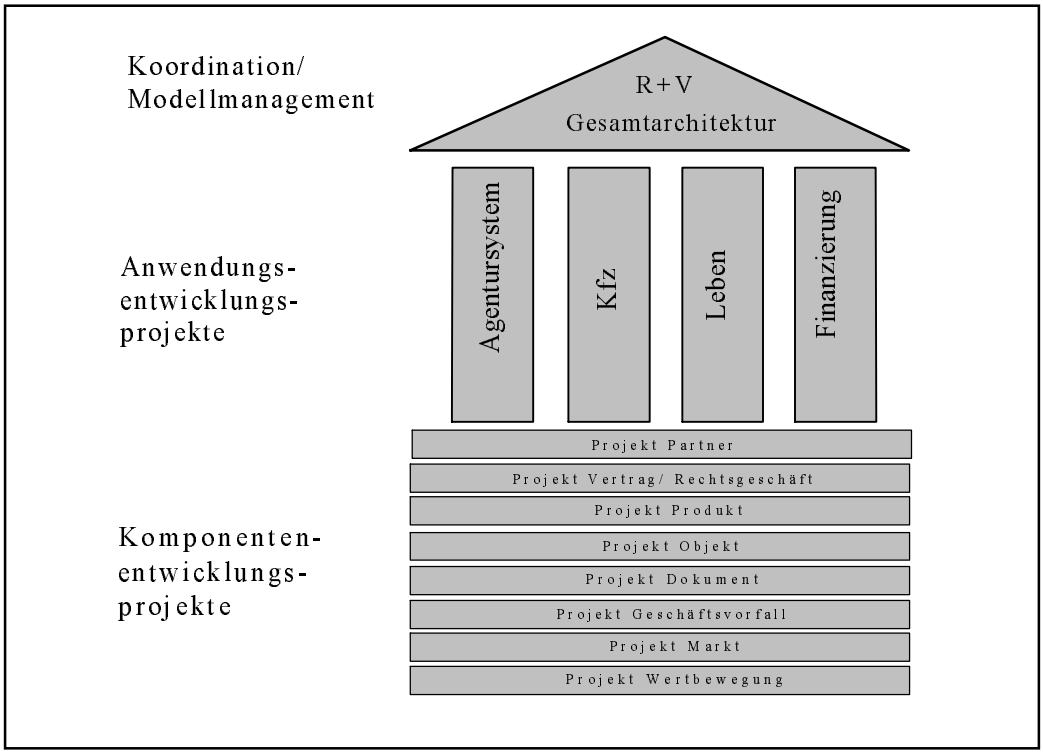

Abb. 3.8: Projektstruktur bei komponentenbasierter Anwendungsentwicklung

Die Darstellung in Abbildung 3.5 verdeutlicht die fundamentale Bedeutung der Komponenten, auf die die Anwendungsentwicklung aufbaut. Die Anwendungen werden unter dem gemeinsamen Dach einer einheitlichen Architektur entwickelt.

Die wesentliche Rolle bei der Wiederverwendung übernimmt das *Komponentenmanagement*. Eine zu schaffende Gruppe von Analytikern, die die Gesamtheit der Anwendungsarchitektur im Unternehmen überblickt, koordiniert die Anwendungsentwicklung über Projektgrenzen hinweg. Zu den Aufgaben gehören die Verteilung der Anforderungen auf die Komponentenentwicklungsprojekte, deren Erweiterung um tatsächliche und potentielle Anforderungen anderer Projekte, die Überführung der Ergebnisse einer Anforderung in das Anwendungsmodell sowie die Planung strategischer Komponenten ohne konkrete Anforderungen.

Die *Komponentenentwicklungsprojekte* erzeugen entsprechend den Vorgaben des Komponentenmanagements die benötigten Komponenten. Durch die Beschränkung der Verantwortlichkeit einer Komponentenentwicklungsgruppe auf eine Kerneinheit (im Sinne des [UWM92]) wird die Überschaubarkeit des zu betreuenden Teilgebietes gewährleistet.

Die Anforderungen der neuen Projektstruktur an die Organisation der Modelle im COMPOSER soll im weiteren betrachtet werden.

3.2.7.2 Organisation der Modelle

a) Gesamtmodell

Die R+V-Gesamtarchitektur ihrer Anwendungen wird in einem Gesamtmodell abgelegt. Die Entwicklung neuer Komponenten basiert wie bisher auf bereits bestehenden Datenmodellausschnitten und vorhandenen technischen Funktionen, die im Gesamtmodell für die Entwicklung der Komponenten und Anwendungen zur Verfügung gestellt werden.

b) Komponentenmodelle

Entsprechend der Organisation der Komponentenentwicklungsprojekte werden für jede Komponente getrennte Modelle in COMPOSER angelegt.

Zu jeder Komponente sind zwei Modelle anzulegen. Das erste ist das Spezifikationsmodell, in dem die Spezifikationen der einzelnen Operationen abzulegen sind. Das

Spezifikationsmodell dient zum Suchen und Abrufen der Spezifikationen für die Anwendungsentwicklung.

Als zweites Modell existiert für jede Komponente ein Implementierungsmodell. Dieses enthält die zu den Spezifikationen entwickelten Umsetzungen. Die aus den Implementierungen erzeugten Quelldateien werden abrufbar gegliedert nach Komponenten und Operationen zur Verfügung gestellt. Auf das Implementierungsmodell kann allein die Komponentenentwicklungsgruppe zugreifen, für Anwendungsentwickler bleibt die Implementierung der Komponente verborgen.

Wird zu einer existierenden Komponente eine neue Version erstellt so erhält diese ihre eigenen Modelle. Damit kann gewährleistet werden, daß die alte Komponente erhalten bleibt. Wiederverwendbare Bestandteile der alten Komponente werden in die entsprechenden Modelle der neuen Komponente kopiert. Somit behalten die Dienste der neuen Komponente die gleichen Namen und Identifikatoren und können bei gleichen Schnittstellen anstatt der alten Komponente verwendet werden.

Die Suche in den Modellen wird wesentlich erleichtert. Zum einen existieren bei der vorgeschlagenen Aufteilung acht unterschiedliche Komponenten, zu denen es eine überschaubare Anzahl aktueller Versionen geben wird. Aufgrund der eindeutigen Zuordnung von Diensten zu genau einer Kerneinheit ist das zu durchsuchende Modell einfach zu erkennen. Mit Hilfe des COMPOSER-Data Model Browser ist das Anzeigen der Dienste und der dazugehörigen Beschreibungen einfach möglich. Somit kann schnell und einfach entschieden werden, inwieweit die Anforderungen durch die jeweilige Komponente erfüllt werden.

c) Anwendungsmodelle

Jede Anwendung behält weiter wie bisher ihre eigenen Modelle für Entwicklung, Test und produktiven Einsatz. Die Veränderungen des Vorgehens haben keinen Einfluß auf die Organisation der Anwendungsmodelle.

3.3 Zu schaffende Voraussetzungen

Aufbauend auf die bereits geschaffenen Voraussetzungen für Wiederverwendung von Komponenten müssen weitere Maßnahmen getroffen werden.

3.3.1 Organisatorische Maßnahmen

Das neue Vorgehen stellt neue Anforderungen an die Organisation der Mitarbeiter. Wie schon teilweise umgesetzt müssen klare Trennungen zwischen Anwendungs- und Komponentenentwicklung umgesetzt werden. Die Durchsetzung der vorgeschlagenen Projektstruktur erfordert die Schaffung einer *neuen Organisationseinheit* zur Überwachung, Koordination und Steuerung der komponentenbasierten Anwendungsentwicklung. Die Gruppe von Analytikern muß in der Lage sein, vorausschauend zu arbeiten und die Gesamtarchitektur der R+V-Versicherung zu überblicken. Darüber hinaus müssen die nötigen Entscheidungsbefugnisse für die Erfüllung ihrer Aufgaben gegeben sein. Eine wichtige Aufgabe ist die ständige Überwachung des Marktes im Hinblick auf wiederverwendbare Komponenten. Das schließt Aufbau und Pflege von Kooperationen mit anderen Unternehmen der Versicherungsbranche ein.
Weiterhin muß die Qualität der zur Verfügung gestellten Komponenten sichergestellt werden. Dazu müssen ausreichende Test- und Bewertungsrichtlinien und -verfahren entwickelt werden.
Eine weitere wichtige Voraussetzung ist die gezielte Aus- und Weiterbildung der Mitarbeiter im Hinblick auf Wiederverwendung von Komponenten. Die Einführung eines Belohnungssystems für erfolgreiche Wiederverwendung unterstützt eine schnelle Umsetzung und tritt soziologischen Hemmnissen entgegen.

3.3.2 Technische Maßnahmen

Wichtigste technische Voraussetzung für die Wiederverwendung von Komponenten ist ein leistungsfähiges Repository. Bis zur Verfügbarkeit eines kommerziellen Produktes kann durch die vorgeschlagenen Maßnahmen die Verwaltung der Komponenten mit den Möglichkeiten von COMPOSER gewährleistet werden. Die Berücksichtigung von Standards wie COM, CORBA oder OLE wird mit Einführung der neuen Version

COMPOSER 4 unterstützt. Wegen der weiteren Nutzbarkeit der mit COMPOSER 3 erstellten Software in der Nachfolgeversion geschieht dies auf Basis des verwendeten Werkzeuges.
Das in Abschnitt 3.2.1 geschaffene Bild einer Komponente im Umfeld der R+V-Versicherung muß weiterentwickelt werden, um größtmöglichen Nutzen aus der komponentenbasierten Anwendungsentwicklung ziehen zu können.

3.4 Zusammenfassung

Die komponentenbasierte Anwendungsentwicklung verspricht im untersuchten Umfeld der R+V-Versicherung wesentliche Verbesserungen der Software-Entwicklung zu ermöglichen.
Neben einer besseren Überschaubarkeit der Anwendungen und der möglichen parallelen Entwicklung der Komponenten und Anwendungen verspricht die vorgeschlagene Projektstruktur ein hohes Maß an Wiederverwendung. Daneben wird die Weiterentwicklung einer unternehmensweit einheitlichen Gesamtarchitektur unterstützt. Diese ist eine wichtige Grundlage für entscheidungsunterstützende Systeme, die eine Analyse der Geschäftssituation des Unternehmens ermöglicht.
Die verstärkte Modularisierung der Anwendungen und die daraus folgenden Eigenschaften, wie Flexibilität, Austauschbarkeit von Anwendungsteilen, Erweiterbarkeit und vor allem verbesserte Wartbarkeit führen zu einer neuen Qualität der Software.
Mir Hilfe der zusätzlichen Möglichkeiten der noch in diesem Jahr verfügbaren Version 4 des Werkzeuges COMPOSER werden Entwicklung und Wiederverwendung von Komponenten weiter erleichtert.

Anhang

Literaturverzeichnis

[AF93] Arnold, Robert S.; Frakes, William B.: Software Reuse and Reengieering in [Ar93]

[Ap95] Appelfeller, Wieland:
Wiederverwendung im objektorientierten Software-Entwicklungsprozeß, dargestellt am Beispiel der Entwicklung eines Lagerlogistiksystems;
Peter Lang, Europäischer Verlag der Wissenschaften, Frankfurt am Main, 1995

[Ap96] Appelfeller, Wieland: Phasenübergreifende Wiederverwendung durch den Einsatz von oo-Konzepten;
in: OBJEKTSpektrum 1/96, 28-37, Januar/ Februar 1996

[Ar93] Arnold, Robert S.: Software Reengineering
IEEE Computer Society Press, Los Alamitos, 1993

[Ba96] Baumöl, Ulrike et al.:
Einordnung und Terminologie des Software-Reengineering;
in Informatik-Spektrum 19: 191-195 (1996); Springer-Verlag, 1996

[BFB95] Biffl, Stephan; Futschek, Gerald; Brem, Christian: Ein Modell zur stufenweisen Umsetzung von Software-Wiederverwendung in der Praxis;
in: Informatik Foschung und Entwicklung 10/1995; 197-213; 1995

[Bö89] Börstler, Jürgen: Wiederverwendbarkeit und Softwareentwicklung - Probleme, Lösungsansätze und Bibliographie;
Aachener-Informatik-Berichte Nr. 89-5

[Bi95] Biedermann, Sven: Objektbauelemente und Objektbaugruppen für die Wiederverwendung von Software
Diplomarbeit an der Universität Leipzig: Fakultät Mathematik/ Informatik; Institut für Informatik, Leipzig, 1995

[CB93] Caldiera, Gianluigi; Basili, Victor R.: Identifying and Qualifying Reusable Software Components; in [Ar93]

[CE95] Carroll, Martin D.; Ellis, Margaret A.: Designing and Coding Reusable C++
Addison-Wesley Publishing Company Inc., New York; 1995

[Du93] Dumke, Reiner: Modernes Software Engineering - Eine Einführung
Friedrich Vieweg & Sohn Verlagsgesellschaft mbH, Braunschweig/Wiesbaden, 1994

[Ga96] Gamma, Erich et. al.:
Entwurfsmuster - Elemente wiederverwendbarer objektorientierter Software
Addison-Wesley, Bonn, 1996

[GDV96] Gesamtverband der Deutschen Versicherungswirtschaft e.V.:
VAA-Die Anwendungsarchitektur der Versicherungswirtschaft;
Gesamtverband der Deutschen Versicherungswirtschaft, Bonn, 1996

[Gö93] Gödicke, Michael: On the Structure of Software Description Languages: A Component Oriented View; Universität Dortmund - Fachbereich Informatik; Forschungsbericht Nr. 473/1993

[GM93] Garnett, E.S.; Mariani, J.A.: "Software Reclamations" in [Ar93]

[He93] Heß, Helge: Wiederverwendung von Software: Frameworks für betriebliche Informationssysteme; Betriebswirtschaftlicher Verlag Dr. Th. Gabler GmbH, Wiesbaden, 1993

[Jo95] Johnston, Ron: Benefits of Encapsulating IEF Composer Applications http://www. jmcsinc.com/articles/oo_encap.htm; 1995

[Ju96] de Judicibus, Dario: Reuse-a Cultural Change in [Sa96]

[Ka96] Kauba, Elisabeth: Wiederverwendung als Gesamtkonzept - Organisation, Methoden, Werkzeuge; in OBJEKTspektrum 1/96, 20-27, Januar/Februar 1996

[Ka97] Kauba, Elisabeth: Software-Re-Use ist eine Frage guter Organisation; in Computerwoche 2 97, 13-14, Januar 1997

[Kü94] Küffmann, Karin: Software Wiederverwendung
Friedrich Vieweg & Sohn Verlagsgesellschaft mbH, Braunschweig/Wiesbaden, 1994

[Li96] Lindner, Ulrich: Massive Wiederverwendung: Konzepte, Techniken und Organisation;
in: OBJEKTSpektrum 1/96, 10-17, Januar 1996

[McC93] McClure, Carma: Software-Automatisierung - Reengineering Repository Wiederverwendbarkeit; Eine Coedition der Verlage:
Carl Hanser Verlag München Wien
Prentice-Hall International Inc. London, 1993

[McI68] McIlroy, M.D.: Mass-produced Software-Components,
in: Software Engineering Concepts and Techniques,
NATO Conference Software Engineering, Garmisch, 1968

[MO94] Meyer, Hanns-Martin; Obermayr, Karl:
Objekte integrieren mit OLE2
Springer-Verlag, Berlin, 1994

[Ni96] Nietsch, Michael:
Wiederverwendungsorientierte Software-Entwicklung
Deutscher Universitäts Verlag GmbH, Wiesbaden, 1996

[NT95] Nierstrasz, Oscar; Tsichritzis, Dennis: Object-Oriented Software Composition
Prentice Hall, New York; 1995

[PB93] Pomberger, Gustav; Blaschek, Günther:
Software Engineering: Prototyping und objektorientierte Software-Entwicklung;
Carl Hanser Verlag, München, 1993

[Po93] Pomberger, Gustav:
Software-Engineering - Auf dem Wege vom Handwerk zur industriellen Fertigung von Software-Produkten ?;
in [FT93]

[Pr96] Prieto-Diaz, R.: Reuse as a New Paradigm for Software Development in [Sa96]

[Qu94] Quibeldey-Cirkel, Klaus:
Das Objekt-Paradigma in der Informatik
B.G. Teubner, Stuttgart, 1994

[Re95] Rezagholi, Mohsen: Programm zur schrittweisen Ausrichtung der Softwareerstellung auf Wiederverwendung;
in: Softwaretechnik-Trends 15:4; 38-43, November 1995

[Sa96] Sarshar, Marjan (Ed.): Systematic Reuse: Issues in Initiating and Improving a Reuse Program;
Proceedings of the International Workshop on Systematic Reuse, Liverpool, 8-9 January 1996; Springer-Verlag London Limited 1996

[Sh97] Short, Keith: Component Based Development and Object Modelling
Texas Instruments Software; Februar 1997

[Sn91] Sneed, Harry: Software Wartung
Verlagsgesellschaft Rudolf Müller GmbH, Köln; 1991

[SO97] Softlab GmbH:
http://www.softlab.de/german/news/doo.html, März 1997

[St91] Stritzinger, Alois: Reusable Software Components and Application Frameworks; Concepts, Design Principles and Implications
Dissertation an der Johannes Keppler Universität Linz; 1991

[TI95] Texas Instruments, Microsoft: Re-Engineering Application Development;
A Texas Instruments-Microsoft White Paper, Juli 1995

[TI96] Texas Instruments: Component-Based Development - Fundamentals;
Texas Instruments Incorporated; Juli 1996

[TIIP] Texas Instruments: Component-Based Development - Component-Based Vision; Texas Instruments Incorporated Internal Paper; 1995

[UWM92] Unternehmensweites Modell der R+V-Versicherung
Wiesbaden 1992

[You93] Yourden, Ed: Re-3 part I in [Ar93]

[Ze95] Zendler, A.: Konzepte, Erfahrungen und Werkzeuge zur Software-Wiederverwendung, Tectum Verlag, Marburg, 1995

[ZG95] Zendler, A.; Gastinger, S.: Vergleichende Analyse von Werkzeugen zum Aufbau und Einsatz von Bibliotheken für wiederverwendbare Software-Dokumente;
Tectum Verlag, Marburg, 1995

Anhang A

Glossar

Action Block	Action Blocks sind nach außen angeschlossene Funktionen in COMPOSER, die über Parameter gesteuert eine definierte Aufgabe erfüllen. Ein Action Block (AB) besteht aus einem Deklarationsteil, in dem vier Arten von Datensichten definiert werden (Importsichten, Exportsichten, lokale Sichten und Entity-Sichten). Die definierten Sichten werden im Implementierungsteil genutzt, um die Funktionalität des Action Block zu implementieren.
built-Schritt	Teilschritt der Anwendungsgenerierung, in dem die Quelldateien zur Gesamtanwendung verknüpft werden.
COMPOSER-Enzyklopädie (CE)	Zentraler Bestandteil des Werkzeuges COMPOSER, in dem Anwendungsmodelle verwaltet werden. Die CE bietet leistungsfähige Möglichkeiten zum Daten- und Funktionsaustausch zwische den verschiedenen Anwendungen.
Data Model Browser	Bestandteil des Werkzeuges COMPOSER, der Anzeige und Verwaltung von Daten- und Funktionen eines Modells ermöglicht.
Datensicht	Eine Datensicht ist eine Datenstruktur ähnlich einem Record, in der Variablen unterschiedlicher Typen zusammengefaßt werden. Diese sind semantisch zusammengehörend.
Domäne	Unter einer (Anwendungs-)Domäne versteht man einen Unternehmensbereich oder eine Sparte [BFB95], sie ist gekennzeichnet durch ähnliche Anwendungsstrukturen.
Migration	Kopiervorgang unter Beibehaltung der Informationen zur Quelle im Werkzeug COMPOSER

Objektdatenbank	Datenbank, in der alle Objekte gemäß der Definition des [UWM92] gespeichert werden.
Procedure Step	Ein Procedure Step (PS) stellt einen Teilprozeß dar, der aus einer Abfolge von Aktivitäten besteht. Der gesamte Teilprozeß kann wiederverwendet werden. Der Übergang zu einem Procedure Step erfolgt durch sogenannte *Flows* (nur eine Richtung) oder *Links* (mit anschließender Rückkehr) , bei denen Datenstrukturen übergeben werden.
Software-Dokument	Unter einem Software-Dokument soll analog zu [Ba96] jede Repräsentation von Zwischenergebnissen und Ergebnissen verstanden werden, die im Rahmen eines Software-Entwicklungsprozesses entstehen (z.B. Quellcode, konzeptionelles Datenschema, Handbücher).
Vorgehensmodell	Festlegungen über Ablauf und Organisation der Software-Erstellung
Workset	Unimplementierte Datenstruktur im Werkzeug COMPOSER, dient zur Datenübergabe und zur Darstellung von Daten auf dem Bildschirm, zu denen keine Datenbank existiert.

Anhang B

Implementierungsbeispiel eines Dienstes

```
Model : SG69-PARTNER-CBD-DIPLOM            19 Sept. 1997 10.51
Subset: (complete model)

           BSD Action Block: IKF_KFZ_SUCHEN_I
_____________________________________________________________________

Action Block Description:
Sucht Fahrzeuge mit gegebenen Kennzeichen oder Fahrgestellnummer oder zu
gegebenem Partner.
Die Ergebnisse werden als Trefferliste zur•ckgegeben.

 +- IKF_KFZ_SUCHEN_I
 |  IMPORTS:
 |   Work View  imp_kfz kfz (optional,transient,import only)
 |    fk_partner (optional)
 |    kennzeichen (optional)
 |  EXPORTS:
 |   Group View exp_lb_kfz (100,explicit,export only)
 |    Work View  exp_sel ief_supplied (transient)
 |     select_char
 |    Work View  exp_le_kfz kfz (transient)
 |     auspr„gung
 |     hp_klasse
 |     tk_klasse
 |     vk_klasse
 |     zulassungskreis
 |     hubraum
 |     leistung
 |     fk_partner
 |     sfklasse
 |     kennzeichen
 |     typ
 |     hersteller
 |     uid
 |   Work View  exp_kfz kfz (transient,export only)
 |    auspr„gung
 |    hp_klasse
 |    tk_klasse
 |    vk_klasse
 |    zulassungskreis
 |    hubraum
 |    leistung
 |    fk_partner
 |    sfklasse
 |    kennzeichen
 |    typ
 |    hersteller
 |    uid
 |   Work View  exp_component irg_component (transient,export only)
```

```
|    origin_servid
|    rollback_indicator
|    severity_code
|    reason_code
|    return_code
|  LOCALS:
|  ENTITY ACTIONS:                              :Datensichten für DB-Zugriffe
|   Entity View ent_hersteller hersteller
|    name
|    uid
|   Entity View ent_typ typ
|    baujahr_von
|    bezeichnung
|    baujahr_bis
|    uid
|   Entity View ent_variante variante
|    hp_klasse
|    vk_klasse
|    tk_klasse
|    lstg_ps
|    lstg_kw
|    bezeichnung
|    uid
|   Entity View ent_fahrzeug fahrzeug
|    fahrgestellnummer
|    status
|    uid
|    zulassungskreis
|    fk_partner
|    kennzeichen
|
| NOTE ************************************************************
|
|    Pre-Condition
|        Kennzeichen ist nicht leer.
|
|    Post Condition
|        a) alle KFZ mit diesem Kennzeichen in der Liste
|            return code: +0001
|
|        b) keine KFZ mit diesem Kennzeichen
|            return-code: +0002
|
|        c) genau ein KFZ mit gegebenen Kennzeichen
|         Treffer in exp_kfz
|            return-code: +0003
|
|    ************************************************************
|
|    Pre-Condition
|        Kennzeichen ist leer
|        Partner ist nicht leer
|
|    Post-Condition
|        a) Alle KFZ zu Partner in der Liste
|            return-code: +0001
```

```
 |
 |        b) Keine KFZ zu Partner
 |            return-code: +0002
 |
 |        c) genau ein Kfz zu Partner
 |         Kfz in exp_kfz
 |            return-code: +0003
 |
 |    ********************************************************
 | +- IF imp_kfz kfz kennzeichen IS NOT EQUAL TO SPACES
 | |    OR imp_kfz kfz fk_partner IS NOT EQUAL TO SPACES
 | | +- IF imp_kfz kfz kennzeichen IS NOT EQUAL TO SPACES
 | | | SET SUBSCRIPT OF exp_lb_kfz TO 0
 | | | += READ EACH ent_fahrzeug fahrzeug
 | | | |    WHERE DESIRED ent_fahrzeug fahrzeug kennzeichen IS EQUAL TO
 | | | |       imp_kfz kfz kennzeichen
 | | | | SET SUBSCRIPT OF exp_lb_kfz TO SUBSCRIPT OF exp_lb_kfz + 1
 | | | | +- READ ent_variante variante
 | | | | |    WHERE DESIRED ent_variante variante schlieát_ein
 | | | | |       CURRENT ent_fahrzeug fahrzeug
 | | | | +- WHEN successful
 | | | | | SET exp_le_kfz kfz auspr„gung TO ent_variante variante bezeichnung
 | | | | | SET exp_le_kfz kfz hp_klasse TO ent_variante variante hp_klasse
 | | | | | SET exp_le_kfz kfz leistung TO ent_variante variante lstg_ps
 | | | | | SET exp_le_kfz kfz tk_klasse TO ent_variante variante tk_klasse
 | | | | | SET exp_le_kfz kfz vk_klasse TO ent_variante variante vk_klasse
 | | | | +- WHEN not found
 | | | | | SET exp_component irg_component return_code TO "-0001"
 | | | <------ESCAPE
 | | | | +--
 | | | | +- READ ent_typ typ
 | | | | |    WHERE DESIRED ent_typ typ hat CURRENT ent_variante variante
 | | | | +- WHEN successful
 | | | | | SET exp_le_kfz kfz typ TO ent_typ typ bezeichnung
 | | | | +- WHEN not found
 | | | | | SET exp_component irg_component return_code TO "-0001"
 | | | <------ESCAPE
 | | | | +--
 | | | | +- READ ent_hersteller hersteller
 | | | | |    WHERE DESIRED ent_hersteller hersteller produziert
 | | | | |       CURRENT ent_typ typ
 | | | | +- WHEN successful
 | | | | | SET exp_le_kfz kfz hersteller TO ent_hersteller hersteller name
 | | | | +- WHEN not found
 | | | | | SET exp_component irg_component return_code TO "-0001"
 | | | <------ESCAPE
 | | | | +--
 | | | | SET exp_le_kfz kfz uid TO ent_fahrzeug fahrzeug uid
 | | | | SET exp_le_kfz kfz kennzeichen TO ent_fahrzeug fahrzeug kennzeichen
 | | | | SET exp_le_kfz kfz fk_partner TO ent_fahrzeug fahrzeug fk_partner
 | | | | SET exp_le_kfz kfz zulassungskreis TO ent_fahrzeug fahrzeug
 | | | |       zulassungskreis
 | | | +--
 | | | +- IF SUBSCRIPT OF exp_lb_kfz IS EQUAL TO 0
 | | | | SET exp_component irg_component return_code TO "+0002"
<------------ESCAPE
```

```
 | | | +--
 | | | +- IF SUBSCRIPT OF exp_lb_kfz IS EQUAL TO 1
 | | | | SET exp_component irg_component return_code TO "+0003"
 | | | | MOVE exp_le_kfz kfz TO exp_kfz kfz
<------------ESCAPE
 | | | +--
 | | | SET exp_component irg_component return_code TO "+0001"
 | | +--
 | +--
 +--
```

Versicherung

Ich versichere, die vorliegende Arbeit selbständig und unter ausschließlicher Verwendung der angegebenen Hilfsmittel angefertigt zu haben.

Leipzig, 01.12.1997