February 2007

Anwendungsarchitektur-Beurteilung unter Berücksichtigung zu erwartender Flexibilität

Felix R. Müller

In: Oberweis, Andreas, u.a. (Hg.) 2007. *eOrganisation: Service-, Prozess-, Market-Engineering*; 8. Internationale Tagung Wirtschaftsinformatik 2007. Karlsruhe: Universitätsverlag Karlsruhe

ISBN: 978-3-86644-094-4 (Band 1)
ISBN: 978-3-86644-095-1 (Band 2)
ISBN: 978-3-86644-093-7 (set)

Anwendungsarchitektur-Beurteilung unter Berücksichtigung zu erwartender Flexibilität

Felix R. Müller

Lehrstuhl für Controlling und Logistik
Universität Regensburg
93053 Regensburg
felix.mueller@wiwi.uni-regensburg.de

Abstract

Für die Gestaltung einer Anwendungsarchitektur existieren Richtlinien, die zu einer „guten" Abgrenzung von Anwendungssystemen führen sollen. Es ist jedoch noch unklar, wie die Güte einer Aufgabenzerlegung und -zuordnung beurteilt werden kann. In diesem Beitrag wird argumentiert, dass die Flexibilität einer Anwendungsarchitektur ein wichtiges Gestaltungsziel darstellt. Dieses Ziel ist im Software Engineering unter dem Namen *Wartbarkeit* bekannt. Ausgehend von der Annahme, dass komplexe Systeme bei sonst gleichem Leistungsumfang und bei gleichem Anpassungsbedarf einen größeren Wartungsaufwand verursachen, wird versucht, die Größen Kohäsion und Kopplung – für die im Software Engineering anerkannte Metriken existieren – für den Bereich der Gestaltung von Informationssystemen auf Anwendungssystemebene zu adaptieren.

1 Motivation und Fragestellung

Für die Bedienung von Informationsverarbeitungsbedarfen in einem Unternehmen stehen im Rahmen der Gestaltung der Anwendungsarchitektur mehrere Alternativen zur Verfügung. Die Auswahl einer architektonischen Alternative aus dem Lösungsraum hat Einfluss auf den Entwicklungs- und Einführungsaufwand, den Aufwand für den Betrieb der Anwendungssysteme und auf den Aufwand, der für Wartungstätigkeiten zu erwarten ist. Es ist deshalb notwendig, architektonisch verschiedene Lösungen für ein Anforderungsprofil beurteilen zu können, um

eine sinnvolle Auswahl zu treffen. In diesem Beitrag soll ein Werkzeug entwickelt werden, das eine solche Beurteilung ermöglicht.

2 Begrifflicher Hintergrund

Eine Anwendungsarchitektur ist ein Modell eines Informationssystems, das betriebliche IV-Anforderungen (Aufgaben) sowie (Anwendungs-)Softwaresysteme[1] beinhaltet, die jene realisieren. Ein Softwaresystem ist eine Zusammenfassung von Softwaremodulen, die zusammen mindestens eine Funktion erfüllen. Hier wird der Begriff *(Anwendungs-)Softwaresystem* verwendet, da der Begriff *Anwendungssystem* auch für das Ensemble mehrerer Softwaresysteme, der Aufgabenrealisierung und ggfs. menschlicher Aufgabenträger verwendet wird. Zwischen diesen Softwaresystemen können Integrationsbeziehungen bestehen. Ein Dienst wird durch funktionale und nicht-funktionale Anforderungen spezifiziert. Funktionale Anforderungen legen fest, welche zu automatisierenden Aufgaben ein Dienst beinhaltet, nicht-funktionale Anforderungen stellen eine Einschränkung des Lösungsraums dar, da sie festlegen, unter welchen Bedingungen die Leistung erbracht werden muss. Es lassen sich nun Mengen von Software-Systemen bilden, die diese Anforderungen erfüllen (Sachziel der „Automatisierungs- und Integrationsaufgabe“ [FeSi01,220]). Architekturentscheidungen beziehen sich auf die Größe der Softwaresysteme[2] und die Art der Integrationsbeziehungen zwischen den Softwaresystemen [Mert04,20].

3 Anwendungsbereich

Die Beurteilung alternativer gültiger architektonischer Lösungen für ein Anforderungsprofil auf Anwendungssystemebene lässt sich als Teilaufgabe des Integrationsmanagements betrachten, das die „Zusammenfassung aller mit der Integration verbundenen Aufgaben“ [Rose99,5] darstellt. Das Problem lässt sich der Gestaltung eines Informationssystems im Rahmen der Aufgabenzerlegung und der Zuweisung zu automatisierender Aufgabenbestandteile auf Aufgabenträger zuordnen [FeSi01,215-217]. Ist ein „Neuaufwurf“ bestehender Anwendungssysteme und deren Integrationsbeziehungen untereinander zulässig, ist eine derartige Beurteilung jedoch

[1] Da nur Anwendungs-Softwaresysteme betrachtet werden, wird im Folgenden vereinfachend das Präfix weggelassen.
[2] Die Größe eines Softwaresystems lässt sich zunächst über die Anzahl der durch dieses automatisierten Aufgabenbestandteile messen; für eine präzisere Betrachtung s. u.

auch sinnvoll, da durch architektonische Entscheidungen auf Anwendungsebene mitunter eine bessere Zielerreichung realisiert werden kann.

4 Ziele

Zu Zielen der Gestaltung der Anwendungsarchitektur existiert kein universell anerkannter Katalog. Als wichtiges Teilziel im Rahmen des Integrationsmanagements wird der „optimale Integrationsgrad“ [Rose99,13] behandelt. Ferstl entwickelt – aufbauend auf einer groben Formalziel-Klassifizierung der „Gestaltungsaufgabe Automatisierung von IS“ – Einzelziele der Integration [Fers92,9–14], die jedoch sowohl innerhalb der Klasse als auch klassenübergreifend von Ausprägungen anderer Einzelziele abhängig sein können. Schwinn argumentiert, dass ein optimaler Integrationsgrad schwer zu ermitteln sei und gruppiert fünf Gestaltungsziele, die diesen beeinflussen. Als Oberziel dieser Gestaltungsziele führt er „Agilität“ ein [ScWi05; HaSc06]. Es bleibt jedoch offen, ob die jeweiligen „Gruppen“ vollständig und abgeschlossen sind und wie die Zielerreichung sinnvoll gemessen werden kann. Huber et al. erwähnen „Faktoren“, denen zukünftige Anwendungsarchitekturen genügen müssen; diese sind Flexibilität, Integrationsfähigkeit und Standardisierung [HABÖ00,173–174].

Sowohl Flexibilität als auch Agilität lassen sich unter dem Begriff Anpassungsfähigkeit subsumieren. Flexibilität ist die Fähigkeit eines Systems, auf system- oder umweltinduzierte Veränderungen sinnvoll zu reagieren [Kalu93; Steg04]. Für die Anwendungsebene eines Informationssystems lässt sich Flexibilität als die Fähigkeit beschreiben, geänderte Informationsverarbeitungsbedarfe zu befriedigen (vgl. [Dunc95; GeLe05,44]) oder sich sonstigen Umweltveränderungen (z. B. der Infrastruktur) anzupassen.[3] Für den Bereich der Applikationsintegration ist Agilität „die ‚Leichtigkeit‘(…), mit der sich Änderungen oder Erweiterungen [an der Anwendungslandschaft] durchführen lassen“ [HaSc06,277].

Für den Bereich des Software-Designs ist Wartbarkeit (engl. *maintainability* [IEEE90,46]) ein besonders relevantes Qualitätskriterium, da sich Architektur-Entscheidungen darauf auswirken, in welchem Umfang mögliche Änderungen an einer Komponente Anpassungen anderer Systemelemente bedingen. Synonym wird der Begriff Änderbarkeit (engl. *modifiability* [BCKB98; BBKM78]) verwendet. Beide Begriffe beschreiben, wie einfach sich Änderungen an einem System vollziehen lassen, um Fehler zu beseitigen, die Leistungsfähigkeit zu erweitern oder um

[3] Für die Infrastruktur können ähnliche Betrachtungen angestellt werden, s. z. B. [Pfau97,73].

das System an eine veränderte Umwelt anzupassen. Diese Anwendungsbereiche korrespondieren mit einer gängigen Klassifizierung von Wartungstätigkeiten (*corrective*, *perfective*, *adaptive* [Scha05,480]).

Es verhält sich also so, dass für die Gestaltung von Software-Systemen und die eines Informationssystems auf Anwendungsebene ein ähnliches Ziel formuliert ist: Durch eine sinnvolle Gestaltung des jeweiligen Systems sollen potentielle Anpassungen möglichst günstig durchgeführt werden können.[4] Maßnahmen, die im Lebenszyklus eines Informationssystems auf Anwendungssystemebene u. U. notwendig sind, lassen sich als Wartungstätigkeiten[5] in eine der oben vorgestellten Kategorien einordnen; also lässt sich dieses Qualitätskriterium für die Gestaltung eines Informationssystems verwenden (s. a. [Ande02,26]).

Es ließe sich argumentieren, dass diese Reduktion auf Wartbarkeit als einziges Qualitätsziel eine unstatthafte Vereinfachung darstelle. Bezugnehmend auf den Katalog in [Fers92,11–14] ist festzustellen:

- Das Formalziel *Korrektheit* ist nicht weiter skaliert: Eine architektonische Variante, die die funktionalen Anforderungen an den Dienst nicht korrekt erfüllt, ist ungültig. Es sollen hier ausschließlich gültige architektonische Lösungen beurteilt werden.

- Das *Echtzeitverhalten* kann – analog zur Softwareentwicklung [Scha05,277f.] – als nicht-funktionale Anforderung verstanden werden, die den Lösungsraum bereits vorab einschränkt.

- *Integration* kann als Formalziel nicht unabhängig von Flexibilität behandelt werden, da die Ausprägungen der Zielerreichung für Einzelziele der Integration maßgeblich die Flexibilitätseigenschaften einer Lösung beeinflussen. So sind beispielsweise anpassende Wartungstätigkeiten bei minimaler Funktionsredundanz c. p. mit minimalem Aufwand durchführbar (s. auch [HaSc06,281]).

Das Wirtschaftlichkeitsziel für Umsetzung und Durchführung[6] ist nicht vernachlässigbar. Systeme, die jeweils eine andere Anwendungsarchitektur aufweisen, können sich in den Kosten für Implementierung und Betrieb – Aktivitäten, die *nicht* Wartungstätigkeiten sind – unterscheiden:

[4] Für die Quantifizierung des zu erwartenden Aufwands für Anpassungen stehen verschiedene Ansätze zur Verfügung. Vereinfachend werden hier z. B. Personen-Stunden angenommen.
[5] Genauer: Wartungs-, Systemverbesserungs- und Systemanpassungsmaßnahmen [Kore72,200].
[6] Fünftes Formalziel im o. g. Katalog, ähnlich [ScWi05]: „Minimale Projektaufwände für die Integration."

Eine Alternative mag z. B. den Betrieb spezieller Infrastrukturkomponenten erforderlich machen, wohingegen eine andere nur durch den Einsatz versierter Experten realisiert werden kann. Dadurch, dass mögliche durchzuführende Anpassungen nicht vollständig im Voraus bekannt und damit nicht bewertbar sind, können diese den Entwicklungs- und Betriebskosten nicht gegenübergestellt werden. Nichtsdestotrotz existieren Beiträge, die die Eintrittswahrscheinlichkeiten aller denkbaren Änderungen der Informationsverarbeitungsbedarfe als bekannt voraussetzen und diese in einem Entscheidungsmodell verarbeiten (z. B. [GeSc06]) oder abstrakte Turbulenzmaße verwenden und qualitative Betrachtungen dokumentieren (z. B. [HiSH05]).

5 Beurteilung anhand zu erwartender Wartbarkeit

Der Aufwand, der für zukünftige mögliche Wartungstätigkeiten anfällt, lässt sich nur schwer abschätzen, da die Anpassungsbedarfe nicht vollständig antizipierbar sind. Deshalb muss auf Indikatoren zurückgegriffen werden, die für Wartbarkeit signifikant sind.

5.1 Indikatoren zur Abschätzung der Wartbarkeitseigenschaften eines Systems

Im Software Engineering werden Wartbarkeit und Einfachheit als proportionale Größen angesehen [StMC74,115]. Um die steigende Komplexität von Softwareprodukten beherrschbar zu machen, wurden in den 1970er Jahren Prinzipien, Methoden und Werkzeuge entwickelt, mit denen sich eine Strukturierung des zu gestaltenden Systems durchführen lässt, u. a. die Modularisierung, also die Zusammenfassung abgeschlossener funktionaler Einheiten mit definierten Schnittstellen (vgl. [Balz98,571–574]). Komplexe Systeme durch Modularisierung und Abstraktion verständlicher zu machen ist jedoch kein Instrument, das spezifisch für die Softwareentwicklung ist; vielmehr handelt es sich um eine Vorgehensweise, die in vielen Disziplinen thematisiert wird [Alex64; Simo65]. Auf den allgemeinen Fall übertragen handelt es sich hierbei um das Problem der Abgrenzung von Subsystemen [Ulri68,107–109].

Für die Modularisierung der Anwendungslandschaft stehen Methoden zur Verfügung, die aus einer Zuordnung von Funktionen oder Prozessen zu Informationsobjekten Softwaresystemkandidaten und deren Datenaustauschbeziehungen untereinander ableiten und somit automatisch zu einer gültigen Zerlegung und Zuordnung führen (z. B. [Gags71] oder als Teil des Business Systems Planning [IBM 84]). Der Verwendung einer derartigen Vorgehensweise für den Vergleich von architektonischen Lösungen stehen jedoch zwei Probleme entgegen: Erstens ist nicht nach-

gewiesen, dass eine Aufgabenzuordnung, die Resultat der Anwendung der Methode auf ein Anforderungsprofil ist, zwingend eine maximale Wartbarkeit gewährleistet. Zweitens bleibt offen, wie alternative architektonische Lösungen verglichen werden können, da keine Abstandsmaße für verschiedene Zerlegungen und Beziehungen definiert sind.

Einfache – und dadurch günstig wartbare – Systeme zeichnen sich durch lose gekoppelte Subsysteme aus; das System ist „beinahe zerlegbar“ [Simo65,69–76].[7] Änderungen eines Subsystems wirken sich minimal auf andere Subsysteme aus [WaWe90]. Hohe Kohäsion innerhalb der Subsysteme fördert sowohl eine lose Kopplung [StMC74] als auch eine gute Wartbarkeit der Subsysteme selbst. Diese Eigenschaften, Kopplung und Kohäsion, können dazu verwendet werden, die Ergebnisse alternativer Modularisierung zu beurteilen. Im Software Engineering werden zusätzlich zu Metriken, die sich auf Kohäsion und Kopplung beziehen, weitere Maße zur Bestimmung struktureller Komplexität vorgeschlagen[8], Kohäsion und Kopplung nehmen jedoch eine zentrale Rolle ein.[9]

5.2 Messung von Kohäsion und Kopplung

Es ist nun zu untersuchen, wie Kohäsion und Kopplung in einem Informationssystem auf Anwendungsebene gemessen werden können. Dazu soll erfasst werden, welche Mittel im Software Engineering Anwendung finden. Als Grundlage werden typische Einteilungen verwendet, die sich auf Softwareentwicklung nach dem prozeduralen Programmierparadigma beziehen.

5.2.1 Kohäsion

Die Eigenschaft *Kohäsion* beschreibt den Bezug der Elemente innerhalb eines Moduls. Die Größe gibt an, wie stark die Aufgaben, die ein Modul realisiert, zusammenhängen.[10] Die Eigenschaft ist ordinal skaliert [StMC74,121], es existieren jedoch Metriken zur Bestimmung verschiedener Kohäsionsgrade für prozedural organisierte Programme (z. B. [BiOt94]). Die folgende Darstellung orientiert sich an einer gängigen Einteilung verschiedener Kohäsionsgrade (vgl. [Myer78;Scha05,170–175]); diese sind nach steigender Intensität geordnet (*informational cohesion* ist also die erstrebenswerteste Form der Kohäsion).

[7] „nearly decomposable“, Übersetzung übernommen aus der deutschen Ausgabe [Simo94].
[8] Für eine Übersicht s. [Keme95].
[9] „Any study of past reasearch on structural complexity of software leads to the conclusion that coupling and cohesion are fundamental underlying dimensions“ [DKST05,983].
[10] „The manner and degree to which the tasks performed by a single software module are related to one another.“ [IEEE90,17].

Operationen in Modulen entsprechen betriebliche Aufgabenbestandteile in Softwaresystemen: Beide repräsentieren aus Anforderungssicht bestimmte sinnvolle Funktionen. Zwar ist die Abgrenzung von Operationen bereits Bestandteil des Softwareentwicklungsprozesses, die Ableitung zu automatisierender Aufgabenbestandteile ist jedoch auch zentral für die Gestaltung von Informationssystemen.

Zufällige Kohäsion liegt demnach vor, wenn in einem Softwaresystem Aufgaben zusammengefasst sind, die in keinem fachlichen Zusammenhang stehen (z. B. *Lohnnebenkosten ermitteln* und *Stapler disponieren*). Analog zur Argumentation im Software Engineering ist zufällige Kohäsion als schlechteste Art des Zusammenhalts in einem Softwaresystem zu betrachten: Sind Änderungen an einer Aufgabenrealisierung im Softwaresystem, ist davon eine fachlich unabhängige Funktion betroffen. Ferner ist zufällige Kohäsion ein Indikator für die Kopplung zwischen Softwaresystemen, sofern nicht davon ausgegangen werden muss, dass sämtliche Aufgaben isoliert voneinander, also unabhängige Prozesse sind. Es kann dann geschlossen werden, dass mindestens eine Kopplungsbeziehung zu einem anderen Modul existieren muss.

In einem Softwaresystem liegt logische Kohäsion dann vor, wenn die Aufgabenbestandteile zwar logisch zusammenhängen (z. B. *Belege erstellen* als Modul, das für sämtliche Arten von Belegen Funktionalität bereitstellt), innerhalb des Moduls jedoch starke Abhängigkeiten existieren. Diese Form der Kohäsion ist problematisch, da Funktionen im Softwaresystem starke Abhängigkeiten aufweisen, so dass bspw. bei Änderungen in der Druckausgabe auch Anpassungen der Bildschirmanzeige erforderlich sind. Außerdem lässt sich wie beim Vorliegen zufälliger Kohäsion auf eine ungünstige Kopplung zwischen den Softwaresystemen schließen (unnötige Parameter, Kontrollkopplung).

Werden Aufgaben in einem Softwaresystem zusammengefasst, deren Bearbeitung zeitlich zusammenfällt, liegt zeitliche Kohäsion vor (z. B. *Wareneingang buchen* und *Stapler disponieren*). Diese Zusammenfassung ist als ungünstig zu betrachten, da das zeitliche Zusammentreffen selbst Änderungen unterworfen ist. Werden Geschäftsprozesse umgestaltet, ist der zeitliche Bezug zweier Aufgaben möglicherweise nicht mehr gegeben. Schließt man die Möglichkeit aus, neue Softwaresysteme zu bilden, die dem neuen Ablauf Rechnung tragen, so können Softwaresysteme, die ursprünglich zeitliche kohärent gestaltet sind zu solchen mit zufälliger Kohäsion degenerieren. Eine analoge Argumentation lässt sich für Systeme mit ablauforientierter Kohäsion anführen, in denen Aufgaben zusammengefasst sind, deren sequentielle Durchfüh-

rung in einem Teilprozess festgelegt ist. Deshalb soll diese Unterscheidung im Folgenden vernachlässigt werden.

Im Unterschied zur ablauforientierten Kohäsion gilt für kommunikative Kohäsion (*communicational cohesion*) zusätzlich die Einschränkung, dass alle Operationen auf gemeinsamen Ein- oder Ausgabedaten operieren (z. B. *Sendung erfassen* und *Packstück-Labels erstellen*). Systeme mit kommunikativer Kohäsion sind einfacher als Systeme, in denen ablauforientierte Kohäsion vorherrscht; die Wartungsfreundlichkeit (des Moduls) ist – ohne weitere Beeinflussung der Kopplung – größer, da das System c. p. kleiner ist, jedoch keinen zusätzlichen komplexen Kommunikationsaufwand verursacht. Bei Änderungen am Datenmodell sind – verglichen mit zeitlicher und prozeduraler Kohäsion – weniger Funktionen betroffen. Deshalb kann diese Kohäsionsart auch auf Anwendungsebene nicht mit den beiden vorher genannten zusammengefasst werden.

Realisiert ein Subsystem genau eine Aufgabe, so liegt funktionale Kohäsion vor (z. B. *Stapler disponieren*). Diese zählt für die klassische Softwareentwicklung zu den besten Kohäsionsarten, da neben der Wartbarkeit durch erleichterte Fehlersuche und einfache Austauschbarkeit auch die Wiederverwendung positiv beeinflusst wird. Die positive Bewertung lässt sich auf Anwendungssystemebene jedoch nicht ohne weiteres übernehmen, da noch keine Aussagen über die Ansteuerung der Einzelfunktionen oder über Datenhaltungsaspekte getroffen werden.

Schließlich können Aufgaben, die voneinander unabhängig auf demselben Datum operieren, in Softwaresystemen zusammengefasst werden (*informational cohesion*). Diese Art der Kohäsion liegt beispielsweise abstrakten Datentypen und Objekten im Rahmen des objektorientierten Programmierparadigmas zu Grunde, wenn diese gut gestaltet sind. Bei der Übertragung dieses Konzepts auf die Anwendungsarchitektur entsteht pro „Informationsobjekt“ oder „Resource Business Element“ [MSJL06,85–88] ein Softwaresystemkandidat, in dem sämtliche Aufgaben, die vorrangig auf dieser Information operieren, zusammengefasst sind (z. B. *Lieferscheineintrag erstellen* und *Lieferschein abschließen*). Auch für diese Art der Kohäsion gilt. dass die Einteilung ohne eine Betrachtung der Ablauflogik nicht beurteilt werden kann. Diese kann über alle Subsysteme verteilt oder in einem oder mehreren separaten Modulen implementiert sein; je nach Realisierung ist ein derartiges System mehr oder weniger komplex.

In Tabelle 1 sind die behandelten Kohäsionsarten übersichtsartig dargestellt. In der Spalte *Art (SW)* sind die Arten aus der Software-Entwicklung aufgeführt; in der Spalte *Art (IS)* die zu unterscheidenden Arten für die Beurteilung von Anwendungssystemen.

Art (SW)	Beschreibung	Art (IS)
Coincidental	Modul beinhaltet unzusammenhängende Operationen.	Coincidental
Logical	Modul beinhaltet zusammenhängende Operationen, die vom aufrufenden Modul angesteuert werden.	Logical
Temporal	Modul beinhaltet Operationen, deren Aufruf zeitlich zusammenfällt.	} Temporal
Procedural	Modul beinhaltet Operationen, deren Aufruf in einer bestimmten Reihenfolge zusammenfällt.	
Communicational	Modul beinhaltet Operationen, deren Aufruf in einer bestimmten Abfolge zusammenfällt und die auf den selben Daten operieren.	Communicational
Functional	Modul führt genau eine Operation aus.	Functional
Informational	Modul beinhaltet voneinander unabhängige Operationen, die auf denselben Daten operieren.	Informational

Tabelle 1 Kohäsionsarten

5.2.2 Kopplung

Für den Bereich der Softwareentwicklung ist Kopplung definiert als ein Maß der Verknüpfungsstärke, die aus einer Verbindung zweier Module erwächst.[11] Diese Definition bezieht sich auf Entwicklung nach dem prozeduralen Programmierparadigma. Ein Modul im weiteren Sinne ist eine Darstellung einer funktionalen Einheit oder einer semantisch zusammengehörenden Funktionsgruppe, die abgeschlossen ist, definierte Schnittstellen für Externbezüge besitzt und „im qualitativen und quantitativen Umfang handlich, überschaubar und verständlich“ ist [Balz98,572]. Der Kopplungsgrad zwischen Modulen hängt von der Komplexität der Schnittstelle, des Verbindungsbezugs (Modul als solches oder Element innerhalb des Moduls) sowie der Art der Nachricht ab [StMC74,117–121]. Im Folgenden wird eine gängige Einteilung von Kopplungsarten (nach abnehmendem Kopplungsgrad sortiert) aufgegriffen (vgl. [Your79]) – *content coupling* ist also als stärkste Form der Kopplung am wenigsten erstrebenswert.

Es sollen nun Beziehungen *zwischen* Softwaresystemen untersucht werden. Auch hier wird angenommen, dass diese mit den Modulen des Software Engineering korrespondieren: Ein Softwaresystem beinhaltet also die Implementierung mindestens eines Aufgabenbestandteils und kann mit anderen Softwaresystemen in Beziehung stehen. Die Stärke der Beziehung wird durch den Grad der Kopplung angegeben.

Inhaltliche Kopplung liegt vor, wenn ein Softwaresystem direkt ein Datum oder Funktionalität eines anderen Softwaressystems ändert. Dies ist zum Beispiel der Fall, wenn ein Softwaresystem *Konstruktion* eine interne Variable im Softwaresystem *Behälterplanung* überschreibt. Dieses Design macht selbst bei internen Änderungen im referenzierten Softwaresystem, die also

[11] „The measure of strength of association established by a connection from one module to another“ [StMC74,117].

nicht die externen Schnittstellen der Software betreffen, Anpassungen der referenzierenden Softwaresysteme erforderlich.

Für Anwendungsarchitektur besonders relevant ist die Kopplung über gemeinsame Datenbanken (*common coupling*), wobei mehrere Softwaresysteme die enthaltenen Daten ändern können (z. B. „Datenintegration“ [Fers92,19–21]). Aus Sicht des Software Engineering wird diese Kopplungsart als Indikator für ein schlechtes Design angesehen, da bei Änderung der Deklaration einer globalen Variable alle Funktionen, die diese Variable verwenden, angepasst werden müssen, da Sicherheitsprobleme auftreten können und da sich die Anzahl derartiger Kopplungsbeziehungen eines Moduls ohne Änderungen am Modul selbst ändern kann („clandestine common coupling“ [SJWH03]). Diese Probleme gelten für Anwendungsarchitekturen analog: Änderungen am Datenschema können Wartung gekoppelter Softwaresysteme erforderlich machen und die Anzahl an Softwaresystemen, die an einer solchen Beziehung teilnehmen, kann sich ändern, ohne dass diese Veränderung für ein bestimmtes Softwaresystem offensichtlich ist. Rückverfolgbarkeits- und Sicherheitsprobleme können ebenso auftreten.

Kontrollkopplung (*control coupling*) liegt vor, wenn ein Softwaresystem über einen Parameter beim Aufruf eines anderen Softwaresystems Einfluss auf den Kontrollfluss nimmt. Dies ist z. B. dann der Fall wenn ein Softwaresystem *Produktionssteuerung* das Softwaresystem *Auftragsfreigabe* aufruft und dieses für einen Auftrag eine Meldung zurückgibt, die dem aufrufendem System die Änderung der Auftragsgröße vorschreibt. Dazu muss das aufgerufene System Kenntnis von der Funktionsweise des aufrufenden Moduls haben. Ändert sich die Logik im aufrufenden Subsystem, sind Anpassungen des aufgerufenen Systems erforderlich.

Werden beim Aufruf eines Softwaresystems als Parameter unnötigerweise komplexe Informationsobjekte übergeben, liegt Datenbereichskopplung (*stamp coupling*) vor. In der Softwareentwicklung ist dies zu vermeiden, da die Schnittstellenkomplexität steigt, die Wiederverwendbarkeit sinkt und Sicherheitsprobleme auftreten können. Allerdings ist anzumerken, dass die schlechte Bewertung einer Datenbereichskopplung dadurch verstärkt wird, dass in einigen verbreiteten Programmiersprachen (z. B. Java) komplexe Strukturen normalerweise als Referenz und einfache Datentypen als Wert übergeben werden, so dass zusätzliche Komplexität dadurch entsteht, dass eine Rückverfolgbarkeit der Änderungen an einer Struktur oder einem Objekt durch häufige „Weitergabe“ erschwert wird. Diese Unterscheidung muss also auch auf Anwendungssystemebene getroffen werden, um eine Beurteilung dieses Musters durchführen zu können. Wird beispielsweise einem Softwaresystem *Versand* zur Erstellung der Versandbelege als

Argument ein Informationsobjekt *Kunde* übergeben, so sind Statistiken zum Bestellverhalten des Kunden für die Ausführung der Aufgabe zunächst unerheblich. Dennoch hat das Softwaresystem in diesem Falle Zugriff auf den kompletten Kundenstammdatensatz. Müssen Änderungen der Kennzahlen zum Bestellverhalten am Informationsobjekt nachvollzogen werden, so muss in diesem Fall auch das Softwaresystem *Versand* untersucht werden.

Erhält ein Softwaresystem beim Aufruf einer Funktion genau die Informationsobjekte, die zur Erfüllung der Aufgabe erforderlich sind, unterhält dies mit dem aufrufenden System eine Beziehung vom Typ Datenkopplung (*data coupling*). Hierbei bleibt zunächst unberücksichtigt, ob ein System zur Durchführung der Aufgabe auf andere Datenbestände zurückgreift, die jedoch vom aufrufenden System nicht verändert werden. Datenkopplung wird im Software Engineering als Indikator für ein gutes Design betrachtet.[12] In Tabelle 2 sind die Kopplungsarten zusammengefasst (Darstellung analog zu Tabelle 1).

Art (SW)	Beschreibung	Art (IS)
Content	A referenziert direkt den Inhalt von B.	Content
Common	A und B haben Zugriff auf ein globales Datum.	} External
External	A und B kommunizieren über eine gemeinsame Datei.	
Control	A ruft B mit einer Nachricht auf, um den Kontrollfluss in B zu beeinflussen.	Control
Stamp	A übergibt B unnötigerweise eine komplexe Datenstruktur	Stamp
Data	A übergibt B eine einfache Variable oder sinnvollerweise eine komplexe Datenstruktur	Data

Tabelle 2 Kopplungsarten

6 Beispiel

Als Anwendungsbeispiel dient ein Geschäftsprozess, der im Rahmen des Forschungsprojekts „Supra-adaptives Logistiknetzwerk Automobilwirtschaft in Bayern“ erhoben wurde. Aus diesem Prozess wurden zu automatisierende Aufgabenbestandteile abgeleitet, die verschieden realisiert werden können. Im Folgenden werden zwei Alternativen dargestellt und der oben entwickelten Beurteilung unterworfen.

6.1 Problem

Einige Logistikdienstleister (LDL), die an der Schnittstelle zwischen 1-st-Tier-Zulieferer und Hersteller (OEM) in der Automobilindustrie agieren, sehen sich mit der Situation konfrontiert, dass Zulieferer Packstücke (PS) mit VDA-Warenanhänger [VDA 94] übergeben, auf denen Barcodes in einem proprietären Format aufgedruckt sind. Für die Abwicklung beim LDL ist es

[12] „Data coupling is a desirable goal.“ [Scha05,180].

notwendig, im Wareneingang jedes PS zusätzlich mit einem eigenen Label zu versehen. Es ist also sinnvoll, das PS-Label bereits initial mit einer vom LDL automatisiert lesbaren PS-ID und mit einer Routing-Angabe zu versehen. Der Prozess ist in Abbildung 1 zusammengefasst.

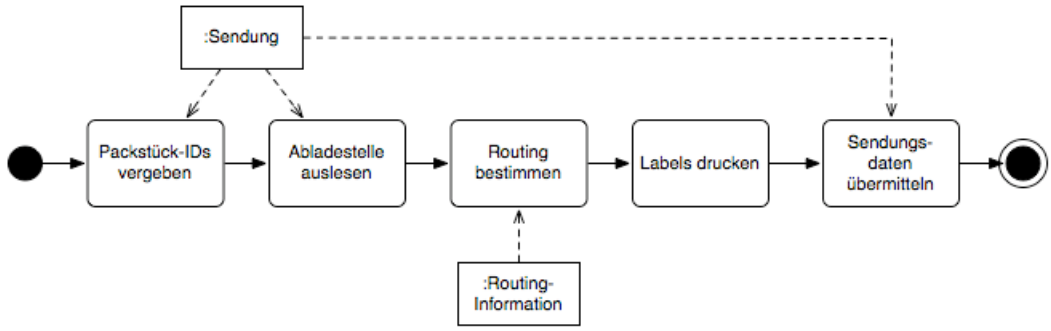

Abbildung 1 Prozess Packstückbelabelung

6.2 Alternative 1

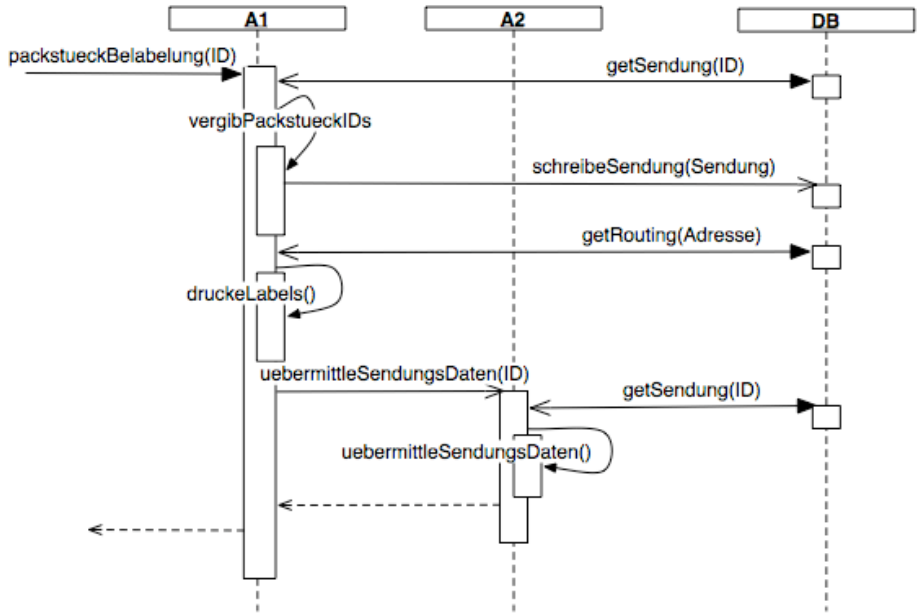

Abbildung 2 Umsetzung Packstückbelabelung – Alternative 1

Der erste Realisierungsvorschlag umfasst zwei Softwaresysteme (vgl. Abbildung 2): In A1 sind sämtliche Funktionen außer der Übermittlung der Sendungsdaten (A2) zusammengefasst. Für A1 liegt zeitliche Kohäsion vor: Die enthaltenen Funktionen sind aus einer Aufrufabfolge heraus betrachtet zusammenhängend, operieren jedoch auf unterschiedlichen Ein- bzw. Ausgabedaten. A2 beinhaltet genau eine Funktion, ist also funktional kohärent. A1 speichert die erstellten PS-IDs im Sendungs-Objekt ab. A2 liest diese Daten aus und übermittelt sie an ein Softwaresystem beim Logistikdienstleister. Es liegt also externe Kopplung über eine Datenbank vor.

6.3 Alternative 2

In der zweiten Variante werden drei Softwaresysteme gebildet (vgl. Abbildung 3): A1 für die Vergabe der PS-IDs, A3 für die Bestimmung der Zielniederlassung und A2 für die Durchführung der sonstigen Aufgaben. A1 und A3 sind funktional kohärent (genau eine Aufgabe wird durchgeführt), für A2 liegt maximal zeitliche Kohäsion vor (Aufgaben fallen im Prozessmodell zusammen); da A2 eine Aufgabe im Ablauf übernimmt und somit den zeitlichen Zusammenhang für A3 *unterbricht*, ließe sich auch zufällige Kohäsion unterstellen. Es wird jedoch angenommen, dass A3 keine weiteren Aufgaben realisiert; der Zusammenhang ist somit über die Realisierung von Aufgaben für *einen* Geschäftsprozess – PS-Belabelung nämlich – hergestellt. Dem Softwaresystem A1 wird eine Sendung übergeben, für die PS-IDs generiert werden. Diese werden zurückgegeben. A3 liest das Sendungsobjekt übergibt die Abladestelle an das Softwaresystem A2, das Labels erstellt und die Sendungsdaten übermittelt. A1 erhält als Eingabedatum ein Sendungsobjekt, aus dem es jedoch nur bestimmte Daten ausliest; es liegt also Datenbereichskopplung vor. Als Rückgabewerte werden PS-IDs geliefert (Datenkopplung). A3 wird mit der Abladestelle als Argument aufgerufen; auch dieses Datum ist minimal für die Durchführung der Aufgabe. Liefert A3 lediglich das Kürzel der Empfangsniederlassung, liegt auch für die Rückgabe Datenkopplung vor.

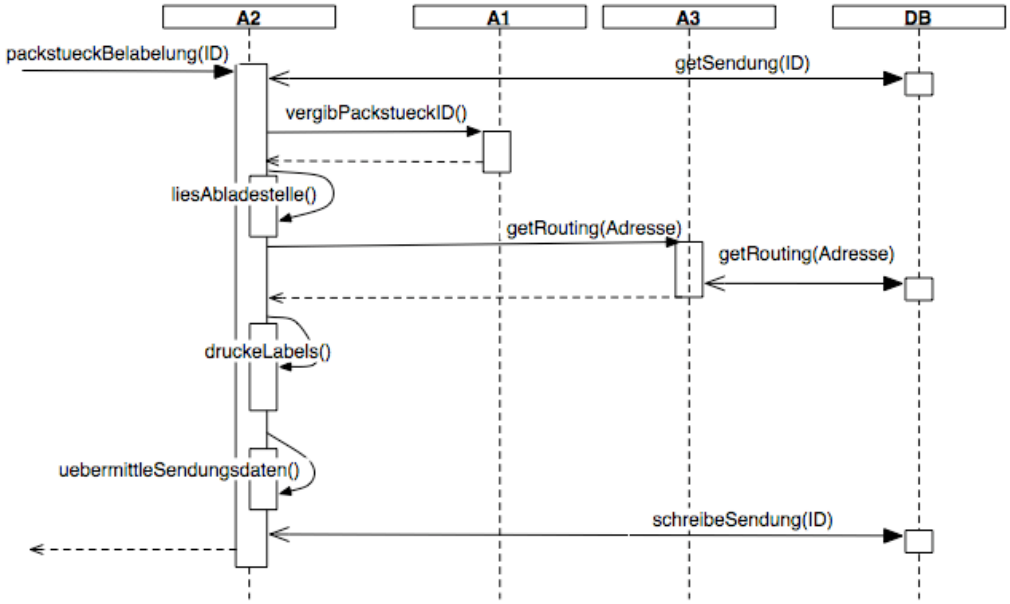

Abbildung 3 Umsetzung Packstückbelabelung – Alternative 2

Diese architektonische Alternative ist gemäß einem realistischen Anwendungsfall konstruiert worden: Ein LDL stellt seinen Kunden zur Abwicklung von Transporten über sein Netzwerk bisweilen ein eigenes Softwaresystem zur Verfügung, das über eine Schnittstelle, die vom Ver-

sandsystem angesprochen werden kann, für eine Adresse die entsprechende Empfangsniederlassung zurückliefert.
Dadurch, dass sich kein Softwaresystem durch eine schlechtere Kohäsion als ein Softwaresystem in der ersten Variante auszeichnet, und einige Instanzen externer Kopplung vermieden werden, ist die zweite Variante aus architektonischer Sicht vorzuziehen.

7 Zusammenfassung und weitere Schritte

Aufbauend auf bestehenden Ansätzen zur Beurteilung architektonischer Alternativen wurde argumentiert, dass Flexibilität mit dem Konzept der Wartbarkeit aus der Domäne der Softwareentwicklung korrespondiert. Es wird dort davon ausgegangen, dass Wartbarkeit mit Strukturkomplexität korreliert ist. Deshalb wurde versucht, Metriken zur Messung von Kohäsion und Kopplung auf die Ebene der Anwendungsarchitektur zu übertragen. Einige Stufen der jeweiligen Eigenschaften konnten dabei zusammengefasst werden, da eine weitere Unterscheidung für die Gestaltung von Anwendungssystemen nicht sinnvoll ist.
Die Beurteilung erfolgt bislang qualitativ und bedarf einer Einschätzung beteiligter Interessenträger. Das vorliegende Anwendungsbeispiel ist sehr einfach gehalten: Es gibt keinerlei Überschneidung, die eine Abwägung erforderlich machen würde. Es ist mit den vorgestellten Metriken also bislang nicht möglich, nicht-triviale Anwendungsfälle zu vergleichen. Um diese Probleme bearbeiten zu können, bietet sich beispielsweise der Einsatz des Analytic Hierarchy Process' (AHP, [Saat80]) als Instrument zur Entscheidungsunterstützung an.
Da die Anwendung von Integrationskonzepten einen Einfluss auf die Aufgabenzuordnung und die Verknüpfung betrachteter Softwaresysteme hat, soll abgeschätzt werden, welche Ausprägungen für die Eigenschaften Kohäsion und Kopplung des Ergebnisses einer Gestaltung gemäß dem jeweiligen Konzept möglich sind. Des Weiteren soll ermittelt werden, welchen Einfluss die Anwendung von Referenzmodellen und Instrumenten der strategischen Informationssystem-Planung auf die oben vorgestellten Eigenschaften eines zu erwartenden „Endprodukts" hat.

Literatur

[Alex64] *Alexander, Christopher*: Notes on the Synthesis of Form, Harvard University Press, Cambridge (USA) 1964.

[Ande02] *Andersson, Jonas*: Enterprise Information Systems Management: An Engineering Perspective Focusing on the Aspects of Time and Modifiability. Dissertation, Department of Electrical Engineering, KTH, Stockholm (Schweden), 2002.

[Balz98] *Balzert, Helmut*: Lehrbuch der Software-Technik, Spektrum, Heidelberg 1998.

[BBKM78] *Boehm, Barry W.; Brown, John R.; Kaspar, Hans; Myron, Lipow; MacLeod, Gordon J.; Merritt, Michael J.*: Characteristics of Software Quality, North-Holland, Amsterdam (NL) 1978.

[BCKB98] *Bas, Len; Clements, Paul; Kazman, Rick; Bass, Ken*: Software Architecture in Practice, Addison-Wesley, Boston (USA) 1998.

[BiOt94] *Bieman, James M.; Ott, Linda M.*: Measuring Functional Cohesion. In: IEEE Trans. Softw. Eng. 20 (1994) 8, S. 644–657.

[DKST05] *Darcy, David P.; Kemerer, Chris F.; Slaughter, Sandra A.; Tomayko, James E.*: The Structural Complexity of Software: Testing the Interaction of Coupling and Cohesion. In: IEEE Trans. Softw. Eng. 31 (2005) 11, S. 982–995.

[Dunc95] *Duncan, Nancy Bogucki*: Capturing Flexibility of Information Technology Infrastructure: A Study of Resource Characteristics and their Measure. In: JMIS 12 (1995) 2, S. 37–57.

[Fers92] *Ferstl, Otto K.*: Integrationskonzepte betrieblicher Anwendungssysteme (Fachbericht Informatik 1/1992), Koblenz 1992 .

[FeSi01] *Ferstl, Otto K.; Sinz, Elmar J.*: Grundlagen der Wirtschaftsinformatik. 4. Aufl., Oldenbourg, München 2001.

[Forl06] Bayerischer Forschungsverbund supra-adaptive Logistiksysteme (Forlog): Supra-adaptives Logistiknetzwerk Automobilwirtschaft in Bayern, http://www.forlog.de, Abruf 2006-10-24.

[Gags71] *Gagsch, Siegfried*: Probleme der Partition und Subsystembildung in betrieblichen Informationssystemen. In: *Grochla, Erwin; Szyperski, Norbert* (*Hrsg.):* Management-Informationssysteme – Eine Herausforderung an Forschung und Entwicklung. Gabler, Wiesbaden 1971, S. 623–652.

[GeLe05] *Gebauer, Judith; Lee, Fei*: Towards an Optimal Level of Information System Flexibility – A Conceptual Model. In: *Bartmann, Dieter et al.* (*Hrsg.):* Proceedings of the 13th ECIS. Regensburg 2005. CD-ROM Edition.

[GeSc06] *Gebauer, Judith; Schober, Franz*: Information System Flexibility and the Cost Efficiency of Business Processes. In: JAIS 7 (2006) 3, S. 122–147.

[HABÖ00] *Huber, Thomas; Alt, Rainer; Barak, Vladimir; Österle, Hubert*: Future Application Architecture for the Pharmaceutical Industry. In: *Österle, Hubert; Fleisch, Elgar; Alt, Rainer* (*Hrsg.):* Business Networking. Springer, Berlin 2000, S. 163–183.

[HaSc06] *Hagen, Claus; Schwinn, Alexander*: Measured Integration – Metriken für die Integrationsarchitektur. In: *Schelp und Winter* [ScWi06], S. 267–292.

[HiKL95] *Hirschheim, Rudy; Klein, Heinz K.; Lyytinen, Kalle*: Information Systems Development and Data Modeling – Conceptual and Philosophical Foundations, Cambridge University Press, Cambridge (UK) 1995.

[HiSH05] *Hilhorst, Cocky; Smits, Martin; van Heck, Eric*: Strategic Flexibility and IT Infrastructure Investments – Empirical Evidence in two Case Studies. In: *Bartmann, Dieter et al. (Hrsg.)*: Proceedings of the 13th ECIS. Regensburg 2005. CD-ROM Edition.

[IBM 84] IBM (Hrsg.): Business Systems Planning – Information Systems Planning Guide (GE20-0527-4). 4. Aufl., IBM, Armonk (USA) 1984.

[IEEE90] *IEEE Institute of Electrical and Electronics Engineers*: IEEE Standard Glossary of Software Engineering Terminology, IEEE Std 610.12-1990, 02/1991.

[Kalu93] *Kaluza, Bernd*: Flexibilität, betriebliche. In: *Wittmann, Waldemar; et al.* (*Hrsg.):* Handwörterbuch der Betriebswirtschaft. Schäffer-Poeschel, Stuttgart 1993, Bd. 1, S. 1173–1183. 5. Aufl.

[Keme95] *Kemerer, Chris F.*: Software complexity and software maintenance: A survey of empirical research. In: Annals of Software Engineering 1 (1995) 1, S. 1–22.

[Kore72] *Koreimann, Dieter S.*: Systemanalyse, de Gruyter, Berlin 1972.

[Mert04] *Mertens, Peter*: Integrierte Informationsverarbeitung, Bd. 1. 14. Aufl., Gabler, Wiesbaden 2004.

[MSJL06] *McGovern, James; Sims, Oliver; Jain, Ashish; Little, Mark*: Enterprise Service Oriented Architectures – Concepts, Challenges, Recommendations, Springer, Berlin 2006.

[Myer78] *Myers, Glenford J.*: Composite, structured design, Van Nostrand, New York (USA) 1978.

[Pfau97] *Pfau, Wolfgang*: Betriebliches Informationsmanagement – Flexibilisierung der Informationsinfrastruktur, Markt- und Unternehmensentwicklung. DUV Gabler, Wiesbaden 1997. zugl. Habil. Universität Freiburg.

[Rose99] *Rosemann, Michael*: Gegenstand und Aufgaben des Integrationsmanagements. In: *Scheer, August-Wilhelm; Rosemann, Michael; Schütte, Reinhard* (*Hrsg.):* Integrationsmanagement (Arbeitsbericht Nr. 65). Institut für Wirtschaftsinformatik, Münster 1999, S. 5–18.

[Saat80] *Saaty, Thomas L.*: The analytic hiearchy process – Planning, priority setting, resource allocation. McGraw-Hill, New York (USA) 1980.

[Scha05] *Schach, Stephen R.*: Object-oriented and Classical Software Engineering. 6. Aufl., McGraw-Hill, New York (USA) 2005.

[ScWi05] *Schwinn, Alexander; Winter, Robert*: Entwicklung von Zielen und Messgrößen zur Steuerung der Applikationsintegration. In: *Ferstl, Otto K.; Sinz, Elmar J.; Eckert, Sven; Isselhorst, Tilman* (*Hrsg.):* Wirtschaftsinformatik. Physica-Verlag, Heidelberg 2005, S. 587–606.

[ScWi06] Schelp, Joachim; Winter, Robert *(Hrsg.)*: Integrationsmanagement, Springer, Berlin 2006.

[Simo65] *Simon, Herbert A.*: The Architecture of Complexity. In: *van Bertalanffy, Ludwig; Rapoport, Anatol* (*Hrsg.):* General Systems. Society for General Systems Research, Bedford (USA) 1965, Bd. 10, S. 63–76.

[Simo94] *Simon, Herbert A.*: Die Wissenschaft vom Künstlichen. 2. Aufl., Springer, Wien 1994.

[SJWH03] *Schach, Stephen R.; Jin, Bo; Wright, David R.; Heller, Gillian Z.; Offutt, Jeff*: Quality Impacts of Clandestine Common Coupling. In: Software Quality Journal 11 (2003) 3, S. 211–218.

[Steg04] *Steger, Hans-Diego*: Systemflexibilität – Konzeption und Gestaltungsansätze einer systemorientierten Unternehmensflexibilität, Eberhard, München 2004.

[StMC74] *Stevens, Wayne P.; Myers, Glenford J.; Constantine, Larry L.*: Structured Design. In: IBM Systems Journal 13 (1974) 2, S. 115–139.

[Ulri68] *Ulrich, Hans*: Die Unternehmung als produktives soziales System – Grundlagen der allgemeinen Unternehmungslehre, Paul Haupt, Bern 1968.

[VDA 94] *VDA Verband der Automobilindustrie*: Warenanhänger (barcodefähig); VDA-Empfehlung 4902 V.4. http://www.vda.de/de/service/bestellung/downloads/4902.pdf, Abruf 2006-07-06.

[WaWe90] *Wand, Yair; Weber, Ron*: An Ontological Model of an Information System. In: IEEE Trans. Softw. Eng. 16 (1990) 11, S. 1282–1292.

[Your79] *Yourdon, Edward*: Structured Design: Fundamentals of a Discipline of Computer Program and Systems Design, Prentice Hall, Englewood Cliffs (USA) 1979.