Ilmenauer Beiträge zur Wirtschaftsinformatik

Herausgegeben von U. Bankhofer, V. Nissen
D. Stelzer und S. Straßburger

Michael Lüttich, René Fiege

Anwendung von Axiomatic Design für den Entwurf Serviceorientierter Architekturen

Arbeitsbericht Nr. 2008-02, April 2008

Technische Universität Ilmenau
Fakultät für Wirtschaftswissenschaften
Institut für Wirtschaftsinformatik

Autor: Michael Lüttich, René Fiege

Titel: Anwendung von Axiomatic Design für den Entwurf Serviceorientierter Architekturen

Ilmenauer Beiträge zur Wirtschaftsinformatik Nr. 2008-02, Technische Universität Ilmenau, 2008

ISSN 1861-9223

ISBN 978-3-938940-20-4

Anschrift: Technische Universität Ilmenau, Fakultät für Wirtschaftswissenschaften, Institut für Wirtschaftsinformatik, PF 100565, D-98684 Ilmenau.
http://www.tu-ilmenau.de/fakww/Ilmenauer_Beitraege.1546.0.html

Gliederung

Abbildungsverzeichnis

Tabellenverzeichnis

Abkürzungsverzeichnis

AD	Axiomatic Design
DET	Data Element Type
DP	Designparameter
EPK	ereignisgesteuerte Prozesskette
FA	funktionale Anforderung
IFPUG	International Function Point Users Group
IT	Informationstechnologie
KA	Kundenanforderung
M	Modul
OMG	Object Management Group
PV	Prozessvariable
RET	Record Element Type
S	Service
SO	Serviceoperation
SOA	serviceorientierte Architektur
UML	Unified Modeling Language

Zu den Architekturzielen der Serviceorientierung gehören u. a. eine geringe Kopplung, hohe Autonomie, angemessene Granularität sowie gute Wiederverwendbarkeit und Komponierbarkeit von Services. Zurzeit existieren jedoch noch keine standardisierten Vorgehensmodelle zur Entwicklung serviceorientierter Architekturen, welche die Erreichung der genannten Architekturziele explizit berücksichtigen und fördern. In diesem Punkt besitzt die Entwurfsmethodik Axiomatic Design das Potenzial, die Entwicklung einer SOA in den Phasen der Analyse und des Entwurfs wesentlich zu unterstützen. Axiomatic Design entstammt ursprünglich dem technischen Bereich (Mechanical Engineering), ist aber so generisch gestaltet, dass es den strukturierten Entwurf beliebiger Objekte und Systeme ermöglicht. Gegenstand dieses Berichtes ist es deshalb, Axiomatic Design für den Entwurf serviceorientierter Architekturen anzupassen und anzuwenden. Hierfür werden die Entwurfskonstrukte und Entwurfsschritte von Axiomatic Design in die SOA-Welt übertragen und die der Methode zugrunde liegenden Prinzipien geeignet für den Architekturentwurf interpretiert. Das erarbeitete Vorgehen wird an einem konkreten Beispiel demonstriert, wobei eine Möglichkeit zur Darstellung von SOA-Entwürfen mit der Unified Modeling Language entwickelt wird. Abschließend wird der Beitrag von Axiomatic Design im Entwurf serviceorientierter Architekturen kritisch geprüft. Hierzu wird der Einfluss der Methode auf die Qualität der Entwurfsergebnisse im Hinblick auf Kopplung, Kohäsion und Granularität der Services untersucht. Für diese Analyse werden im Rahmen eines Messkonzepts Komplexitätsmetriken für die Beurteilung der drei genannten Aspekte definiert.

Schlüsselworte: Axiomatic Design, Serviceorientierte Architekturen, Entwurf, Architekturziele

1 Einleitung

1.1 Problemstellung

Die betriebliche Informationstechnologie (IT) wird durch die Dynamik der globalisierten Wirtschaft vor immer neue Herausforderungen gestellt. IT-Systeme müssen möglichst flexibel und kosteneffizient auf Veränderungen in den Geschäftsprozessen reagieren können.[1] Gleichzeitig sollen sie eine Vielzahl von Schnittstellen anbieten. In der aktuellen Diskussion um Möglichkeiten, diesen Anforderungen entgegen zu treten, stehen serviceorientierte Architekturen (SOAs) vielfach im Mittelpunkt. Die ihnen zugrunde liegenden Architekturprinzipien, wie z. B. die lose Kopplung, Wiederverwendbarkeit und Autonomie der Services, sind bewährte Grundsätze des Software Engineering. Das Konzept der Serviceorientierung wurde mittlerweile in diversen Veröffentlichungen behandelt.[2] Es existieren bereits erste Ansätze für den Entwurf und die Implementierung serviceorientierter Architekturen.[3] Bislang wurde dem Einfluss solcher Vorgehensmodelle auf die Erreichung der oben genannten Architekturprinzipien jedoch wenig Aufmerksamkeit geschenkt.

1.2 Zielsetzung

Mit Axiomatic Design (AD) liegt eine Methodik für den Entwurf beliebiger Objekte bzw. Systeme vor. Sie basiert auf der expliziten Formulierung funktionaler Anforderungen und einem strukturierten Vorgehen beim Entwurf. Durch die Anwendung von Axiomatic Design sollen entkoppelte, modulare und robuste Lösungen entstehen. Diese Vorzüge von AD sollen auf den SOA-Entwurf übertragen werden. Ziel des vorliegenden Berichtes ist es deshalb, Axiomatic Design für die Konzeption serviceorientierter Architekturen anzupassen und anzuwenden. Die resultierende Vorgehensweise soll bezüglich ihrer Praktikabilität und ihres Beitrags zur Erfüllung ausgewählter Architekturziele kritisch

---

1 Vgl. Richter, Haller, Schrey /Serviceorientierte Architektur/ 413

2 Vgl. beispielsweise Erl /Service-Oriented Architecture/; Krafzig, Banke, Slama /Enterprise SOA/; Bieberstein u. a. /SOA Compass/ oder Marks, Bell /Planning and Implementation/

3 Vgl. z. B. Erl /Service-Oriented Architecture/ 355-611 und Krafzig, Banke, Slama /Enterprise SOA/ 277-307

geprüft werden. Hierzu soll ein Messkonzept entwickelt werden, das die Erreichung der Architekturziele quantitativ bewertet.

1.3 Methodik

Damit Axiomatic Design die Entwicklung einer SOA unterstützen kann, müssen seine Entwurfsschritte entsprechend angepasst und konkretisiert werden. Die Konstrukte von Axiomatic Design und deren Beziehungen müssen so übertragen werden, dass sie auf die Elemente einer serviceorientierten Architektur anwendbar sind. Weiterhin müssen die Grundkonzepte (Axiome) von AD im Sinne der Architekturprinzipien der Serviceorientierung interpretiert werden. Schließlich soll der Beitrag von Axiomatic Design für die Erreichung der Architekturziele quantitativ untersucht werden. Hierfür werden entsprechende Softwaremetriken zur Messung der Zielerfüllung definiert.

1.4 Aufbau

In diesem Bericht wird im zweiten Kapitel zunächst die Entwurfsmethode Axiomatic Design vorgestellt. Beginnend mit einer kurzen Betrachtung der Grundlagen dieser Methodik werden anschließend die verwendeten Entwurfselemente und -prinzipien sowie die Schritte des Entwurfsprozesses beschrieben. Im dritten Kapitel dieses Berichtes werden die Bestandteile einer SOA und deren Beziehungen untereinander sowie ein Strukturierungskonzept dargestellt. Außerdem werden die grundlegenden Architekturprinzipien der Serviceorientierung skizziert.

Das vierte Kapitel beschreibt die Anpassung von Axiomatic Design für den Entwurf serviceorientierter Architekturen. Zuerst werden bereits bestehende Vorgehensmodelle für die Entwicklung einer SOA gewürdigt, um im Anschluss daran AD in die Phasen eines solchen Modells einzuordnen. Nachfolgend wird ausgeführt, wie die einzelnen Bestandteile von Axiomatic Design für die Ausgestaltung einer SOA übertragen werden können. Das erarbeitete Vorgehen wird im fünften Kapitel an einem praktischen Beispiel angewendet.

Im sechsten Kapitel erfolgt die kritische Bewertung des Einsatzes von Axiomatic Design im Entwurf serviceorientierter Architekturen. Hierzu wird die Qualität der Entwurfsergebnisse messbar gemacht, wofür einige der im dritten Kapitel behandelten Architekturziele als Kriterien herangezogen werden. Zur Einschätzung der Zielerreichung werden geeignete Metriken entwickelt und ihre Anwendung am Fallbeispiel demonstriert.

Abschließend werden die Vorteile von Axiomatic Design und mögliche Probleme betrachtet. Der Bericht endet mit einer Zusammenfassung des Inhalts, einer kritischen Auswertung der Ergebnisse sowie einem Ausblick auf künftige Entwicklungen.

2 Axiomatic Design

2.1 Grundlagen des Axiomatic Design

Axiomatic Design wurde Ende der 70er Jahre am Massachusetts Institute of Technology entwickelt[4] und basiert maßgeblich auf den Arbeiten von Suh[5]. Die bis dahin praktizierten Entwurfsmethoden für den technisch-mechanischen Bereich beruhten oft auf der Erfahrung und Intuition des Designers sowie iterativen und kostspieligen Trial-and-Error-Prozessen.[6] Axiomatic Design bietet hingegen eine wissenschaftliche Grundlage für den strukturierten Entwurf beliebiger Objekte. Kerngedanken dieser Methodik sind die funktionale Sichtweise auf den Entwurf und die explizite Formulierung von Entwurfszielen. Die theoretische Basis von AD umfasst ein Domänenkonzept sowie Grundprinzipien guter Entwürfe in Form des Unabhängigkeits- und Informationsaxioms. Diese Axiome[7] stellen Kriterien für die Beurteilung von Entwurfsentscheidungen zur Verfügung. Obwohl Axiomatic Design zunächst hauptsächlich im Umfeld des Maschinenbaus angewendet wurde, ist eine Übertragung auf andere Bereiche wie die Softwareentwicklung und Organisationsgestaltung möglich und bereits erfolgreich vorgenommen worden.[8]

In den folgenden Abschnitten dieses Kapitels werden zunächst das Domänenkonzept und die Entwurfselemente von Axiomatic Design vorgestellt. Anschließend werden die Schritte des Entwurfsprozesses mit AD erläutert. Im letzten Abschnitt erfolgt eine detaillierte Beschreibung der Entwurfsprinzipien in Form des Unabhängigkeits- und Informationsaxioms.

---

4 Vgl. zur Historie von Axiomatic Design Suh /Principles/ 17-22

5 Vgl. Suh /Principles/ und Suh /Axiomatic Design/

6 Vgl. für die folgenden Sätze Suh /Axiomatic Design/ 2-6

7 Ein Axiom ist ein „als absolut richtig anerkannter Grundsatz ... [, der] keines Beweises bedarf“ (Wermke, Kunkel-Razum, Scholze-Stubenrecht /Fremdwörterbuch/ 114).

8 Vgl. Suh /Axiomatic Design/ 1

2.2 Entwurfselemente von Axiomatic Design

Axiomatic Design unterscheidet strikt zwischen den Zielen eines Entwurfs in Form von Anforderungen (Was soll erreicht werden?) und der konkreten Problemlösung (Wie soll es erreicht werden?).[9] Diese Trennung wird durch ein Domänenkonzept realisiert. Jede Domäne begrenzt eine eigenständige Entwurfsaktivität (vgl. Bild 2-1). Während des Entwurfsprozesses werden alle Domänen durchlaufen. Bei der Abbildung von einer Domäne auf die nachfolgende formuliert die Ausgangsdomäne das Entwurfsziel (Was?) und die Zieldomäne die entsprechende Lösung (Wie?). Durch diesen Abbildungsprozess werden den jeweils formulierten Anforderungen korrespondierende Lösungen zugeordnet und damit der Entwurf schrittweise konkretisiert.

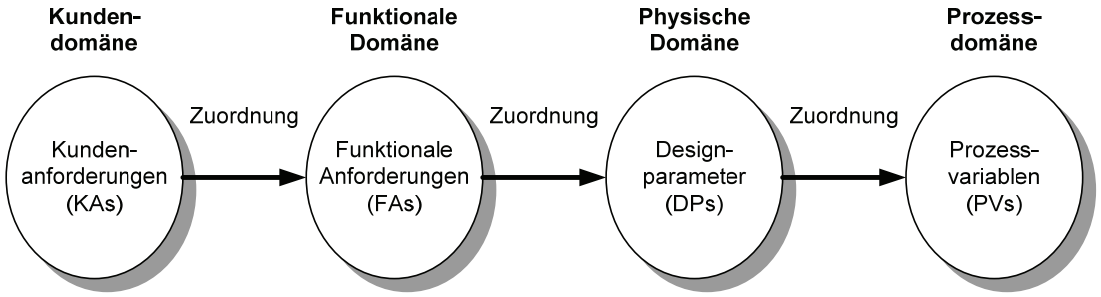


Bild 2-1: Domänen in Axiomatic Design[10]

Innerhalb der Kundendomäne werden zunächst die *Kundenanforderungen (KAs)* an den zu entwickelnden Entwurf erfasst und geordnet.[11] Für die Ermittlung der Forderungen und Wünsche der Kunden können bewährte Methoden wie das Quality Function Deployment oder die Conjoint-Analyse eingesetzt werden.[12]

Beim Übergang zur funktionalen Domäne werden den Kundenanforderungen entsprechende *funktionale Anforderungen (FAs)* zugeordnet. Die FAs beschreiben die Forderungen an den Entwurf aus einer funktionalen Sicht.[13] Sie können zusätzlich durch Restriktionen ergänzt werden. *Restriktionen* definieren Grenzen, z. B. hinsichtlich physischer Parameter oder nichtfunktionaler Merkmale, die durch konkrete

9 Vgl. für diesen Absatz Suh /Axiomatic Design/ 10 f.
10 Quelle: Suh /Axiomatic Design/ 11
11 Vgl. Suh /Axiomatic Design/ 14
12 Vgl. Kurniawan, Zhang, Tseng /Customers/ 2 f.
13 Vgl. Suh /Axiomatic Design/ 14

Entwurfslösungen eingehalten werden müssen.[14] Die Gesamtheit aller funktionalen Anforderungen sollte möglichst minimal sein und sich auf die für den Entwurf wesentlichen Kundenforderungen beschränken.[15]

Zur Erfüllung der FAs werden in der physischen Domäne *Designparameter (DPs)* einer konkreten Lösung ermittelt. Ein Designparameter beschreibt eine Eigenschaft oder einen Bestandteil des Entwurfsergebnisses zur Realisierung der betrachteten funktionalen Anforderung.[16] Zur Umsetzung dieser Lösung sind schließlich in der Prozessdomäne *Prozessvariablen (PVs)* festzulegen. Sie charakterisieren das Vorgehen zur Erstellung oder Realisierung der Entwurfslösung.[17] Der vorliegende Bericht konzentriert sich ausschließlich auf die Zuordnung von Designparametern zu funktionalen Anforderungen und damit auf den Übergang von der funktional-konzeptionellen zur physischen Domäne.

Nach Abschluss des Entwurfs ist dieser vollständig durch seine FAs, DPs und PVs spezifiziert. Zur Umsetzung der Entwurfslösung wird der „Herstellungsprozess“ vollzogen, der durch die Prozessvariablen beschrieben ist, und die Designparameter werden realisiert. Die Eigenschaften des entstehenden Systems sind die konkreten *Ausprägungen* der DPs. Anhand des vorliegenden Systems kann auch die Erfüllung der funktionalen Anforderungen untersucht werden, indem die Ausprägungen der FAs ermittelt werden. Man unterscheidet somit zwischen dem konzeptionellen Entwurf als abstrakter Spezifikation und dem realisierten Entwurf in Form eines konkret vorliegenden Systems (physisches Produkt, Softwareimplementierung usw.).

2.3 Phasen des Entwurfsprozesses

Der Entwurf mit Axiomatic Design vollzieht sich in mehreren Schritten mit iterativen Rücksprüngen. Der Ablauf ist in Bild 2-2 in Form eines V-Modells veranschaulicht. Dieses V-Modell gilt allgemein für jeden Entwurfsprozess mit Axiomatic Design und ist nicht mit dem V-Modell der Softwareentwicklung[18] verwandt.

Ausgangspunkt jedes Entwurfs ist die Ermittlung der Kundenanforderungen. Diese werden

14 Vgl. Suh /Axiomatic Design/ 21
15 Vgl. Suh /Principles/ 39 und 52 f. (Korollar 2)
16 Vgl. Suh /Axiomatic Design/ 14
17 Vgl. Suh /Axiomatic Design/ 14
18 Vgl. Balzert /Software-Qualitätssicherung/ 101 ff.

nach bestimmten Kriterien geordnet,[19] beispielsweise entsprechend ihrer Wichtigkeit. Eine Dekomposition der meist abstrakt formulierten Forderungen ist i. d. R. nicht notwendig.[20]

Anschließend erfolgt die Übertragung der Kundenwünsche in funktionale Anforderungen und Restriktionen.[21] Hierbei ist insbesondere darauf zu achten, dass die funktionalen Anforderungen lösungsneutral, also nicht im Hinblick auf bereits existierende Lösungen, formuliert werden. Die Menge funktionaler Anforderungen kann anhand von organisatorischen Kriterien, Funktionen oder Prozessabläufen des Entwurfsgegenstands gegliedert werden. Die Reihenfolge der FAs resultiert aus ihrer Wichtigkeit oder ihren Input-Output-Beziehungen. Für jede Anforderung kann bereits ein Zielwert und die erlaubte Streubreite für die FA-Ausprägungen festgelegt werden.

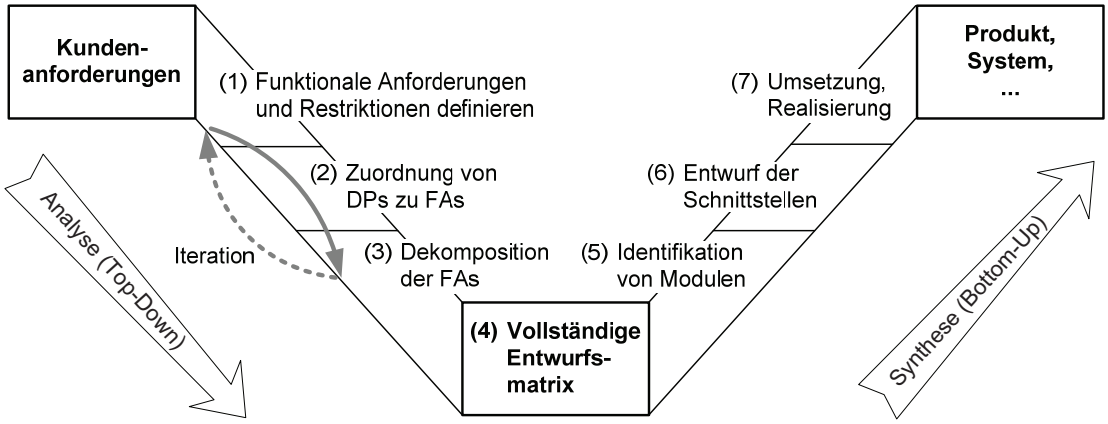


Bild 2-2: V-Modell von Axiomatic Design[22]

Der nächste Schritt im Entwurfsprozess umfasst die Zuordnung konkreter Designparameter zu den funktionalen Anforderungen.[23] Hierfür sind zunächst Ideen zu entwickeln, wie die jeweilige Anforderung durch einen Designparameter erfüllt werden könnte. Bei der Auswahl eines geeigneten DP aus diesen Vorschlägen ist zu beachten, dass dieser die Restriktionen der Anforderung einhält. Nach Möglichkeit sollte jede Anforderung durch einen eigenen Designparameter realisiert werden, so dass die Anzahl der FAs und DPs

19 Vgl. Suh /Axiomatic Design/ 14
20 Vgl. Suh /Axiomatic Design/ 22
21 Vgl. für diesen Absatz Suh /Axiomatic Design/ 15
22 Quellen: Suh /Axiomatic Design/ 267 und Lee, Suh /Design/ 39
23 Vgl. für diesen Absatz Suh /Axiomatic Design/ 18 f.

identisch ist.[24] Ein solcher Entwurf wird als idealer Entwurf bezeichnet, wenn er zusätzlich das Unabhängigkeitsaxiom erfüllt.

Ist die Zuordnung von Designparametern zu den funktionalen Anforderungen beendet, werden für die entstandene Hierarchieebene die Beziehungen zwischen FAs und DPs durch die folgende Entwurfsgleichung dokumentiert:[25]

$$\{FA\} = [A]\{DP\} \qquad \text{bzw.} \qquad \begin{Bmatrix} FA_1 \\ \vdots \\ FA_n \end{Bmatrix} = \begin{bmatrix} A_{11} & \cdots & A_{1n} \\ \vdots & \ddots & \vdots \\ A_{n1} & \cdots & A_{nn} \end{bmatrix} \begin{Bmatrix} DP_1 \\ \vdots \\ DP_n \end{Bmatrix} \tag{1}$$

Die *Entwurfsmatrix* [*A*] in Formel (1) stellt den Zusammenhang zwischen dem Vektor {*FA*} der funktionalen Anforderungen und dem Vektor {*DP*} der Designparameter her.[26] Die Zeilen der Matrix sind mit den entsprechenden FAs und die Spalten mit den DPs verbunden. In diesem Bericht wird von einer identischen Anzahl *n* von FAs und DPs ausgegangen, so dass die Entwurfsmatrix stets eine quadratische $n \times n$-Form besitzt. Das Element A_{ij}[27] der Matrix *A* beschreibt die Beziehung zwischen der funktionalen Anforderung FA_i und dem Designparameter DP_j. Besitzt das Matrixelement den Wert $A_{ij} = X$, so bedeutet dies, dass DP_j einen Einfluss auf FA_i ausübt.[28] Für $A_{ij} = O$ liegt hingegen keine Interdependenz zwischen DP_j und FA_i vor.[29] Die Elemente der Hauptdiagonale der Entwurfsmatrix sind stets besetzt.

Für die Bewertung einer Entwurfsmatrix werden das Unabhängigkeits- und das Informationsaxiom herangezogen (vgl. Abschnitt 2.4). Während das Unabhängigkeitsaxiom Regeln für die Zulässigkeit von Entwürfen definiert, bietet das Informationsaxiom ein quantitatives Maß für die Auswahl zwischen mehreren zulässigen Entwürfen.

---

24 Vgl. für diesen und den folgenden Satz Suh /Axiomatic Design/ 23 f.

25 Vgl. Suh /Principles/ 54 f.

26 Vgl. für diesen Absatz Suh /Axiomatic Design/ 18 f.

27 Die Positionsbeschreibung der Elemente folgt der Matrixnotation, wobei *i* für die Nummer der Matrixzeile (= FA_i) und *j* für die Nummer der Matrixspalte (= DP_j) steht. Für Nichtdiagonalelemente gilt im Weiteren $i \neq j$.

28 Vgl. für diesen und den folgenden Satz Brown /Coupling/ 3

29 Statt den Symbolen X/O können die Matrixelemente auch mit konkreten Zahlenwerten belegt werden. Diese drücken dann die Stärke der Kopplungsbeziehung aus.

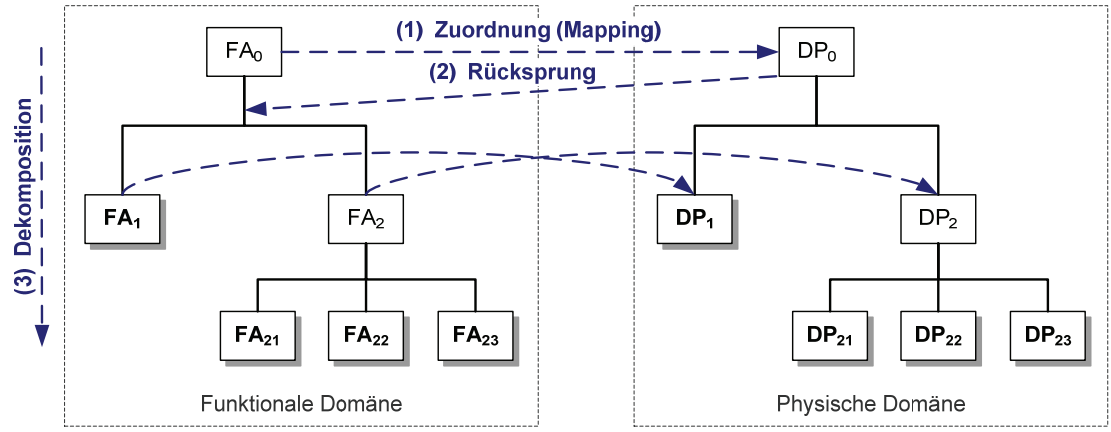


Bild 2-3: Zuordnungs- und Dekompositionsprozess[30]

Nachdem der Entwurf einer Dekompositionsebene abgeschlossen und durch die Entwurfsmatrix dokumentiert ist, muss ein Rücksprung in die funktionale Domäne vorgenommen werden.[31] Dort werden die FAs der im letzten Schritt betrachteten Hierarchieebene bei Bedarf weiter verfeinert. Anschließend erfolgt eine erneute Zuordnung geeigneter Designparameter. Dieses Hin- und Herspringen ist in Bild 2-3 dargestellt und führt zur Entstehung von Dekompositionshierarchien für FAs und DPs.

Die Dekomposition einer Anforderung wird solange fortgeführt, bis der korrespondierende Designparameter genügend Details für eine Implementierung beinhaltet.[32] Durch diese schrittweise Verfeinerung entstehen verschiedene Dekompositionszweige innerhalb der Dekompositionshierarchie. Die oberen Ebenen der FA-Hierarchie repräsentieren die groben Entwurfsziele. Die jeweils unterste Ebene eines Dekompositionsbaums wird durch die *Blattelemente* gebildet (in Bild 2-3 schattiert dargestellt). Die Gesamtheit der Blattelemente der DP-Hierarchie und ihre Beziehungen untereinander definieren den Grobentwurf der generierten Lösung.[33] Ist der Dekompositionsprozess beendet, kann die *vollständige Entwurfsmatrix* aus der FA- und DP-Hierarchie aufgestellt werden. Sie enthält alle Blattelemente der FA- und DP-Hierarchie und bildet die Dekompositionsstruktur des Entwurfs ab.

Nachdem die Analysephase durchlaufen wurde und der Grobentwurf in Form der vollständigen Entwurfsmatrix vorliegt, erfolgt die Detaillierung und Realisierung des

30 Quelle: Suh /Axiomatic Design/ 30
31 Vgl. für diesen Absatz Suh /Axiomatic Design/ 21 f.
32 Vgl. für diesen Absatz Suh /Axiomatic Design/ 29 und 31

Entwurfs (vgl. den rechten Teil des V-Modells in Bild 2-2). Dazu werden in sich abgeschlossene Entwurfselemente, so genannte Module, identifiziert. Ein *Modul* entspricht in Axiomatic Design einer Zeile der Entwurfsmatrix. Es erfüllt die zugehörige funktionale Anforderung, wenn es seine entsprechenden Designparameter als Input erhält.[34] Die Interdependenzen zwischen den Modulen werden durch ihre Schnittstellen zueinander näher beschrieben. Anhand der FA- und DP-Hierarchie der Entwurfsmatrix kann die entsprechende *Modulhierarchie* abgeleitet werden. Sie zeigt, aus Top-Down-Sicht betrachtet, wie die funktionalen Module schrittweise bis auf die Ebene der implementierbaren Blattmodule verfeinert werden.[35] Aus Bottom-Up-Sicht veranschaulicht die Modulhierarchie, wie durch die Aggregation der Blattmodule die Gesamtfunktionalität des entworfenen Systems bereitgestellt wird.

An die soeben geschilderten Entwurfsaktivitäten schließt sich die Umsetzung des Entwurfs an. Hier erfolgt die Abbildung zwischen physischer Domäne und Prozessdomäne. Dieser Aspekt wird in dem vorliegenden Bericht nicht betrachtet.

2.4 Entwurfsprinzipien von Axiomatic Design

2.4.1 Unabhängigkeitsaxiom (Independence Axiom)

Formen der Kopplung im Entwurf und ihre Folgen

Die funktionalen Anforderungen eines Entwurfs sind bei ihrer Formulierung per Definition von Suh unabhängig voneinander.[36] Das Unabhängigkeitsaxiom fordert, dass bei der Zuordnung der Designparameter zu den FAs diese Unabhängigkeit erhalten wird. Ist dies nicht möglich, entstehen Kopplungen im Entwurf. Sie resultieren aus Elementen in der Entwurfsmatrix, so genannten *Nichtdiagonal- oder Kopplungselementen*, die sich ober- oder unterhalb der Hauptdiagonale befinden. Diese Elemente koppeln die beteiligten funktionalen Anforderungen und heben somit ihre Unabhängigkeit voneinander auf. Deshalb sind sie während des Entwurfsprozesses nach Möglichkeit zu vermeiden bzw. bei der Überarbeitung des Entwurfs zu beseitigen. Im entworfenen System werden

33 Vgl. Melvin /Axiomatic System Design/ 43
34 Vgl. Suh /Axiomatic Design/ 200
35 Vgl. für diesen und den folgenden Satz Suh /Axiomatic Design/ 208-211
36 Vgl. für diesen Absatz Suh /Axiomatic Design/ 14 und 16 ff.

Nichtdiagonalelemente als Flüsse von Energie, Material oder Informationen[37] interpretiert. Je nach Form der Entwurfsmatrix können drei Arten der Kopplung unterschieden werden (vgl. Bild 2-4).

(a) Ungekoppelter Entwurf (Diagonalmatrix)

	DP_1	DP_2	DP_3
FA_1	X	O	O
FA_2	O	X	O
FA3	O	O	X

(b) Entkoppelter Entwurf (Dreiecksmatrix)

	DP_1	DP_2	DP_3
FA_1	X	O	O
FA_2	X	X	O
FA3	X	X	X

(c) Gekoppelter Entwurf (Vollmatrix)

	DP_1	DP_2	DP_3
FA_1	X	O	X
FA_2	X	X	O
FA3	X	X	X

Bild 2-4: Kopplungsformen von Entwürfen

Im Fall eines *ungekoppelten* Entwurfs sind nur die Hauptdiagonalelemente der Matrix besetzt.[38] Es existieren keine Nichtdiagonalelemente. Liegt ein *entkoppelter* Entwurf vor, so hat die Matrix eine Dreiecksform. Alle Kopplungselemente befinden sich entweder ober- oder unterhalb der Hauptdiagonale. Lässt sich die Matrix nicht in eine solche Dreiecksform überführen, so handelt es sich um einen *gekoppelten* Entwurf mit einer Vollmatrix. Laut Unabhängigkeitsaxiom sind nur ungekoppelte oder entkoppelte Entwürfe zulässig.

In *ungekoppelten Entwürfen* besteht keine Abhängigkeit zwischen den funktionalen Anforderungen. Jede Anforderung wird nur durch ihren korrespondierenden Designparameter beeinflusst.[39] Dadurch können die DPs in beliebiger Reihenfolge realisiert werden. Jede Anforderung kann somit durch die geeignete Wahl des zugehörigen Designparameters optimal erfüllt werden.[40]

Für *entkoppelte Entwürfe* existiert mindestens eine Reihenfolge, in der die Designparameter realisiert werden müssen, um die funktionale Unabhängigkeit der beteiligten FAs zu erhalten.[41] Im Fall (b) aus Bild 2-4 lautet diese Reihenfolge beispielsweise $DP_1/FA_1 - DP_2/FA_2 - DP_3/FA_3$. Da DP_1 neben FA_1 auch FA_2 und FA_3

37 Vgl. Lee, Jeziorek /Off-Diagonal Term/ 3
38 Vgl. für diesen Absatz Suh /Axiomatic Design/ 19 ff.
39 Vgl. Suh /Principles/ 55
40 Vgl. Suh /Axiomatic Design/ 44
41 Vgl. für diesen Absatz Suh /Principles/ 56

erfüllt, wird dieser Parameter zuerst realisiert. Anschließend kann FA_2 durch DP_2 erfüllt werden, wobei der bereits erfolgte Einfluss von DP_1 auf FA_2 berücksichtigt werden muss. Allerdings wird die Ausprägung von FA_1 hierdurch nicht angetastet. Zuletzt wird FA_3 durch DP_3 erfüllt, ohne die Ausprägungen von FA_1 und FA_2 zu verändern. Wird diese Reihenfolge nicht berücksichtigt, so werden aufgrund der Kopplungsbeziehungen die Ausprägungen bereits erfüllter Anforderungen durch die Realisierung nachfolgender Designparameter wieder verändert. Deshalb werden Iterationen bei der Umsetzung des Entwurfs erforderlich, um die veränderten FA-Ausprägungen wieder den Zielvorgaben anzupassen.[42] Durch die Kopplungen im Entwurf besteht somit stets die Notwendigkeit, die Realisierungsreihenfolge der Designparameter zu ermitteln und zu befolgen.

Gekoppelte Entwürfe sind laut Unabhängigkeitsaxiom unzulässig und ungekoppelten oder entkoppelten Entwürfen unterlegen.[43] Es existiert keine Reihenfolge, wie bei entkoppelten Entwürfen, in der die DPs realisiert werden können und die FAs dabei unabhängig voneinander, d. h. ohne Iterationen, erfüllt werden. In Beispiel (c) aus Bild 2-4 können durch die Reihenfolge DP_1/FA_1 – DP_2/FA_2 zunächst die Anforderungen 1 und 2 unabhängig erfüllt werden. Wird allerdings DP_3 realisiert, so ergibt sich aufgrund des Kopplungselements A_{13} eine Änderung an FA_1. Dadurch wird eine Neueinstellung von FA_1 durch DP_1 erforderlich, welche wiederum Einfluss auf FA_2 und FA_3 hat. Durch diese wechselseitigen Kopplungen (Zyklen) zwischen den FAs wird die per Definition bestehende Unabhängigkeit zwischen ihnen aufgehoben. Sie sind nun über ihre Designparameter aneinander gekoppelt. Bei der Änderung eines Designparameters sind u. U. Anpassungen an allen anderen DPs nötig, um die betroffenen funktionalen Anforderungen dennoch zu erfüllen.[44] Der Entwurf kann somit nur durch Iterationen umgesetzt werden. Nicht in jedem Fall konvergiert eine solche Iteration zu einer zulässigen Lösung.[45] Iterationen bei der Entwurfsrealisierung sind generell mit einem größeren Zeitbedarf und Aufwand zum Finden von geeigneten DP-Ausprägungen zur Erfüllung der FAs verbunden.[46] Zusätzlich erhöhen sie die objektive und die subjektiv wahrgenommene

42 Vgl. Brown /Coupling/ 3
43 Vgl. Suh /Axiomatic Design/ 115
44 Vgl. für die folgenden Sätze Suh /Axiomatic Design/ 21 und Melvin /Axiomatic System Design/ 30 und 40
45 Vgl. Brown /Coupling/ 1
46 Vgl. Melvin, Suh /Rearrangement/ 1

Komplexität des Entwurfs.[47] Dessen Informationsgehalt ist i. d. R. größer als der eines ungekoppelten oder entkoppelten Entwurfs.[48]

Ermittlung der Kopplungsform

Welche Art der Kopplung in einem konkreten Entwurf vorliegt, kann insbesondere bei komplexen Matrizen nicht mehr auf den ersten Blick festgestellt werden. Um die Kopplungsform leichter zu erkennen, kann eine graphentheoretische Interpretation der Entwurfsmatrix herangezogen werden.[49] Die Elemente der Hauptdiagonale der Entwurfsmatrix stellen hierbei die Knoten des Graphen dar und werden mit der Nummer des entsprechenden FA/DP-Paares bezeichnet. Die Nichtdiagonalelemente werden durch gerichtete Kanten[50] zwischen den Knoten repräsentiert und weisen auf eine Kopplungsbeziehung hin. Ein Nichtdiagonalelement A_{ij} zeigt einen Einfluss von DP_j auf FA_i und damit eine Abhängigkeit des Paares FA_i/DP_i von FA_j/DP_j an. Diese Abhängigkeit wird durch eine Kante vom Knoten *i* zum Knoten *j* verkörpert.

	DP$_1$	**DP$_2$**	**DP$_3$**	**DP$_4$**
FA$_1$	X	O	O	O
FA$_2$	X	X	X	O
FA$_3$	O	X	X	O
FA$_4$	X	O	O	X

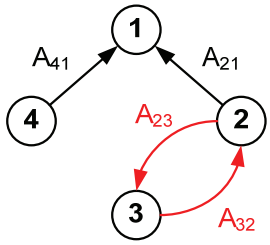


Bild 2-5: Direkter Entwurfszyklus

Zyklen in dieser Graphendarstellung weisen auf einen gekoppelten Entwurf hin.[51] Ein *Zyklus* ist eine geschlossene, gerichtete Kantenfolge, deren Start- und Endknoten identisch sind.[52] Diese Zyklen bedingen Iterationen bei der Realisierung des Entwurfs. In Bild 2-5 ist links eine gekoppelte Entwurfsmatrix und rechts die zugehörige Graphendarstellung gegeben. Die Kopplungselemente an den Positionen A_{23} und A_{32} erzeugen einen Zyklus zwischen den Knoten 2 und 3 (gelb hinterlegt).

47 Vgl. Lee, Jeziorek /Off-Diagonal Term/ 2
48 Vgl. Suh /Axiomatic Design/ 146
49 Vgl. für diesen Absatz Lee, Jeziorek /Off-Diagonal Term/ 3 f.
50 Vgl. Clark, Holton /Graphentheorie Grundlagen/ 252
51 Vgl. Lee, Jeziorek /Off-Diagonal Term/ 3 f.

Neben den *direkten Zyklen* als wechselseitige Kopplung zwischen genau zwei FA/DP-Paaren existieren auch *indirekte Zyklen*. Diese manifestieren sich als geschlossene, gerichtete Folge von Kopplungsbeziehungen über mehr als zwei FA/DP-Paare hinweg.[53]
Gibt es in der Graphendarstellung einer Entwurfsmatrix zwar Kanten aber keine Zyklen, so liegt ein entkoppelter Entwurf vor. Wenn hingegen keine Kanten zwischen den Knoten existieren, so gibt es auch keine Nichtdiagonalelemente und es handelt sich um einen ungekoppelten Entwurf.

Umgang mit Kopplung im Entwurf

Kopplungsbeziehungen zwischen den Entwurfsbestandteilen sind nach Möglichkeit zu vermeiden. Sie reduzieren die Erfolgswahrscheinlichkeit des Entwurfs und erhöhen somit dessen Komplexität.[54] Außerdem engen sie den Gestaltungsspielraum bei der Umsetzung des Entwurfs ein.[55] Jedoch lässt sich die funktionale Unabhängigkeit der FAs während der Dekomposition nicht immer vollständig erhalten und es entstehen Kopplungsbeziehungen zwischen den FA/DP-Paaren.
Für die Beurteilung solcher Abhängigkeiten wird in diesem Bericht zwischen Hin- und Rückkopplungen unterschieden. Diese Unterteilung basiert auf der Anordnung der FA/DP-Paare in der Entwurfsmatrix nach ihren Input-Output-Beziehungen. Eine Abhängigkeit vom betrachteten Modul zu einem in der Matrix vorgeordneten Modul kann als *Rückkopplung* verstanden werden. Das entsprechende Kopplungselement liegt unterhalb der Hauptdiagonale (z. B. Element A_{21} in Bild 2-5). Eine Abhängigkeit vom betrachteten Modul zu einem nachgeordneten Modul kann als *Hinkopplung* interpretiert werden. Sie äußert sich durch ein Kopplungselement oberhalb der Hauptdiagonale (z. B. Element A_{23} in Bild 2-5).

Kopplungen sind laut Unabhängigkeitsaxiom nur dann zulässig, wenn zumindest ein entkoppelter Entwurf in Form einer Dreiecksmatrix gewahrt wird. Existieren auch Kopplungselemente außerhalb der Dreiecksform (rot markiert in Bild 2-6 links), so kann

---

52 Vgl. Tittmann /Graphentheorie/ 15, 131

53 Zyklen können auch durch die so genannte Kopplungsvererbung entstehen. Sie resultieren aus azyklischen Kopplungen zwischen Blattelementen unterschiedlicher Dekompositionszweige, die sich auf die übergeordneten Kompositionsebenen vererben und dort Zyklen bilden (vgl. Brown /Coupling/ 3). Diese spezielle Form der Kopplungsentstehung wird in diesem Bericht nicht weiter betrachtet.

54 Vgl. Lee, Jeziorek /Off-Diagonal Term/ 1 und Suh /Axiomatic Design/ 146

55 Vgl. Suh /Axiomatic Design/ 126

durch eine Änderung der Reihenfolge der FA/DP-Paare dennoch eine Dreiecksmatrix erzeugt werden.[56] In Bild 2-6 wird dies durch das Vertauschen von FA_2/DP_2 und FA_3/DP_3 erreicht.[57] Eine solche Umordnung gelingt jedoch nur in entkoppelten Entwürfen ohne Zyklen.

	DP_1	**DP_2**	**DP_3**
FA_1	X	O	O
FA_2	X	X	X
FA_3	X	O	X

→

	DP_1	**DP_3**	**DP_2**
FA_1	X	O	O
FA_3	X	X	O
FA_2	X	X	X

Bild 2-6: Dreiecksmatrix durch Umordnung der FA/DP-Paare

Umgang mit Zyklen im Entwurf

Eine besondere Form der Kopplung ist durch zyklische Abhängigkeiten in der Entwurfsmatrix gegeben. Viele praktische Probleme enthalten iterative Abläufe in Form von Zyklen. Solche Zyklen führen stets zu unzulässigen, gekoppelten Entwürfen. Sie lassen sich nicht, wie oben geschildert, durch Umordnung der Entwurfselemente beseitigen. Es gibt dennoch drei Möglichkeiten, Zyklen aus einem Entwurf zu entfernen.

Erstens kann der vom Zyklus betroffene Entwurfsteil vollständig überarbeitet werden. Hierbei kann ein anderer Designparameter zur Erfüllung der betreffenden funktionalen Anforderung gewählt werden oder die Dekomposition dieser Hierarchieebene wird neu begonnen.[58] Ein in der Literatur oft zitiertes Beispiel für dieses Vorgehen ist die Überarbeitung eines zyklischen Entwurfs für einen Wasserhahn in Bild 2-7. Eine Änderung der physischen Designparameter (Hebel statt zwei Drehknaufe) führt dabei zu einer Beseitigung des Zyklus.

56 Vgl. Lee, Jeziorek /Off-Diagonal Term/ 1. Ein Algorithmus für diese Umformung ist in Suh /Principles/ 383, 385 dargestellt.

57 Ein Beispiel für die Umordnung der FA/DP-Paare findet sich in Suh /Axiomatic Design/ 33.

58 Vgl. Suh /Axiomatic Design/ 119

(a) Zwei-Knauf-Wasserhahn
(gekoppelter Entwurf)

→

(b) Hebelwasserhahn
(ungekoppelter Entwurf)

		DP$_1$: Heißwasserknauf	**DP$_2$:** Kaltwasserknauf
FA$_1$:	Steuere Wasser-temperatur	X	X
FA$_2$:	Steuere Wasser-fluss	X	X

→

		DP$_1$: Hebel horizontal bewegen	**DP$_2$:** Hebel vertikal bewegen
FA$_1$:	Steuere Wasser-temperatur	X	
FA$_2$:	Steuere Wasser-fluss		X

Bild 2-7: Entkopplung durch Überarbeitung des Entwurfs[59]

Ein zweiter Ansatz zur Beseitigung eines Zyklus besteht in der Erweiterung des Entwurfs. Hierzu müssen zusätzliche Entwurfselemente eingeführt werden, die die zyklischen Abhängigkeiten an einer Stelle aufbrechen.[60] In Frage kommen hierfür die Verwendung redundanter Elemente[61] oder die Anwendung entkoppelnder Entwurfsmuster. In Bild 2-8 wird der vorhandene Zyklus durch das Einfügen eines redundanten Entwurfselements aufgelöst. Dies ist gut in der Graphenrepräsentation der Entwürfe zu erkennen.

59 Quelle: o. V. /Technology/

60 Ein praktisches Beispiel ist die Verbesserung der Dampfmaschine in Suh /Axiomatic Design/ 24-27.

61 Redundante Entwurfsbestandteile widersprechen eigentlich der Forderung eines minimalen Entwurfs nach Korollar 2 in Suh /Principles/ 52 f. In diesem Fall besitzt jedoch die Forderung des Unabhängigkeitsaxioms eine höhere Bedeutung als die des Korollars (vgl. Suh /Axiomatic Design/ 12).

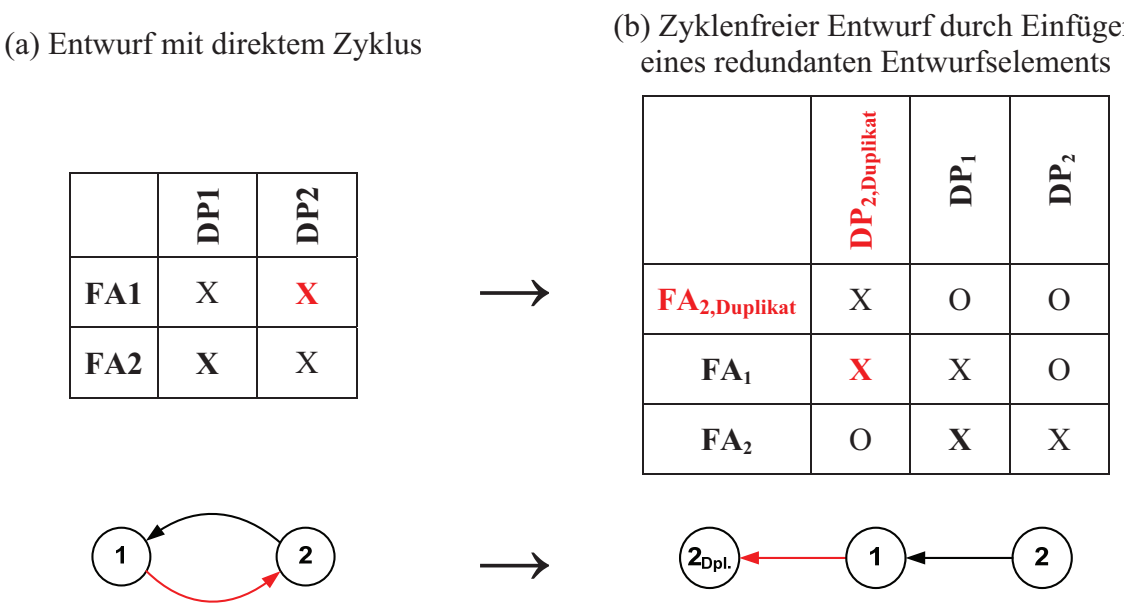


Bild 2-8: Auflösung eines Zyklus durch ein redundantes Entwurfselement

Eine dritte Möglichkeit der Zyklenbeseitigung bietet die Internalisierung des Zyklus in einem Entwurfsmodul. Dadurch wird der Zyklus vollständig in diesem Modul gekapselt und somit vor den restlichen Entwurfselementen verborgen. Diese Variante ist in Bild 2-9 veranschaulicht. Dort wird der Zyklus zwischen den Modulen M_2 und M_3 in einem neuen Modul $M_{2/3}$ gekapselt.

Diese soeben geschilderten Vorgehensweisen zur Beseitigung von Zyklen sind häufig mit einem hohen Änderungsaufwand verbunden. Bei einigen Entwurfsproblemen lassen sich Zyklen zudem nicht durch eine Überarbeitung oder Erweiterung des Entwurfs entfernen, weil sie entweder fachlich gewollt oder nicht vermeidbar sind. Die zyklische Kopplung manifestiert sich hierbei oft in Komponenten, die einen geschlossenen Informationsfluss erfordern und nicht ersetzt werden können.[62] Andere Iterationen ergeben sich aus der Tatsache, dass gewisse Probleme nicht bzw. nicht in endlicher Zeit mathematisch exakt lösbar sind. Diese Aufgaben können durch iterative, approximative Algorithmen mit festgelegten Abbruchkriterien bewältigt werden, oder sie werden an externe Systeme

[62] Beispielsweise geschlossene Signal-Feedback-Systeme wie in Melvin /Axiomatic System Design/ 61.

ausgelagert[63]. In solchen Ausnahmefällen werden in der Literatur auch gekoppelte Entwürfe akzeptiert.

(a) Entwurf mit direktem Zyklus

	DP_1	DP_2	DP_3	DP_4	Modul
FA_1	X				M_1
FA_2		X	**X**		M_2
FA_3		**X**	X		M_3
FA_4	X	X	X	X	M_4

(b) Kapselung des Zyklus in einem Entwurfsmodul

	DP_1	$DP_{2/3}$	DP_4	Modul
FA_1	X			M_1
$FA_{2/3}$		X		$M_{2/3}$
FA_4	X	X	X	M_4

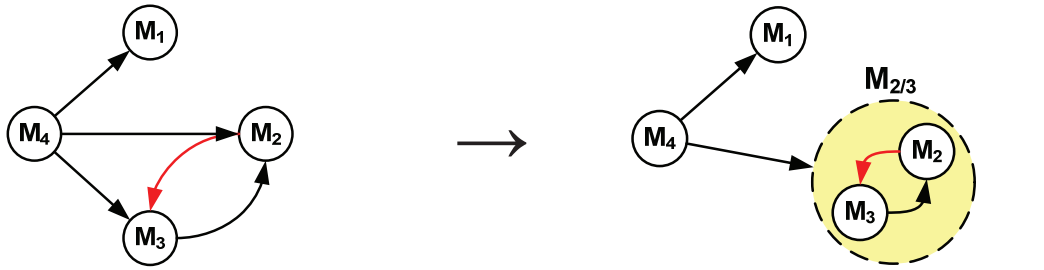


Bild 2-9: Internalisierung eines Zyklus

Um einen gekoppelten Entwurf dennoch zu verbessern, kann versucht werden, die Auswirkungen bestehender und nicht vermeidbarer Zyklen abzuschwächen. Hierfür können drei Empfehlungen gegeben werden. Zunächst sollten die gekoppelten Elemente des Zyklus möglichst nah an der Hauptdiagonale positioniert werden, um die Länge der Iteration zu reduzieren.[64] Dies kann, wie in Bild 2-10, durch eine Umordnung der Reihenfolge der FA/DP-Paare erreicht werden.

Zweitens sollten indirekte Zyklen nach Möglichkeit durch direkte Zyklen ersetzt werden. Direkte Zyklen sind in der Entwurfsmatrix leichter erkennbar und überschaubar. Schließlich sollte bei der späteren Umsetzung des Entwurfs versucht werden, die an einem Zyklus beteiligten Entwurfselemente in einem abgeschlossenen und autonomen Subsystem zu kapseln (vgl. Bild 2-9). Dadurch können der Zyklus und seine Auswirkungen vom

63 Ein Beispiel hierfür ist die Spritzgussfertigung von Kunststoffteilen in Suh /Axiomatic Design/ 257. Dort wird das zugrunde liegende Optimierungsproblem nicht iterativ gelöst, sondern an ein Expertensystem ausgelagert. Im Expertensystem kann eine zulässige Lösung durch die sequentielle Auswertung der Regeln der Wissensbasis gefunden werden.

restlichen Entwurf isoliert werden. Der Zyklus wird damit besser handhabbar und steuerbar.

(a) Entwurf mit direktem Zyklus

	DP_1	DP_2	DP_3	DP_4
FA_1	**X**			X
FA_2		**X**		
FA_3		X	**X**	
FA_4	X	X	X	**X**

(b) Zyklus nahe der Hauptdiagonale durch Umordnung der FA/DP-Paare

	DP_2	DP_3	DP_1	DP_4
FA_2	**X**			
FA_3	X	**X**		
FA_1			**X**	X
FA_4	X	X	X	**X**

Bild 2-10: Positionierung eines Zyklus nahe der Hauptdiagonale

2.4.2 Informationsaxiom (Information Axiom)

Information und Komplexität im Entwurf

Das in Abschnitt 2.4.1 vorgestellte Unabhängigkeitsaxiom fordert die Erhaltung der Unabhängigkeit der funktionalen Anforderungen. Es definiert und begrenzt dadurch die Menge zulässiger Entwürfe. Das Informationsaxiom bietet ergänzend hierzu ein quantitatives Maß für die Bewertung und den Vergleich unterschiedlicher zulässiger Entwürfe.[65] Dieses Maß ist der *Informationsgehalt*. Er erfasst die Menge zusätzlicher Informationen, die neben den Entwurfsgleichungen für die Realisierung des Designs benötigt werden.[66] Der Informationsgehalt I ist definiert durch die *Erfolgswahrscheinlichkeit* eines Entwurfs.[67] Die Erfolgswahrscheinlichkeit $P(FA_i)$ beschreibt die Wahrscheinlichkeit, mit der ein Entwurf in der Lage ist, die betrachtete funktionale Anforderung FA_i zu erfüllen. Das Informationsaxiom fordert, dass beim Vergleich mehrerer Entwurfsalternativen diejenige auszuwählen ist, welche die größte

64 Vgl. Melvin /Axiomatic System Design/ 50
65 Vgl. Suh /Axiomatic Design/ 39
66 Vgl. Suh /Axiomatic Design/ 114
67 Vgl. für diesen Absatz Suh /Axiomatic Design/ 39

Erfolgswahrscheinlichkeit und damit den geringsten Informationsgehalt aufweist. Dieser Zusammenhang ist durch die folgende logarithmische Gleichung definiert:[68]

$$I(FA_i) = \log_2 \frac{1}{P(FA_i)} = -\log_2 P(FA_i) \tag{2}$$

Der Informationsgehalt eines Gesamtsystems I_{System} lässt sich aus der Verbundwahrscheinlichkeit $P_{\{n\}}$ ermitteln, dass alle n funktionalen Anforderungen erfüllt werden:[69]

$$I_{System} = -\log_2 P_{\{n\}}, \text{ wobei } \quad P_{\{n\}} = P(FA_1 \textit{ erfüllt} \wedge \ldots \wedge FA_n \textit{ erfüllt}) \tag{3}$$

Informationsgehalt und Erfolgswahrscheinlichkeit stehen in enger Beziehung zur *Komplexität* eines Entwurfs.[70] Die Komplexität ist in Axiomatic Design ein Ausdruck der Unsicherheit, ob das Entwurfsziel erreicht werden kann. Sie verhält sich entgegengesetzt zur Erfolgswahrscheinlichkeit. Es ist deshalb ein Entwurf zu erarbeiten oder auszuwählen, der eine niedrige Komplexität und somit eine hohe Erfolgswahrscheinlichkeit bzw. einen geringen Informationsgehalt aufweist.

Berechnung des Informationsgehalts

Für den allgemeinen Fall lässt sich der Informationsgehalt eines Entwurfs laut Formel (2) aus dessen Erfolgswahrscheinlichkeit berechnen. Sind die Ausprägungen der funktionalen Anforderung stetig verteilt, so kann die Erfolgswahrscheinlichkeit wie in Bild 2-11 ermittelt werden.[71] Vom Designer wurde während der Formulierung der FAs für jede Anforderung FA_i eine *Zielspanne* $ZS_i=[zs_i^u, zs_i^o]$ festgelegt.[72] Diese Spanne begrenzt den Bereich der Ausprägungen der funktionalen Anforderung, die als zulässig erachtet werden.

Das entsprechend dem Entwurf umgesetzte reale System liefert für den korrespondierenden Designparameter DP_i konkrete Ausprägungen in der physischen Domäne.[73] Diese können am System gemessen oder vorher geschätzt werden. Für jede

---

68 Vgl. Suh /Axiomatic Design/ 39

69 Vgl. Suh /Axiomatic Design/ 39

70 Vgl. für diesen Absatz Suh /Axiomatic Design/ 472-474

71 Auch für den Fall diskreter Ausprägungen der FAs existiert eine Berechnungsvorschrift (vgl. Lee /Complexity/ 43), die jedoch in diesem Bericht nicht betrachtet werden soll.

72 Vgl. Suh /Axiomatic Design/ 41. Es gilt $zs_i^u < zs_i^o$.

73 Vgl. für diesen Absatz Lee /Lecture Note/ 4 f.

Ausprägung des DP_i lässt sich die Erfüllung der zugehörigen funktionalen Anforderung in Form der FA-Ausprägung ermitteln.

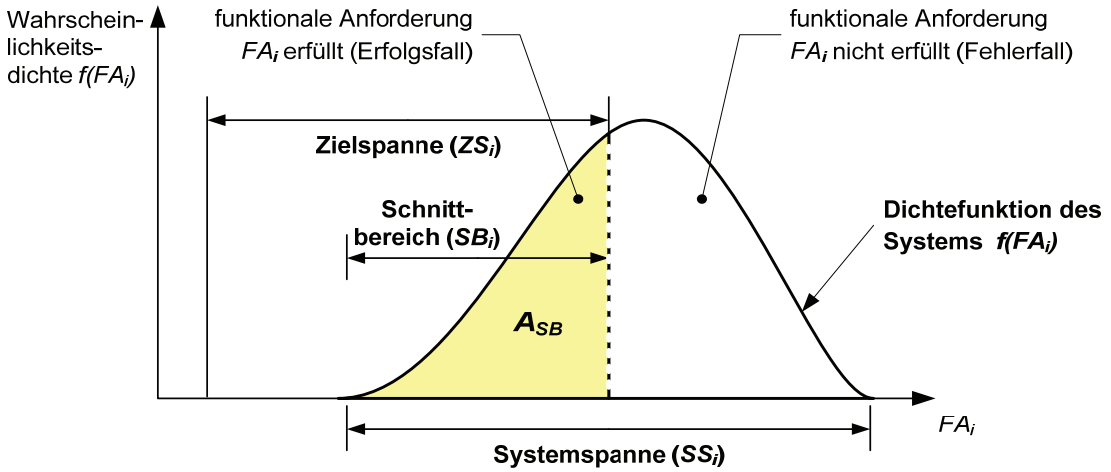


Bild 2-11: Grafische Ermittlung des Informationsgehalts[74]

Somit kann eine Dichtefunktion $f(FA_i)$, der vom System realisierten Ausprägungen von FA_i, aufgestellt werden. Diese Dichtefunktion ist über der *Systemspanne* $SS_i=[ss_i^u, ss_i^o]$ definiert.[75] Dieser Bereich repräsentiert somit die konkreten FA-Ausprägungen, die das realisierte System bereitstellt bzw. bereitstellen wird.

Der Schnittbereich SB_i zwischen vorgegebener Zielspanne und realisierter Systemspanne enthält alle Fälle, in denen das System die funktionale Anforderung erfüllt und somit die FA-Ausprägung innerhalb der Zielspanne liegt.[76] Die Fläche A_{SB} unterhalb der Dichtefunktion über diesem Schnittbereich entspricht deshalb der Erfolgswahrscheinlichkeit des betrachteten Entwurfs. Diese Schnittfläche lässt sich durch die Integration der Dichtefunktion über der Zielspanne berechnen. Die Ermittlung des Informationsgehalts nach Formel (2) kann somit folgendermaßen erweitert werden:

$$I(FA_i) = -\log_2 A_{SB}, \text{ wobei } \quad A_{SB} = P(zs_i^{\,u} \le FA_i \le zs_i^{\,o}) = \int_{zs_i^{\,u}}^{zs_i^{\,o}} f(FA_i)\, dFA_i \qquad (4)$$

74 Quelle: Suh /Axiomatic Design/ 41

75 Vgl. für diesen und den folgenden Satz Suh /Axiomatic Design/ 41. Es gilt $ss_i^u < ss_i^o$.

76 Vgl. für diesen Absatz und Formel (4) Suh /Axiomatic Design/ 41, 70 f. und 473

Reduktion des Informationsgehalts

Das Informationsaxiom fordert eine Minimierung des Informationsgehalts und damit die Verringerung der Menge an zusätzlichen Informationen, die zur korrekten Funktionsweise des entworfenen Systems erforderlich sind.[77] Um den Informationsgehalt eines Entwurfs zu reduzieren, muss dessen Erfolgswahrscheinlichkeit erhöht werden. Es muss also sichergestellt werden, dass das entworfene System stets die gestellten funktionalen Anforderungen erfüllt. Folglich muss die Systemspanne vollständig innerhalb der Zielspanne realisiert werden. Dies kann erreicht werden, indem die Abweichung von Ziel- und Systemspanne verringert wird. Hierzu ist es erforderlich, dass das Unabhängigkeitsaxiom erfüllt ist und ein zulässiger Entwurf vorliegt, da dann die Ausprägungen der funktionalen Anforderungen unabhängig voneinander beeinflusst werden können. Außerdem sollte die Varianz der realisierten FA-Ausprägungen und damit der Umfang der Systemspanne reduziert werden.

Durch die genannten Maßnahmen entsteht ein *robuster Entwurf*.[78] Dieser toleriert größere Schwankungen in den DP-Ausprägungen und kann trotzdem die Erfüllung der funktionalen Anforderungen sicherstellen. Ein robuster Entwurf zeichnet sich durch einen geringen Informationsgehalt und eine niedrige Komplexität aus.

---

77 Vgl. für diesen Absatz Suh /Axiomatic Design/ 45 f.

78 Vgl. für die folgenden Sätze Suh /Axiomatic Design/ 45 und 162

3 Serviceorientierte Architekturen

Nachdem in Kapitel 2 die Entwurfsmethodik Axiomatic Design vorgestellt wurde, soll in diesem Kapitel der Entwurfsgegenstand – die serviceorientierten Architekturen –näher betrachtet werden. Nach einer kurzen Einführung zu den Grundlagen serviceorientierter Architekturen werden ihre Bestandteile und deren Beziehungen zueinander erläutert. Anschließend werden eine Schichtenarchitektur für Services und wesentliche Architekturprinzipien dargestellt.

3.1 Begriffsbestimmung

In Wissenschaft und Praxis konnte sich bisher keine einheitliche und klar abgegrenzte Begriffsdefinition für serviceorientierte Architekturen durchsetzen. Ursächlich hierfür ist die Tatsache, dass das Konzept der SOA ein *Architekturmuster*[79] ist, ein „abstraktes Paradigma“[80] und keine konkrete Technologie. Deshalb sollen serviceorientierte Architekturen im Folgenden anhand ihrer Eigenschaften und Grundprinzipien näher betrachtet werden.

Als Architekturmuster beschreibt eine SOA eine Menge von Komponententypen (Dienstanbieter, Dienstnutzer, Dienstverzeichnis), deren Beziehungen und Interaktionsmuster.[81] Wesentliches Strukturierungsmerkmal einer SOA ist, dass die modellierten Funktionalitäten in Form von unabhängigen, lose gekoppelten Softwareservices organisiert sind.[82] Jeder Service kapselt eine abgegrenzte fachliche Funktion und verbirgt die Implementierungsdetails hinter seiner Schnittstelle.[83] Die Services interagieren mittels eines plattformunabhängigen Nachrichtenaustauschs, der von den technischen Details der Kommunikation abstrahiert. Eine serviceorientierte Architektur ist folglich implementierungsagnostisch[84] und konzentriert sich auf die Organisation fachlicher Aspekte und nicht auf deren technische Umsetzung.

---

79 Vgl. Bass, Clements, Kazman /Architecture/ 25 (Architectural Style)
80 Erl /Service-Oriented Architecture/ 2
81 Vgl. Bass, Clements, Kazman /Architecture/ 94
82 Vgl. Winter, Schelp /Dienstorientierte Architekturgestaltung/ 231
83 Vgl. für die folgenden Sätze Richter, Haller, Schrey /Serviceorientierte Architektur/ 413
84 Vgl. Erl /Service-Oriented Architecture/ 39

Eine SOA beschreibt als Softwarearchitektur „die grundlegende Organisation eines Systems, dargestellt durch dessen Komponenten, deren Beziehungen zueinander und zur Umgebung, sowie die Prinzipien, die den Entwurf ... des Systems bestimmen“[85]. Jedes dieser Organisationselemente – Komponenten, Beziehungen, Architekturprinzipien – wird in den folgenden Abschnitten einzeln vorgestellt.

3.2 Bestandteile einer SOA

Jede serviceorientierte Architektur setzt sich aus bestimmten Grundkomponenten zusammen: den Services, einer Serviceinfrastruktur (Service Infrastructure) sowie einem Serviceverzeichnis (Service Repository/Directory).

Kern jeder SOA sind ihre Services. Jeder Service kapselt eine fachlich abgegrenzte Funktionalität und kann in zwei Rollen agieren. Als *Service Provider* stellt er seine Dienste über eine Schnittstelle zur Verfügung und als *Service Consumer* fragt er die von anderen Providern angebotenen Dienste nach. Diese sehr allgemeine Sichtweise auf Services kann weiter verfeinert werden. Hierzu wird der Service in Form seiner Schnittstelle, seiner Implementierung und seiner Beschreibung betrachtet.
Der Zugriff auf die Funktionalität eines Service erfolgt über dessen Serviceoperationen.[86] Jede Operation stellt einen abgegrenzten Teilaspekt der Servicefunktionalität zur Verfügung. Gemeinsam formen die Operationen die *Serviceschnittstelle*. Sie spezifiziert die Signatur jeder Operation durch Operationsnamen, Eingangs- und Ausgangsparameter bzw. Eingangs- und Ausgangsnachrichten[87]. Dadurch stellt die Schnittstelle eine Art Vertrag zwischen dem Service dar, der sie implementiert (Provider) und dem, der sie nutzt (Consumer).[88] Die *Serviceimplementierung* ist die physische Umsetzung dieser Schnittstelle durch Softwareartefakte.[89]

---

85 IEEE /Architectural Description. IEEE 1471-2000/ zitiert nach Reussner, Hasselbring /Handbuch Software-Architektur/ 1

86 Vgl. für diesen Absatz Erl /Service-Oriented Architecture/ 134 f.

87 Je nach Betrachtungsperspektive wird eine Serviceschnittstelle durch Parameter oder Nachrichten definiert (vgl. für diese Anmerkung Feuerlicht /Service Granularity/ 317). Die implementierungsnahe Sichtweise beschreibt eine Schnittstelle durch die Signatur von Ein- und Ausgangsparametern. Die nachrichtenorientierte Sichtweise hingegen charakterisiert die Schnittstelle durch den Inhalt der kommunizierten Nachrichten. In diesem Bericht wird nicht gesondert zwischen beiden Perspektiven unterschieden. Im Folgenden wird meist von Schnittstellenparametern gesprochen, womit allgemein die ausgetauschten Daten gemeint sind.

88 Vgl. Vogel u. a. /Software-Architektur/ 60

89 Vgl. Krafzig, Banke, Slama /Enterprise SOA/ 60

Neben seiner Schnittstelle wird ein Service im Wesentlichen durch die *Servicebeschreibung* spezifiziert. Dies ist eine Sammlung von Metadaten, die grundlegende Eigenschaften des Service definiert. Die *funktionale Servicebeschreibung* erläutert zum einen die Funktionalität des Service in Form der Schnittstellenbeschreibung und zum anderen die Anforderungen für den Datenaustausch durch ein Informationsmodell.[90] Die *nichtfunktionale Servicebeschreibung* ergänzt diese Informationen um organisatorische und semantische Details. Als Teil der nichtfunktionalen Servicebeschreibung definiert die *Policy* die Eigenschaften eines Service (z. B. Transaktionsverhalten und Quality of Service).[91] Außerdem legt sie Anforderungen und Regeln zur Verwendung dieses Service fest (z. B. zulässige Aktionen und vorgeschriebene Interaktionsfolgen).[92]

Ein *Serviceverzeichnis* speichert die soeben beschriebenen Daten für alle eingetragenen Services.[93] Es ermöglicht dadurch die Suche und den Abruf aller notwendigen Informationen über die vorhandenen Dienstanbieter. Das Serviceverzeichnis ist ein optionaler Bestandteil einer SOA. Es ist jedoch insbesondere bei einer großen Anzahl von Services unverzichtbar und ermöglicht das dynamische Finden und Einbinden von Services zur Laufzeit. Die Grundlage für jegliche Kommunikation innerhalb der SOA bildet die *Serviceinfrastruktur*. Sie umfasst Mechanismen und Kommunikationskonzepte für die Realisierung der Serviceinteraktionen.[94]

3.3 Beziehungen innerhalb einer SOA

Bevor eine Interaktion zwischen zwei Services möglich ist, muss der nachfragende Service zunächst Informationen über mögliche Anbieter einholen.[95] Alle notwendigen Daten sind in den jeweiligen Servicebeschreibungen zusammengefasst. Existiert ein Serviceverzeichnis, so kann jeder Anbieter seine Beschreibung dort hinterlegen. Potenzielle Nachfrager können diese über Anfragen an das Verzeichnis abrufen und einen geeigneten Dienstanbieter auswählen (vgl. Bild 3-1). Die technischen Daten für den

90 Vgl. OASIS /Reference Model/ 21, 16
91 Vgl. Erl /Service-Oriented Architecture/ 242 f. und Krafzig, Banke, Slama /Enterprise SOA/ 61 f.
92 Vgl. OASIS /Reference Model/ 17 f.
93 Vgl. für diesen Absatz Krafzig, Banke, Slama /Enterprise SOA/ 60-62
94 Vgl. Krafzig, Banke, Slama /Enterprise SOA/ 64 f.
95 Vgl. für diesen Absatz Dostal u. a. /Web Services/ 14-16

Zugriff auf die Schnittstelle des Service Provider können von diesem selbst oder vom Serviceverzeichnis erfragt werden.

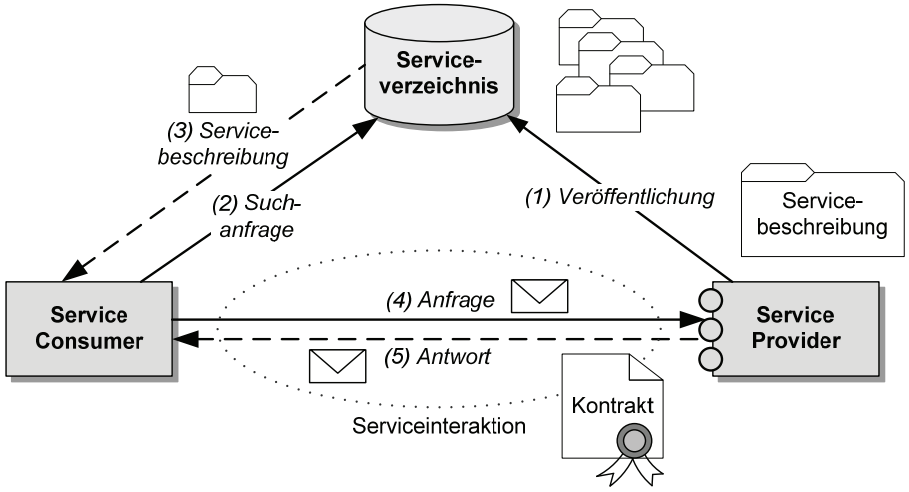


Bild 3-1: Beziehungen zwischen Bestandteilen einer SOA[96]

Die Kommunikation zwischen Dienstnachfrager und -anbieter ist durch die Servicebeschreibung vorgegeben und wird durch den Austausch von Nachrichten realisiert. Der nachfragende Service sendet eine Anfrage an den Dienstanbieter. Er benutzt hierbei die publizierte Serviceschnittstelle. Der Anbieter bearbeitet die Anfrage und sendet ggf. eine Antwortnachricht. Diese Abfolge von Nachrichten formt eine Serviceinteraktion.[97] Die Interaktion basiert auf einem Kontrakt. Dies ist eine explizit ausgehandelte oder implizite Vereinbarung zwischen Service Provider und Consumer über die Bedingungen und den Ablauf der Interaktion. Der Kontrakt basiert i. d. R. auf der Servicebeschreibung des Dienstanbieters, insbesondere den Daten der Service Policy.

3.4 Strukturierung serviceorientierter Architekturen

Eine Unternehmensarchitektur besitzt aus IT-Sicht u. a. die Ebenen der Geschäfts- und Anwendungslogik.[98] Die Geschäftslogik repräsentiert die Geschäftsprozesse des Unternehmens und definiert somit die fachlichen Anforderungen an die Informationstechnologie. Die Anwendungslogik setzt diese Anforderungen in separaten Anwendungssystemen um. Die Services einer SOA können als Abstraktionsschicht

96 Quelle: Dostal u. a. /Web Services/ 12
97 Vgl. für die folgenden Sätze OASIS /Reference Model/ 24 f.
98 Vgl. für diesen Absatz Erl /Service-Oriented Architecture/ 280 f.

zwischen diesen beiden Ebenen aufgefasst werden (vgl. Bild 3-2). Sie können sowohl Prozess- als auch Anwendungslogik kapseln. Diese Verknüpfung von Geschäftsprozessen und Anwendungssystemen durch eine Serviceebene ermöglicht deren unabhängige Anpassung und Entwicklung.[99]

Insbesondere bei serviceorientierten Architekturen von großem Umfang ist eine Strukturierung in Subsysteme oder Schichten erforderlich. Die oben eingeführte Serviceebene lässt sich demnach weiter verfeinern. Nach Inhalt und Eigenschaften der Services unterscheidet man in Bild 3-2 die Schichten der Anwendungsservices, der Geschäftsservices und der Orchestrierungsservices. Eine vollständige Darstellung dieser Ebenenarchitektur ist in Bild A-1 in Anhang A zu finden.

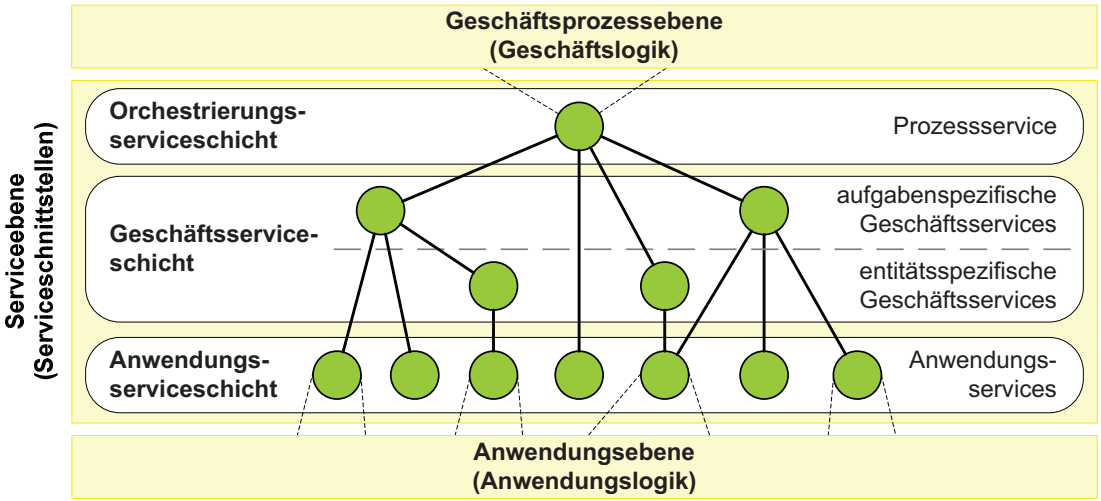


Bild 3-2: Einordnung und Unterteilung der Serviceebene[100]

Die *Anwendungsserviceschicht* kapselt die technologiespezifische Anwendungslogik der zugrunde liegenden Anwendungssysteme.[101] Sie besteht aus *Anwendungsservices*, welche die Datenverarbeitungsfunktionen konkreter Applikationen repräsentieren. Diese Services beziehen sich auf einen spezifischen Systemkontext. Sie sind meist generisch gestaltet und deshalb gut wieder verwendbar.

In der *Geschäftsserviceschicht* wird die fachliche Geschäftslogik in Form von *Geschäftsservices* dargestellt.[102] Diese kombinieren bei Bedarf untergeordnete

99 Vgl. Erl /Service-Oriented Architecture/ 51 und 334
100 Quelle: Erl /Service-Oriented Architecture/ 283 und 337
101 Vgl. für diesen Absatz Erl /Service-Oriented Architecture/ 335 und 337-339
102 Vgl. für diesen Absatz Erl /Service-Oriented Architecture/ 341 f.

Anwendungsservices, um eine Teilaufgabe zu realisieren. Je nach Inhalt unterscheidet man zwei Arten von Geschäftsservices. Ein *entitätsspezifischer Geschäftsservice* kapselt ein Geschäftsobjekt und bietet Operationen zur Datenmanipulation an. Er ist unabhängig von konkreten Geschäftsprozessen und deshalb gut wieder verwendbar. Ein *aufgabenspezifischer Geschäftsservice* repräsentiert hingegen die Geschäftslogik einer konkreten fachlichen Aufgabe oder eines Prozessschritts und ist seltener wieder verwendbar.

Die *Orchestrierungsserviceschicht* repräsentiert ganze Geschäftsprozesse oder Teilprozesse[103] in Form von *Prozessservices*.[104] Diese fungieren oft als Controllerservices[105] innerhalb der SOA. Sie komponieren und steuern Services der untergeordneten Schichten, die Kompositionsteilnehmer, zur Realisierung ihrer Prozessschritte. Die Prozessservices kapseln dabei Kompositionsdetails, wie z. B. Geschäftsregeln, Ausführungsreihenfolgen und Ausnahmen.[106] Dadurch kann das Prozesswissen in der Orchestrierungsebene zentral gesammelt werden. Außerdem werden die Anwendungs- und Geschäftsservices von diesen Informationen entlastet, wodurch sich ihre Wiederverwendbarkeit verbessert.

3.5 Architekturprinzipien serviceorientierter Architekturen

Obwohl sich bisher keine einheitliche Definition serviceorientierter Architekturen durchsetzen konnte, gibt es dennoch einen weitgehenden Konsens über die Kernkonzepte der Serviceorientierung. Für diesen Bericht sind insbesondere die lose Kopplung, hohe Autonomie, ausgewogene Granularität, Wiederverwendbarkeit und Komponierbarkeit der Services von Bedeutung. Anhand einiger dieser Architekturprinzipien soll in Kapitel 6 die Qualität der mit Axiomatic Design entworfenen SOA beurteilt werden. Eine Übersicht zu den Architekturzielen der Serviceorientierung und deren Beziehungen untereinander ist in Bild A-2 in Anhang A zu finden.

Die genannten Architekturprinzipien basieren auf einer Reihe von allgemeinen Entwurfskonzepten des Software Engineering. Diese bewährten Konzepte können

---

103 Zur Unterscheidung von Geschäftsprozess, Teilprozess, Prozessschritt und Arbeitsschritt vgl. Schmelzer, Sesselmann /Geschäftsprozessmanagement/ 109-111. Ein Prozessservice deckt i. d. R. einen Geschäfts- oder Teilprozess ab. Ein aufgabenspezifischer Geschäftsservice behandelt einen Prozessschritt. Entitätsspezifische Geschäftsservices und Anwendungsservices realisieren meist abgegrenzte Arbeitsschritte.

104 Vgl. für diesen Absatz Erl /Service-Oriented Architecture/ 335 und 344

105 Vgl. Erl /Service-Oriented Architecture/ 128

106 Vgl. Erl /Service-Oriented Architecture/ 403

selbstverständlich auch für den Entwurf einer SOA herangezogen werden. Grundlage jedes guten Entwurfs ist zunächst die Wahl einer geeigneten *Abstraktion*.[107] Durch das Abstrahieren konzentriert man sich auf die für den Entwurf wesentlichen Aspekte und blendet unwichtige Details aus. Eine Umsetzung dieses Prinzips besteht in der *Modularisierung*. Hierbei wird das zu entwerfende System in logisch zusammengehörige Bestandteile, die Module, zerlegt. In einer SOA erfolgt die Modularisierung nach der Idee des *Separation of Concerns*.[108] Das Gesamtproblem wird dabei in abgegrenzte, beherrschbare Elemente (Concerns) zerlegt, so dass jeder Service eine „klar umrissene fachliche Aufgabe“[109] kapselt. Ziele der Modularisierung und des Separation of Concerns sind ein hoher Zusammenhalt bzw. eine hohe Kohäsion innerhalb der Module und eine lose Kopplung zwischen den Modulen.[110]

Die *lose Kopplung* zwischen den Services einer SOA basiert maßgeblich auf der Kapselung der Funktionalität hinter einer Serviceschnittstelle, dem so genannten *Information Hiding*. Hierdurch können die Servicenutzer auf die angebotenen Dienste zugreifen, ohne von den Implementierungsdetails wissen zu müssen.[111] Durch die *Trennung von Schnittstelle und Implementierung* können die zugrunde liegenden Softwarekomponenten ausgetauscht und geändert werden, solange sie die Schnittstellendefinition erfüllen. Eine lose Kopplung wird technisch durch asynchrone Kommunikationsmuster, einen dokumentenbasierten Nachrichtenaustausch und dynamisches Binden unterstützt.[112] Aufgrund dieser Entwurfsprinzipien und Mechanismen besitzen die Services genügend Informationen, um miteinander kommunizieren zu können und dennoch unabhängig voneinander zu bleiben.[113] Dadurch können Kopplung und gegenseitige Abhängigkeiten innerhalb der SOA minimiert und die gesamte IT-Architektur flexibel gestaltet werden.

---

107 Vgl. für diesen Absatz Riebisch /Architektur- und Komponentenentwicklung/ 72 f. und Erl /Service-Oriented Architecture/ 298 f.
108 Vgl. Erl /Service-Oriented Architecture/ 290
109 Richter, Haller, Schrey /Serviceorientierte Architektur/ 413
110 Vgl. Riebisch /Architektur- und Komponentenentwicklung/ 73
111 Vgl. Vogel u. a. /Software-Architektur/ 124-126
112 Vgl. Krafzig, Banke, Slama /Enterprise SOA/ 46 f.
113 Vgl. Erl /Service-Oriented Architecture/ 297

Das Architekturprinzip der *Autonomie* fordert, dass Services einer SOA einen abgegrenzten Funktionsbereich kapseln.[114] Bezüglich dieses Autonomiebereichs sollen die Services größtmögliche Handlungshoheit besitzen und unabhängig von externen Einflüssen sein. Ziel sind Services mit einer hohen *Kohäsion*. Die Kohäsion ist ein entscheidender Aspekt beim fachlichen Zuschnitt der Services bzw. ihrer Operationen. Sie führt zur Forderung nach einer *ausgewogenen Granularität* der Services. Die Servicegranularität beschreibt den Umfang und die Art der implementierten Funktionalität.[115] Je enger der Aufgabenbereich eines Service gefasst ist und je spezieller er auf diesen Bereich ausgerichtet ist, desto feiner ist seine Granularität.[116] Die Granularität eines Service ist ausgewogen und angemessen, wenn sie dem Abstraktionsniveau der zugrunde liegenden fachlichen Anforderung entspricht.

Lose Kopplung, hohe Autonomie und angemessene Granularität bilden die Grundlagen für eine gute *Wiederverwendbarkeit* von Services.[117] Sind die Operationen möglichst generisch entworfen, so kann der umgebende Service einfach und häufig wieder verwendet werden. Das Ziel aller Bemühungen im Entwurf ist eine gute *Komponierbarkeit* der Services. Sie ist erreicht, wenn Services auf einfache Weise als Teilnehmer von Kompositionen fungieren oder selbst andere Services komponieren können.[118] Die Komposition ist somit eine spezielle Form der Wiederverwendung.

Hinsichtlich der Art der Servicekomposition lassen sich die Orchestrierung und die Choreographie unterscheiden.[119] Beide Formen beschreiben die Kombination von Services zu Prozessabläufen. Eine *Orchestrierung* steuert einen Prozessablauf innerhalb einer organisatorischen Einheit. Die Prozesslogik kann deshalb zentral vorgehalten werden. Hingegen sind bei einer *Choreographie* die Services verschiedener Organisationen beteiligt. Hierbei existiert jedoch kein Teilnehmer, der die Interaktion allein steuert, so dass die interorganisatorische Zusammenarbeit abgestimmt oder ausgehandelt werden muss.

---

114 Vgl. für die folgenden Sätze Erl /Service-Oriented Architecture/ 303 f.

115 Vgl. für die folgenden Sätze Riebisch /Architektur- und Komponentenentwicklung/ 73 und Erl /Service-Oriented Architecture/ 299

116 Vgl. Fiege, Stelzer /Analyse/ 5

117 Vgl. für diesen Absatz Erl /Service-Oriented Architecture/ 292 f.

118 Vgl. für diesen und den folgenden Satz Erl /Service-Oriented Architecture/ 301 f.

119 Vgl. für diesen Absatz Erl /Service-Oriented Architecture/ 200-211 und Benatallah u. a. /Composition/ 52

4 Anpassung von Axiomatic Design für den Entwurf von SOA

In den vorangegangenen zwei Kapiteln wurden zum einen die Entwurfsmethodik Axiomatic Design und zum anderen der Entwurfsgegenstand der serviceorientierten Architekturen unabhängig voneinander beschrieben. In diesem Kapitel soll Axiomatic Design dahingehend angepasst und erweitert werden, dass es den Entwurf einer SOA unterstützen kann. Hierzu wird zunächst auf bereits existierende Vorgehensmodelle für die Entwicklung einer serviceorientierten Architektur eingegangen. Anschließend wird Axiomatic Design in die Phasen der SOA-Entwicklung eingeordnet und seine Bestandteile und Axiome werden geeignet übertragen.

4.1 Vorgehensmodelle für die Entwicklung einer SOA

Aufgrund der speziellen Eigenschaften und Anforderungen serviceorientierter Architekturen müssen bei ihrer Entwicklung einige Besonderheiten berücksichtigt werden. Deshalb sollten traditionelle Vorgehensmodelle für die Softwareentwicklung entsprechend angepasst oder um geeignete Schritte ergänzt werden.[120] Vorschläge hierfür bieten beispielsweise Krafzig, Banke und Slama mit ihrem iterativen Thin Thread-Modell[121] sowie Bieberstein u. a. mit einer Sammlung von Tätigkeiten bei der serviceorientierten Analyse und dem serviceorientierten Entwurf[122].

Die Entwicklung einer serviceorientierten Architektur wird durch das Vorgehensmodell von Erl[123] umfassend beschrieben. Der Autor gliedert den Entwicklungsprozess in verschiedene Phasen. Die Grundsteine für die SOA werden in den ersten beiden Phasen, der serviceorientierten Analyse und dem serviceorientierten Entwurf, gelegt. In diesen beiden Phasen müssen die Besonderheiten und Architekturziele serviceorientierter Architekturen explizit berücksichtigt werden. Es schließen sich die plattformspezifische Implementierung sowie Test und Verifikation der Services an.[124] Nachfolgend wird die SOA durch die Einrichtung der Infrastruktur und die Verteilung der Services zum Einsatz

120 Vgl. Erl /Service-Oriented Architecture/ 358
121 Vgl. Krafzig, Banke, Slama /Enterprise SOA/ 288 f.
122 Vgl. Bieberstein u. a. /SOA Compass/ 95-99
123 Vgl. Erl /Service-Oriented Architecture/ 355-611

gebracht. Während des Betriebs erfolgt die Administration der Services und der SOA-Infrastruktur.

Die genannten Schritte entsprechen weitestgehend den Phasen der Softwareentwicklung nach Balzert[125]. Auch Balzert unterteilt den Entwicklungsprozess in Planung, Definition, Entwurf, Implementierung, Abnahme und Einführung sowie Wartung und Pflege der Software. Eine Gegenüberstellung der Phasen der SOA- und Softwareentwicklung ist in Bild B-2 in Anhang B zu finden.

Während der Phase der *serviceorientierten Analyse* wird erarbeitet, wie die fachliche Logik des Entwurfsbereichs zu Services zugeschnitten wird und welche Serviceebenen erforderlich sind.[126] Hierzu müssen zunächst die Anforderungen an die SOA erfasst und analysiert werden. Anschließend können z. B. ausgehend von Prozessanalysen Kandidaten für Serviceoperationen ermittelt werden. Durch die Gruppierung dieser Operationen zu logischen Einheiten entstehen Servicekandidaten[127]. Bereits zu diesem Zeitpunkt lassen sich durch die Extraktion von Prozesslogik aus den Services mögliche Servicekompositionen identifizieren. Bei den genannten Analysetätigkeiten sind stets die Architekturprinzipien der Serviceorientierung, wie z. B. Wiederverwendbarkeit, lose Kopplung und Autonomie, zu berücksichtigen und in die Entwürfe einzuarbeiten.

Im sich anschließenden *serviceorientierten Entwurf* werden aus den Servicekandidaten der Analysephase konkrete Serviceentwürfe abgeleitet.[128] Zunächst sind jedoch grundlegende Architekturentscheidungen zur Strukturierung der SOA zu treffen, beispielsweise die Festlegung der Serviceebenen. Anschließend erfolgt der Entwurf der Services durch die Ausarbeitung ihrer Schnittstellen. Hierzu müssen die Serviceoperationen inklusive ihrer Eingangs- und Ausgangsnachrichten sowie die erforderlichen Nachrichteninhalte in Form von Datenstrukturen definiert werden.[129] Nachdem erste Services vollständig spezifiziert wurden, können sie in der Orchestrierungsebene durch Prozessservices komponiert

---

124 Vgl. für die folgenden Sätze Erl /Service-Oriented Architecture/ 360-362

125 Vgl. Balzert /Software-Entwicklung/ 51

126 Vgl. für diesen Absatz Erl /Service-Oriented Architecture/ 377

127 Während der serviceorientierten Analyse und des serviceorientierten Entwurfs werden zunächst nur Servicekandidaten ermittelt. Erst im Feinentwurf werden aus den Kandidaten die endgültigen Services abgeleitet. Für eine bessere Lesbarkeit wird in den folgenden Betrachtungen häufig nur von Services gesprochen, auch wenn es sich streng genommen erst um Servicekandidaten handelt.

128 Vgl. für diesen Absatz Erl /Service-Oriented Architecture/ 448-451

129 Vgl. Erl /Service-Oriented Architecture/ 498

werden. Auch bei den Entwurfsaktivitäten sind wiederum die Architekturprinzipien und Standards der Serviceorientierung zu berücksichtigen.

Die vorgestellten Phasen der SOA-Entwicklung können in unterschiedlicher Weise zu einem Vorgehensmodell organisiert werden. Beim *Top-Down-Vorgehen* wird zunächst eine umfassende, serviceorientierte Unternehmensanalyse durchgeführt. Diese legt den Grundstein für eine hohe Qualität des SOA-Entwurfs, erfordert aber einen enormen Aufwand an Zeit und Ressourcen.[130] Beim *Bottom-Up-Vorgehen* werden Services meist nach Bedarf und ausgehend von der Anwendungsebene erstellt. Dies liefert zwar schnell einsatzfähige Anwendungsservices, bewirkt aber oft eine unzureichende Berücksichtigung serviceorientierter Architekturprinzipien.[131] Eine *Mischform* beider Strategien vereint deren Vorteile.[132] Hierbei werden parallel zur Top-Down-Analyse bereits erste Services entworfen und entwickelt. Während des Fortschreitens der Analyse müssen die erstellten Services immer wieder auf ihre Konsistenz zu den neuen Analyseergebnissen geprüft und ggf. überarbeitet werden. Dieses Vorgehen ermöglicht sowohl eine strategische Ausrichtung der SOA als auch die zeitnahe Implementierung erster Services. Es führt jedoch zu einem sehr komplexen und aufwendigen Entwicklungsprozess.
Das in diesem Bericht dargestellte Konzept zur Anwendung von Axiomatic Design für den Entwurf serviceorientierter Architekturen unterstützt in seiner derzeitigen Form nur die Top-Down-Vorgehensweise.

4.2 Einordnung von Axiomatic Design in den SOA-Entwicklungsprozess

Trotz der umfassenden Berücksichtigung serviceorientierter Architekturprinzipien bietet das Vorgehensmodell von Erl relativ wenige konkrete Hilfsmittel zur Entscheidungsfindung während der einzelnen Analyse- und Entwurfsaktivitäten. An dieser Stelle lässt sich Axiomatic Design in den Entwicklungsprozess einbinden. Es unterstützt alle Aspekte des Architekturentwurfs: die Zerlegung eines Systems in seine Komponenten, die Identifizierung der Beziehungen zwischen den Komponenten und die Anwendung der Architekturprinzipien.[133] Die Zuordnung entsprechender Entwurfselemente aus Axiomatic

130 Vgl. Erl /Service-Oriented Architecture/ 363-366
131 Vgl. Erl /Service-Oriented Architecture/ 366-368
132 Vgl. für die folgenden Sätze Erl /Service-Oriented Architecture/ 370-373
133 Vgl. Reussner, Hasselbring /Handbuch Software-Architektur/ 1

Design zu den Architekturbestandteilen ist in Tabelle 4-1 dargestellt. Eine ausführliche Übersicht findet sich auch in Tabelle B-1 in Anhang B.

Architekturbestandteile	Entwurfsunterstützung durch Axiomatic Design	Entsprechungen in einer SOA
Gesamtarchitektur **└ Systemkomponenten** **└ Systemmodule**	**Gesamtentwurf** **└ Gruppierung von Modulen** **└ Entwurfsmodule (spezifiziert durch FAs und DPs)**	**SOA** **└ Services, Kompositionen** **└ Serviceoperationen**
Beziehungen zwischen Systemkomponenten	**Kompositions- und Kopplungsbeziehungen in der Entwurfsmatrix**	**Serviceinteraktionen und -abhängigkeiten**
Architekturprinzipien	**Unabhängigkeits- und Informationsaxiom**	**serviceorientierte Architekturprinzipien**

Tabelle 4-1: Unterstützung des Architekturentwurfs durch Axiomatic Design

Als Entwurfsmethodik kann AD insbesondere die Phasen der serviceorientierten Analyse und des serviceorientierten Entwurfs begleiten. Hierzu werden die Schritte aus dem linken Zweig des V-Modells von Axiomatic Design aus Bild 2-2 in die Analyse- und Entwurfsphase der SOA-Entwicklung eingebettet. Diese Schritte umfassen den Top-Down-Entwurf der Systemarchitektur. Die Aktivitäten des rechten Bereichs des V-Modells behandeln den Feinentwurf und die Implementierung des Entwurfsergebnisses. Sie können durch geeignete Methoden der Softwaretechnik realisiert werden.[134] In Bild B-2 in Anhang B ist das Unterstützungspotenzial von Axiomatic Design in den einzelnen Phasen des Software- und SOA-Entwurfs dargestellt und kommentiert. Damit Axiomatic Design die Analyse und den Entwurf einer SOA geeignet unterstützen kann, müssen die Entwurfskonstrukte und -schritte entsprechend angepasst werden. Außerdem ist eine Interpretation der Axiome im Sinne der Serviceorientierung vorzunehmen. Diese Aspekte werden in den folgenden Abschnitten behandelt.[135]

134 Vgl. Suh /Axiomatic Design/ 266

135 Diese Abschnitte basieren auf den Vorarbeiten in Fiege, Stelzer /Analyse/.

4.3 Übertragung der Entwurfselemente

Eine Softwarearchitektur besteht u. a. aus Softwarekomponenten. Dies sind wohl definierte, in sich abgeschlossene Bausteine mit klar voneinander abgegrenzten funktionalen Zuständigkeiten.[136] Jede Komponente besitzt eine genau spezifizierte Schnittstelle und explizit definierte Abhängigkeiten zu ihrer Umgebung.[137] Die Komponenten einer Softwarearchitektur werden in Axiomatic Design durch die Entwurfsmodule repräsentiert und sind durch die KAs, FAs, DPs und deren hierarchische Anordnung beschrieben.

Die Kundenanforderungen entsprechen im SOA-Entwurf den Wünschen der verschiedenen Interessengruppen der SOA. Die Ermittlung dieser Anforderungen wird durch Axiomatic Design nicht direkt unterstützt. Sie erfolgt in der Softwareentwicklung während der Planungsphase im Rahmen der Erarbeitung des Lastenhefts. Für diese Aufgabe können Methoden des Requirements Engineering eingesetzt werden, beispielsweise die Erstellung und Auswertung von Geschäftsprozessdiagrammen[138].

Die funktionalen Anforderungen werden aus den ermittelten Kundenanforderungen abgeleitet. Die FAs von Axiomatic Design formulieren im SOA-Entwurfsprozess die fachlichen Anforderungen an die zu entwerfende serviceorientierte Architektur.[139] Die Gesamtheit aller FAs führt zu einem Pflichtenheft bzw. zu einer funktionalen Spezifikation der SOA. Zur Ermittlung der Anforderungen können Funktionsbäume[140] oder Geschäftsprozessbeschreibungen[141] genutzt werden. Zu den funktionalen Anforderungen können auch Restriktionen formuliert werden. Sie entsprechen im Softwareumfeld den nichtfunktionalen Qualitätsanforderungen an die SOA (z. B. hinsichtlich Verfügbarkeit, Sicherheit usw.).

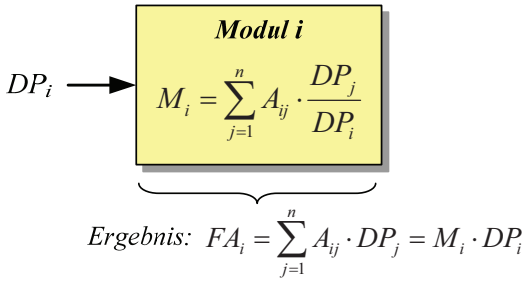
Bei den Designparametern der physischen Domäne ergibt sich die erste wesentliche Änderung gegenüber der ursprünglichen Definition in Axiomatic Design. Die DPs repräsentieren in AD eigentlich eine physische Lösungsmöglichkeit für eine funktionale Anforderung. Im Softwareentwurfsprozess hingegen werden die Designparameter

136 Vgl. Vogel u. a. /Software-Architektur/ 129
137 Vgl. Ludewig, Lichter /Software Engineering/ 379
138 Vgl. Balzert /Software-Entwicklung/ 64 f.
139 Vgl. Suh /Axiomatic Design/ 245
140 Vgl. Balzert /Software-Entwicklung/ 124 f.

allgemein als Input für die entworfenen Softwaremodule verstanden.[142] Im Rahmen der objektorientierten Softwareentwicklung repräsentiert ein DP beispielsweise die Daten eines Objekts.[143] Übertragen auf den SOA-Entwurf beschreibt ein Designparameter somit die Daten, die für die Erfüllung der korrespondierenden fachlichen Anforderung benötigt werden bzw. bei der Erfüllung der Anforderung entstehen.

(a) Ableitung der Module in Axiomatic Design

	DP_1	...	$\mathbf{DP_i}$	...	DP_n	
FA_1	X					M_1
...		X				…
$\mathbf{FA_i}$	$\mathbf{A_{i1}}$	**...**	$\mathbf{A_{ii}}$		$\mathbf{A_{in}}$	$\mathbf{M_i}$
...				X		…
FA_n					X	M_n



(b) Angepasste Modulinterpretation für den SOA-Entwurf

	$DP_{1,in} \mid DP_{1,out}$	⋮	$\mathbf{DP_{i,in} \mid DP_{i,out}}$	⋮	$DP_{n,in} \mid DP_{n,out}$	
FA_1	X					SO_1
...		X				…
$\mathbf{FA_i}$	$\mathbf{A_{i1}}$	**...**	$\mathbf{A_{ii}}$	**…**	$\mathbf{A_{in}}$	$\mathbf{SO_i}$
...				X		…
FA_n					X	SO_n

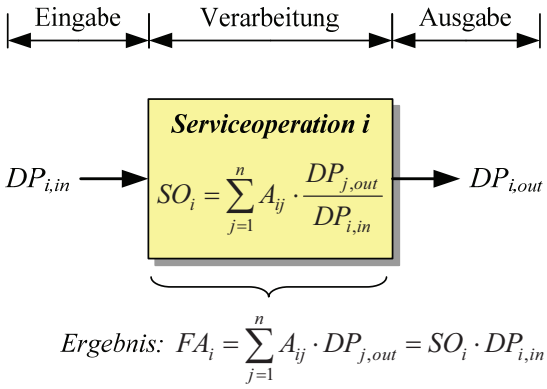


Bild 4-1: Ableitung und Interpretation der Entwurfsmodule[144]

141 Vgl. Balzert /Software-Entwicklung/ 126 ff. und Pimentel, Stadzisz /Use Case/ 3

142 Vgl. Suh /Axiomatic Design/ 245

143 Vgl. Suh /Axiomatic Design/ 267

144 Quelle: Abbildungsteil (a) in Anlehnung an Suh /Axiomatic Design/ 19 und 153. Im Gegensatz zur Quelle läuft der Summationsindex der Modulgleichung von $j=1..n$.

Die Module des Entwurfs ergeben sich in Axiomatic Design aus den Zeilen der Entwurfsmatrix. Sie repräsentieren das funktionale Verhalten, wenn sie die korrespondierenden Designparameter als Input erhalten (vgl. Bild 4-1 (a)).[145]

Im Rahmen des SOA-Entwurfs entspricht jedes Entwurfsmodul M_i einer Serviceoperation SO_i.[146] Die Serviceoperation SO_i erfüllt die zugrunde liegende fachliche Anforderung FA_i. Hierfür benötigt sie Eingangsdaten und erzeugt Ausgangsdaten, repräsentiert durch den korrespondierenden Designparameter DP_i.[147] Jede Serviceoperation kapselt außerdem alle Nichtdiagonalelemente A_{ij} ($j=1..n$) der entsprechenden Zeile i und damit die Abhängigkeiten zu anderen Serviceoperationen des Entwurfs (vgl. Bild 4-1 (b)).
Der Designparameter definiert die Schnittstelle der zugehörigen Serviceoperation. Er kann unterteilt werden in die Eingangsdaten (DP_{in}) und die Ausgangsdaten (DP_{out}). Die Eingangsdaten definieren die Input-Schnittstelle. Sie müssen der Operation vom Aufrufer als Eingangsparameter oder Inhalt der entsprechenden Eingangsnachricht bereitgestellt werden. Die Ausgangsdaten definieren die Output-Schnittstelle. Sie umfassen die vom Service bereitgestellten Daten, die als Ausgangsparameter oder als Inhalt der Ausgangsnachricht an den Aufrufer zurückgegeben werden (vgl. Bild 4-1 (b)).

4.4 Übertragung der Beziehungen

4.4.1 Übertragung der Modulhierarchie

Im Rahmen des Entwurfsprozesses mit Axiomatic Design werden den in der funktionalen Domäne formulierten FAs entsprechende Designparameter zugeordnet. Übertragen auf den SOA-Entwurf werden für jede fachliche Anforderung die erforderlichen Daten ermittelt und beschrieben. Die Funktionalität zur Erfüllung der Anforderung wird durch das zugrunde liegende Entwurfsmodul bzw. die entsprechende Serviceoperation repräsentiert. Im Dekompositionsprozess werden die FAs nach fachlichen, funktions- oder prozessbezogenen Kriterien untergliedert.

---

145 Vgl. Suh /Axiomatic Design/ 200 und 271

146 In den nachfolgenden Kapiteln werden bei der Ableitung der Servicehierarchie des SOA-Entwurfs die Module M_i aus der Entwurfsmatrix als Serviceoperationen SO_i interpretiert und bezeichnet.

147 EVA-Prinzip: Eingabe – Verarbeitung – Ausgabe

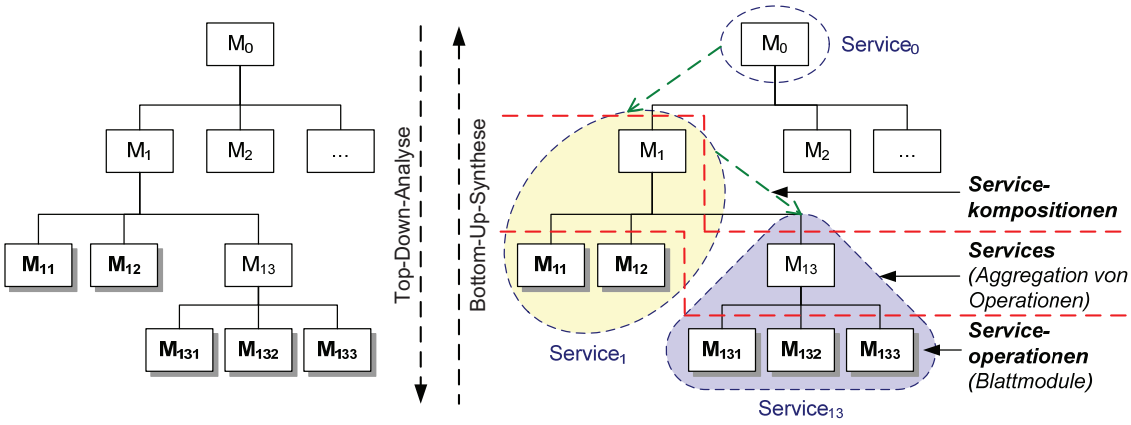


Bild 4-2: Ableitung der Servicehierarchie aus der Modulhierarchie

Parallel erfolgt die Verfeinerung der korrespondierenden DPs. Durch diesen Zuordnungs- und Dekompositionsprozess entstehen während der Top-Down-Analyse die vollständige Entwurfsmatrix (vgl. Abschnitt 2.3) und dadurch die Modulhierarchie (vgl. Bild 4-2 links).

Im SOA-Entwurf stellt die Modulhierarchie die *Servicehierarchie* dar. Die Servicehierarchie wird durch eine Bottom-Up-Synthese[148] aus der Modulhierarchie abgeleitet. Hierbei werden die einzelnen Entwurfselemente der Modulhierarchie schrittweise zu Bestandteilen der SOA aggregiert und komponiert (vgl. Bild 4-2 rechts). Die Blattmodule der Entwurfsmatrix repräsentieren, wie in Abschnitt 4.3 erläutert, die Serviceoperationen. Sie befinden sich auf der jeweils untersten Ebene des Dekompositionsbaums. In Bild 4-2 entspricht beispielsweise das Blattmodul M_{131} der Serviceoperation SO_{131}. Auf der nächsthöheren Ebene können die Serviceoperationen zu logischen Einheiten, den Servicekandidaten, gruppiert werden. Dies kann entsprechend den Dekompositionskriterien oder nach anderen fachlichen Gesichtspunkten erfolgen. In Bild 4-2 werden die Blattmodule bzw. Serviceoperationen SO_{131}, SO_{132} und SO_{133} zum $Service_{13}$ aggregiert. Auf allen übergeordneten Ebenen der Hierarchie werden die Servicekandidaten schließlich zu Servicekompositionen zusammengestellt.[149] Diese Ebenen repräsentieren folglich die Kompositionsstruktur des SOA-Entwurfs. Beispielsweise komponiert in Bild 4-2 der $Service_1$ die Operationen des untergeordneten $Service_{13}$.

---

148 Nicht zu verwechseln mit dem Bottom-Up-Vorgehen im Rahmen eines Vorgehensmodells.

149 Im Gegensatz zum Module-Junction-Diagramm des Axiomatic Design (vgl. Suh /Axiomatic Design/ 209-211) werden in der Modul- und Servicehierarchie neben den Blattmodulen auch die übergeordneten Modulebenen dargestellt. Da in einer SOA diese übergeordneten Ebenen (Services, Kompositionen) nicht nur durch bloße „Addition“ ihrer Blattmodule entstehen, sondern zusätzliche Logik beinhalten, müssen sie auch explizit dargestellt werden.

Für spätere mathematische Betrachtungen wird die soeben erläuterte Servicehierarchie in Form von Mengen beschrieben. Ein Service S_h des SOA-Entwurfs ist durch die Menge SO seiner Serviceoperationen SO_i definiert:

$$Service\ S_h := \ SO(S_h) = \{SO_i \ | \ SO_i \in S_h\} \tag{5}$$

Der gesamte SOA-Entwurf ergibt sich schließlich als Menge S der verschiedenen Services:

$$SOA\text{-}Entwurf := \ S = \{S_h \ | \ S_h \in SOA\} \tag{6}$$

4.4.2 Übertragung der Kopplungsbeziehungen

Die Beziehungen zwischen den Komponenten eines Entwurfs sind vollständig aus der Entwurfsmatrix ableitbar. Sie werden durch die Dekompositionsstruktur und die Nichtdiagonalelemente der Matrix dargestellt. Die Dekompositionsstruktur in Form der Modul- bzw. Servicehierarchie beschreibt in einer serviceorientierten Architektur die hierarchische Struktur und die Kompositionsbeziehungen der Services. Die Nichtdiagonalelemente repräsentieren zusätzliche Abhängigkeiten zwischen den beteiligten Serviceoperationen bzw. Servicekandidaten. Hierbei können verschiedene Arten von Abhängigkeiten unterschieden werden: die funktionale Abhängigkeit, die Datenabhängigkeit und die Steuerungsabhängigkeit.[150]

	DP_i	**...**	**DP_j**	
FA_i	X	...	**A_{ij}**	M_i bzw. SO_i
...	...	X	...	...
FA_j	**A_{ji}**	...	X	M_j bzw. SO_j

Bild 4-3: Abhängigkeiten zwischen Serviceoperationen

Eine *funktionale Abhängigkeit* liegt vor, wenn eine Serviceoperation die Funktionalität einer anderen Operation benötigt. In diesem Fall zeigt das Kopplungselement A_{ji} in Bild 4-3 eine Abhängigkeit zwischen Serviceoperation j (FA_j/DP_j) und Serviceoperation i (FA_i/DP_i) an. Dies bedeutet, dass SO_j zur Erfüllung ihrer zugrunde liegenden Anforderung FA_j die Dienste von Operation SO_i benötigt und diese veranlassen muss, ihre Funktion

[150] Vgl. Balzert /Software-Qualitätssicherung/ 523 f.

auszuführen.[151] Die abhängige Operation SO_j fungiert somit als Dienstnutzer (Service Consumer) und die unabhängige Operation SO_i als Dienstanbieter (Service Provider).

Eine *Datenabhängigkeit* besteht, wenn eine Operation Daten benötigt, die von einer anderen Operation bereitgestellt werden.[152] So zeigt das Kopplungselement A_{ji} beispielsweise an, dass die Serviceoperation *j* zur Erfüllung von FA_j Daten benötigt, die von Operation *i* zur Verfügung gestellt werden. Folglich entsteht ein Datenfluss von SO_i zu SO_j.[153] Die Ausgangsdaten der Serviceoperation *i* ($DP_{i,out}$) werden zu Eingangsdaten für die Operation *j*. SO_i ist somit die Datenquelle und SO_j die Datensenke. Eine solche Datenabhängigkeit ist eine spezielle Form der funktionalen Abhängigkeit. Die genutzte Funktionalität besteht darin, dass die unabhängige Serviceoperation die benötigten Daten bereitstellt.

Eine dritte mögliche Form der Abhängigkeit ist die *Steuerungsabhängigkeit*.[154] Das Kopplungselement A_{ji} zeigt in diesem Fall an, dass die Ausführung der Serviceoperation *j* von der Ausführung der Operation *i* abhängig ist.[155] Die steuernde Operation SO_i beeinflusst somit die Ausführung der abhängigen Operation SO_j. Ein solcher Kontrollfluss kann als Datenfluss interpretiert werden, indem der Steuerparameter als Datum an die abhängige Operation weitergegeben wird.
In den weiteren Ausführungen wird meist allgemein von einer Abhängigkeit zwischen Entwurfselementen gesprochen. Nur falls nötig, wird explizit zwischen den soeben geschilderten Formen unterschieden.

Die Abhängigkeiten zwischen den Serviceoperationen werden vollständig über deren Schnittstellen bzw. die entsprechenden Schnittstellendaten in Form der DPs abgebildet. Dies entspricht dem Architekturprinzip der losen Servicekopplung. Die unabhängige Operation fungiert jeweils als Dienstanbieter oder Datenquelle und die abhängige Operation als Dienstnutzer oder Datensenke. Eine Übertragung der Abhängigkeiten aus der Entwurfsmatrix auf die Servicehierarchie einer SOA ist in Bild B-1 in Anhang B

---

151 Dies entspricht der Aufrufsicht (Calls Structure) auf eine serviceorientierte Architektur (vgl. Bass, Clements, Kazman /Architecture/ 37).
152 Vgl. Page-Jones /Systemdesign/ 74-77 (Datenkopplung) und 77-80 (Datenstrukturkopplung)
153 Dies entspricht der Datenflusssicht (Data Flow) auf eine SOA (vgl. Bass, Clements, Kazman /Architecture/ 37).
154 Vgl. Page-Jones /Systemdesign/ 80-84
155 Dies entspricht weitestgehend der Kontrollflusssicht (Control Flow) auf eine SOA (vgl. Bass, Clements, Kazman /Architecture/ 37).

dargestellt.

In welcher Form eine Abhängigkeit später in der Implementierungsphase technisch aufgelöst wird, sollte während der Entwurfsphase bewusst offen gehalten werden.[156] Eine funktionale Abhängigkeit wird bei der Implementierung i. d. R. durch eine direkte Aufrufbeziehung zwischen den beteiligten Operationen realisiert. Eine Datenabhängigkeit kann durch eine direkte Interaktion von Datenquelle und -senke umgesetzt werden. Sie kann aber auch durch die indirekte Übermittlung der Daten mittels Datenspeicher oder durch die Sammlung und Bereitstellung der Daten durch einen übergeordneten Controllerservice abgewickelt werden.[157] Diese Abstraktion von der konkreten technischen Umsetzung entspricht der fachlichen, nicht-technischen Sichtweise während der Entwurfsphase.

4.5 Interpretation und Übertragung der Entwurfsprinzipien

4.5.1 Interpretation und Übertragung des Unabhängigkeitsaxioms

Interpretation der Kopplung im SOA-Entwurf

Insbesondere beim Entwurf von Softwarearchitekturen ist die hierarchische Modularisierung ein wichtiges Strukturierungsmittel.[158] Mit einer funktionalen Aufgabenteilung, wie sie im SOA-Entwurf mit Axiomatic Design verfolgt wird, ist stets auch ein gewisser Grad der Kopplung zwischen den entstehenden Modulen verbunden. So ist die Interaktion von Softwaremodulen, z. B. durch Funktionsaufrufe, Datenaustausch oder Kontrollflüsse, ein natürlicher und notwendiger Weg zur Realisierung des angestrebten Systemzwecks. Allerdings sollte die Modulkopplung stets auf ein vernünftiges Maß begrenzt werden, um negative Auswirkungen zu vermeiden.

Hierbei kann das Unabhängigkeitsaxiom als Gestaltungsrichtlinie herangezogen werden. Auf den Entwurf serviceorientierter Architekturen übertragen, fordert es eine größtmögliche und fachlich sinnvolle Unabhängigkeit der Serviceoperationen bzw.

---

156 Vgl. Schumann, Gerisch /Software-Entwurf/ 206

157 Die konkrete Ausgestaltung der Datenbereitstellung sollte sich nach der Spezifität der Daten richten. Allgemeine und oft benötigte Prozessdaten sollten tendenziell über einen zentralen Controllerservice verteilt werden (Push-Strategie). Hingegen sollten aufgabenspezifische Daten eher von den entsprechenden fachlich spezialisierten Services selbst beschafft werden (Pull-Strategie).

158 Vgl. Balzert /Software-Qualitätssicherung/ 569

Services. In serviceorientierten Architekturen manifestiert sich die Kopplung durch Interaktionen zwischen Services und dem dafür notwendigen Wissen über den Kommunikationspartner bzw. dessen Schnittstelle. Zumindest der Initiator eines Nachrichtenaustauschs bzw. Operationsaufrufs benötigt Informationen in Form der Servicebeschreibung und ist dadurch an die Schnittstellendefinition des Dienstanbieters gebunden. Einen wesentlichen Beitrag zur Entkopplung der Services bildet die Kommunikation über wohl definierte Schnittstellen. Dennoch sind bei der Erarbeitung eines Entwurfs Abhängigkeiten nicht vollständig zu vermeiden. In den nachfolgenden Abschnitten wird deshalb erläutert, wie bestehende Abhängigkeiten beseitigt oder abgeschwächt werden können.

Umgang mit Kopplung im SOA-Entwurf

Das Unabhängigkeitsaxiom fordert eine Minimierung der Kopplungsbeziehungen zwischen den Entwurfsbestandteilen.[159] Es ist stets ein ungekoppelter oder entkoppelter Entwurf anzustreben. Hierbei gelten die aus Axiomatic Design bekannten Empfehlungen des Abschnitts 2.4.1 entsprechend. Um die Entwurfsmatrix in eine Dreiecksform zu transformieren, besteht prinzipiell die Möglichkeit der Umordnung der FA/DP-Paare. Je nach Dekompositionskriterium sind jedoch gewisse Einschränkungen zu berücksichtigen. Wurde die Dekomposition des Entwurfs auf Basis von Prozessen oder sequentiellen Abläufen durchgeführt, ist eine Umordnung nur innerhalb des betrachteten Dekompositionszweigs sinnvoll möglich. Andernfalls würde die hierarchische Entwurfsstruktur und damit der sequentielle Zusammenhang der Teilprozesse und Prozessschritte zerstört.[160]

Rückkopplungen in der Entwurfsmatrix sind häufig fachlich nicht zu vermeiden. Sie führen zur so genannten *sequentiellen Kopplung*. Diese Kopplungsform besteht zwischen zeitlich aufeinander folgenden, voneinander abhängigen Teilaufgaben in Form von Input-Output-Beziehungen.[161] Dies ist typisch für prozessorientiert gestaltete Architekturen, wobei ein Prozessschritt meist von seinem Vorgänger und dessen Arbeitsergebnissen abhängig ist. Eine solche sequentielle Kopplung allein führt zunächst zu einem zulässigen

159 Vgl. Clapis, Hintersteiner /Software Development/ 274
160 Vgl. Melvin /Axiomatic System Design/ 51
161 Vgl. Brown /Coupling/ 4 f.

entkoppelten Entwurf (vgl. Abschnitt 2.4.1).
Rückkopplungen bei gleichzeitiger Existenz von Hinkopplungen bergen allerdings die Gefahr der Entstehung von Zyklen im Entwurf.[162] Hin- und Rückkopplung zwischen zwei Entwurfselementen führen zu einem direkten Zyklus. Eine gerichtete, geschlossene Folge von Hin- und Rückkopplungen kann einen indirekten Zyklus im Entwurf entstehen lassen. Zyklen führen stets zu einem gekoppelten Entwurf und sind deshalb zu vermeiden.

Umgang mit Zyklen im SOA-Entwurf

Eine spezielle Form der Kopplung sind Zyklen zwischen Bestandteilen einer serviceorientierten Architektur. Diese Zyklen entstehen durch gegenseitige Benutzungsbeziehungen[163] oder Abhängigkeiten zwischen den Serviceoperationen. Zyklen in einem physischen Entwurf äußern sich als Iterationen während der Einstellung der technischen Designparameter. Im Gegensatz dazu bewirken Zyklen in einem Softwareentwurf iterative Abläufe bzw. Schleifen bei der Ausführung der Software.

Auch im Softwarebereich führen zyklische Kopplungen zu negativen Effekten.[164] Sie reduzieren die Wiederverwendbarkeit der Bestandteile eines Zyklus, da diese aufgrund ihrer engen Kopplung nur schwer getrennt voneinander genutzt werden können. Außerdem führt die höhere Komplexität zu einer schlechteren Überschaubarkeit des Entwurfs. Um die Funktion eines Elements zu erfassen, müssen stets auch die anderen Elemente des Zyklus in die Betrachtung einbezogen werden. Zyklische Strukturen sind zudem wartungs- und änderungsunfreundlich. Die Auswirkungen von Änderungen an einem Zyklusbestandteil breiten sich innerhalb des Zyklus schnell aus und lassen sich nur unzureichend vorhersagen. In einer Entwurfsmatrix können zudem keine Angaben über den Ausgangspunkt einer Iteration, mögliche Abbruchbedingungen oder die Anzahl von Wiederholungen gemacht werden. Durch die geschilderten Effekte erhöhen Zyklen insgesamt die Komplexität des Entwurfs und verschlechtern dessen Verständlichkeit.

Zyklische Kopplungsstrukturen erzeugen stets Iterationen bei der Realisierung des Entwurfs. Sie sind deshalb nach Möglichkeit bereits während der Dekomposition zu

---

162 Vgl. Melvin, Suh /Rearrangement/ 1
163 Vgl. Roock, Lippert /Refactorings/ 40
164 Vgl. für diesen Absatz Roock, Lippert /Refactorings/ 40 f. und Vogel u. a. /Software-Architektur/ 117

vermeiden.[165] Für den Umgang mit Zyklen in einem SOA-Entwurf gelten grundsätzlich die allgemeinen Hinweise aus Abschnitt 2.4.1. Die dort genannten Vorschläge zur Beseitigung und Abschwächung von Zyklen können für den Entwurf serviceorientierter Architekturen geeignet übertragen und konkretisiert werden.

Um vorhandene Zyklen in einem SOA-Entwurf zu eliminieren, existieren drei Möglichkeiten. Erstens kann der zyklische Entwurfsteil vollständig neu entworfen werden. Hierzu ist eine neue Lösung für die funktionale Anforderung zu entwickeln, die nicht zu zyklischen Abhängigkeiten führt. Zweitens können Zyklen durch bestimmte Entwurfsmuster aufgebrochen werden. Die gekoppelten Serviceoperationen können z. B. durch Datenpuffer, die Duplikation von Funktionalität oder die Verwendung von Stellvertretermustern[166] entkoppelt werden. Eine dritte Möglichkeit ist die Internalisierung von Zyklen in Serviceoperationen. Sie kann beispielsweise durch die Kapselung von zyklischer Prozesslogik in separaten Komponenten[167] erreicht werden.

In einigen Situationen sind Zyklen jedoch fachlich notwendig und beabsichtigt.[168] Sie können folglich nicht beseitigt oder aufgebrochen werden. In diesen Fällen sollten die Auswirkungen der vorhandenen Zyklen abgeschwächt werden (vgl. Abschnitt 2.4.1). Hierzu sollten sie möglichst nah an der Hauptdiagonale liegen, damit die Länge der Iteration reduziert wird.[169] Außerdem sind direkte Zyklen vorteilhafter, da sie zu einer besseren Übersichtlichkeit in der Entwurfsmatrix führen.[170] Bei der Konkretisierung des Entwurfs sollte ein iterativer Ablauf nach Möglichkeit in einem abgeschlossenen, autonomen Subsystem, z. B. einem Service oder einer Servicekomposition, gekapselt werden.[171] Die Iteration kann dadurch von den restlichen Systemkomponenten isoliert werden und bleibt als Kopplung innerhalb des Subsystems handhabbar.[172]Auf höheren Architekturebenen sollten vor allem Kopplungen und Zyklen zwischen autonomen

165 Vgl. Humm, Voß, Hess /Regeln/ 401

166 Vgl. Wanner, Jäger /Wiederverwendung/ 31 f. Ein Beispiel zur Entkopplung eines Berechnungsprogramms und seiner Benutzerschnittstelle ist außerdem in o. V. /Technology/ gegeben.

167 Vgl. das Vermittlermuster in Gamma u. a. /Entwurfsmuster/ 385-397

168 Vgl. Roock, Lippert /Refactorings/ 41. Beispiele sind das iterative Lösen komplexer Probleme durch Näherungsalgorithmen (z. B. in der Fertigungsplanung) oder Abstimmungs- und Verhandlungsprozesse.

169 Vgl. Melvin /Axiomatic System Design/ 50

170 Vgl. Roock, Lippert /Refactorings/ 41

171 Vgl. Melvin /Axiomatic System Design/ 50

172 Ein Beispiel ist die Kapselung einer Iteration in einer Echtzeit-Softwaresteuerung in Melvin /Axiomatic System Design/ 180 f. und 188-190.

Subsystemen vermieden werden, denn ihre Beseitigung kann nicht lokal isoliert erfolgen, sondern erfordert Änderungen an der Architektur.[173]

Zusammenfassend kann festgehalten werden, dass das Unabhängigkeitsaxiom eine Minimierung der Kopplung zwischen den Services bzw. Serviceoperationen fordert. In der Praxis wird es dennoch zu fachlich bedingten Zyklen im Entwurf kommen. Diese sollten weitestgehend beseitigt oder abgeschwächt werden, denn das Unabhängigkeitsaxiom fordert auch im Softwarebereich die Wahrung eines zulässigen, entkoppelten oder ungekoppelten Entwurfs.

4.5.2 Interpretation und Übertragung des Informationsaxioms[174]

Interpretation des Informationsgehalts im SOA-Entwurf

Aus dem soeben erörterten Unabhängigkeitsaxiom lassen sich für den Entwurf einer SOA konkrete Forderungen zur Reduzierung der Kopplung von Services oder Serviceoperationen ableiten. Das Informationsaxiom ergänzt diese Forderungen um ein Konzept zur Bewertung und zum Vergleich verschiedener Architekturentwürfe. Es fordert die Minimierung des Informationsgehalts. Somit ist auch im Softwarebereich der beste Entwurf jener, der die höchste Erfolgswahrscheinlichkeit bzw. eine geringe Komplexität aufweist (vgl. Abschnitt 2.4.2).[175]

Die Definition des Informationsgehalts kann für die Anwendung im Softwarebereich durch die nachfolgende Anpassung vereinfacht werden. Gerade für einen Softwareentwurf kann die Erfüllung der Anforderungen nicht wie im technisch-mechanischen Umfeld direkt gemessen werden.[176] Vielmehr werden Anforderungen meist entweder erfolgreich umgesetzt oder nicht. Folglich kann man von der konkreten Form der FA-Ausprägungen (stetig oder diskret) abstrahieren und unterscheidet nur noch zwischen der erfolgreichen und der nicht erfolgreichen Erfüllung der Anforderungen. Somit kann für die betrachtete Anforderung FA_i eine binäre Zufallsvariable z_i definiert werden.[177] Diese binäre Variable

---

173 Vgl. Melvin /Axiomatic System Design/ 45 und Roock, Lippert /Refactorings/ 62
174 Vgl. zu den folgenden Ausführungen Fiege, Stelzer /Modellierung/ 9 ff., 23 ff.
175 Vgl. Suh /Axiomatic Design/ 247
176 Vgl. für die folgenden Sätze Suh /Axiomatic Design/ 295
177 Vgl. für diesen und den folgenden Absatz sowie Formel (7) Lee /Complexity/ 48 f.

unterscheidet nur zwischen zwei Zuständen – Erfolg und Fehler – und ordnet diesen Zuständen die entsprechenden Wahrscheinlichkeiten zu:

$$z_i = \begin{cases} 1 & \text{(Erfolg)} \quad \text{mit } P(FA_i) \\ 0 & \text{(Fehler)} \quad \text{mit } 1 - P(FA_i) \end{cases} \quad \text{und} \quad I(FA_i) = I(z_i = 1) \qquad (7)$$

Eine erfolgreiche Erfüllung der funktionalen Anforderung liegt vor, wenn die FA-Ausprägung innerhalb der Zielspanne liegt.[178] Die Zufallsvariable nimmt dann den Wert 1 an. Dieses Ereignis tritt mit der Erfolgswahrscheinlichkeit $P(FA_i)$ ein. Konnte die Anforderung hingegen nicht erfüllt werden, so liegt die FA-Ausprägung außerhalb der Zielspanne. In diesem Fall besitzt die Zufallsvariable den Wert 0. Die Wahrscheinlichkeit für dieses Ereignis beträgt 1-$P(FA_i)$.[179] Die Berechnung des Informationsgehalts erfordert nun nicht mehr die genaue Ermittlung der Ausprägungen der Variable FA_i. Zur Berechnung wird lediglich die Wahrscheinlichkeit $P(FA_i)$ betrachtet, dass die funktionale Anforderung erfüllt wurde. Dies entspricht dem Erfolgsereignis (z_i=1).

Berechnung des Informationsgehalts für einen SOA-Entwurf

Bevor die eigentliche Bewertung eines SOA-Entwurfs erfolgen kann, müssen in der Analysephase geeignete fachliche und qualitative Anforderungen an den Entwurf formuliert werden. Die Erfüllung dieser FAs bzw. die Einhaltung der Restriktionen sollte nach Möglichkeit gut ermittelbar sein. Für jede relevante Anforderung muss vor dem Entwurf eine Zielspanne für die korrespondierenden FA-Ausprägungen festgelegt werden. Diese Spanne sollte motivierend wirken, aber auch realistisch erreichbar sein. Hierfür kann man sich an den Ausprägungen der FAs in ähnlichen Projekten in der Vergangenheit orientieren. Das Entwurfsziel kann entweder getrennt für jede einzelne Komponente des Entwurfs oder für den Gesamtentwurf als Ganzes vorgegeben werden.

Nachdem die Entwurfsziele definiert sind, erfolgt die Erarbeitung der entsprechenden Lösung. Für diese Lösung muss anschließend das Ausmaß der Zielerreichung bestimmt werden. Hierzu wird die Systemspanne des Entwurfs für die betrachtete Anforderung ermittelt und der Zielspanne gegenübergestellt. Dies geschieht, indem die vom Entwurf realisierten oder realisierbaren FA-Ausprägungen konkret erfasst oder geschätzt werden.

---

178 Entspricht der linken, markierten Fläche A_{SB} in Bild 2-11

179 Entspricht der rechten Fläche $1\text{-}A_{SB}$ in Bild 2-11

Im Softwarebereich ist insbesondere dieser Schritt zur Bewertung eines Entwurfs mit Schwierigkeiten behaftet. Es gibt verschiedene Herangehensweisen, um die Erfolgswahrscheinlichkeit eines Entwurfs zu bestimmen und damit dessen Informationsgehalt zu berechnen. Man kann zwischen der Messung am bereits umgesetzten Entwurf und der Schätzung auf Basis von Erfahrungswerten unterscheiden. Beide Varianten werden nachfolgend für stetige und diskrete bzw. binäre Ausprägungen der FAs vorgestellt.

Wurde der Softwareentwurf bereits implementiert, so kann die Erfüllung der funktionalen Anforderungen am konkreten Softwaresystem oder Prototypen beobachtet werden. Hierzu muss durch die wiederholte Ausführung der Software bei gleichzeitiger Messung oder Erfassung der Merkmalswerte eine Stichprobe von Ausprägungen der betrachteten Anforderung ermittelt werden.

Sind die FA-Ausprägungen stetig verteilt, so können anhand der gesammelten empirischen Stichprobendaten ein geeigneter Verteilungstyp und dessen Parameter bestimmt werden.[180] Der Informationsgehalt des Entwurfs kann dann durch die Integration der Dichtefunktion $f(FA_i)$ dieser Verteilung laut Formel (4) auf Seite 20 ermittelt werden.[181]

Stetige FA-Ausprägungen sind im Softwarebereich nur für einige Anforderungen, wie die Antwortzeit oder Verfügbarkeit, gegeben. In vielen Fällen liegen sowohl bei der Festlegung der Zielspanne als auch bei der Bestimmung der Systemspanne diskrete und insbesondere binäre FA-Ausprägungen vor.[182] Viele Anforderungen werden entweder erfüllt (Erfolgsfall) oder nicht erfüllt (Fehlerfall). Die gesammelten FA-Ausprägungen lassen sich einer dieser beiden Mengen zuordnen. Zur Berechnung des Informationsgehalts kann dann auf die oben geschilderte binäre Betrachtungsweise und Formel (7) zurückgegriffen werden. An die Stelle der Bestimmung einer Dichtefunktion tritt nun die direkte Ermittlung der Erfolgswahrscheinlichkeit des Entwurfs nach folgendem Muster:

$$P(FA_i) = \frac{Anzahl\ erfolgreicher\ Realisierungen\ von\ FA_i}{Anzahl\ aller\ Realisierungen\ von\ FA_i} \tag{8}$$

Durch Einsetzen dieser Erfolgswahrscheinlichkeit in Formel (2) auf Seite 19 erhält man den Informationsgehalt des Entwurfs.

---

180 Vgl. Precht, Kraft, Bachmaier /Statistik/ 173

181 Vgl. Suh /Axiomatic Design/ 295

In vielen Fällen soll die Erfolgswahrscheinlichkeit jedoch bereits während der Entwurfsphase und nicht erst nach teilweise oder vollständig abgeschlossener Implementierung ermittelt werden. Die FA-Ausprägungen können folglich nicht am konkreten System gemessen werden, sondern müssen geeignet geschätzt oder vorhergesagt werden. Eine Schätzung kann auf dem Erfahrungswissen von Experten oder Vergangenheitsdaten ähnlicher Projekte basieren. Eine Vorhersage ist durch Simulationsexperimente oder die Verwendung von Softwareentwurfsmetriken[183] möglich.

Besitzt die betrachtete Anforderung stetigen Charakter, so muss zunächst ein Verteilungstyp für die FA-Ausprägungen festgelegt werden. In der Praxis kann häufig eine Normalverteilung angenommen werden.[184] Treten nur positive Ausprägungen auf, so kann beispielsweise die logarithmische Normalverteilung zugrunde gelegt werden.[185] Sind die Ausprägungen der funktionalen Anforderung auf ein festes Intervall begrenzt, so kann die zweiseitig gestutzte Normalverteilung verwendet werden.[186]
In einem zweiten Schritt müssen die Parameter der gewählten Verteilung auf Basis einer Schätzung oder Vorhersage bestimmt werden. Im Fall der Normalverteilung betrifft dies den Mittelwert und die Varianz der FA-Ausprägungen. Die Berechnung des Informationsgehalts erfolgt schließlich, wie oben geschildert, durch die Integration der Dichtefunktion der ermittelten Verteilung über der Zielspanne.
Handelt es sich allerdings um eine Anforderung mit binären Ausprägungen, so kann die Erfolgswahrscheinlichkeit anhand von Formel (8) direkt geschätzt und anschließend in die allgemeine Berechnungsformel (2) des Informationsgehalts eingesetzt werden.

Reduktion des Informationsgehalts eines SOA-Entwurfs

Der Informationsgehalt eines Entwurfs ist von dessen Komplexität abhängig.[187] Er lässt sich deshalb durch eine Verringerung der Entwurfskomplexität reduzieren (vgl. Abschnitt 2.4.2). Suh schlägt hierfür u. a. den Einsatz effizienter Algorithmen vor.[188] Dies

182 Vgl. für diesen Absatz und Formel (8) Suh /Axiomatic Design/ 295 und Lee /Complexity/ 44
183 Vgl. Pimentel, Stadzisz /Use Case/ 6 f.
184 Vgl. Precht, Kraft, Bachmaier /Statistik/ 127 ff.
185 Vgl. Bosch /Statistik-Taschenbuch/ 275-277
186 Vgl. Bosch /Statistik-Taschenbuch/ 270-273. Mit der linksseitig gestutzten Normalverteilung (Bosch /Statistik-Taschenbuch/ 273 f.) können rechtsoffene Intervalle und mit der rechtsseitig gestutzten Normalverteilung (Bosch /Statistik-Taschenbuch/ 274) linksoffene Intervalle abgebildet werden.
187 Vgl. Suh /Axiomatic Design/ 472
188 Vgl. Suh /Axiomatic Design/ 295

soll die Fehlerwahrscheinlichkeit der Software reduzieren. Clapis und Hintersteiner beziehen das Informationsaxiom auf die Senkung der Wartungskosten.[189] Auch sie wollen diese Kostensenkung durch eine verminderte Softwarekomplexität erreichen.

Für den Bereich der serviceorientierten Architekturen sind eine Reihe von Maßnahmen zur Komplexitätsreduktion denkbar. Zunächst sollte, wie vom Unabhängigkeitsaxiom gefordert, ein zulässiger Entwurf angestrebt werden. Dazu sind vorhandene Zyklen nach Möglichkeit zu beseitigen oder zumindest abzuschwächen. Die Zahl der Kopplungen zwischen den Serviceoperationen ist auf das notwendige Maß zu beschränken. Zweitens ist eine möglichst lose Kopplung anzustreben. Die Stärke der Kopplung kann durch das Information Hiding reduziert werden. Dies bedeutet, dass die Schnittstellen der Serviceoperationen möglichst schlank gestaltet sein sollten, also nur die wirklich benötigten Datenparameter enthalten.[190] Drittens führt auch die Befolgung der in Abschnitt 3.5 vorgestellten Architekturprinzipien zu einer guten Entwurfsqualität und damit zu einer möglichst geringen Komplexität des Architekturentwurfs.

Schwierigkeiten bei der Berechnung des Informationsgehalts[191]

In den vorangegangenen Abschnitten wurden Möglichkeiten vorgestellt, den Informationsgehalt eines SOA-Entwurfs zu ermitteln und zu reduzieren. Dabei ist deutlich geworden, dass sich die Einbeziehung des Informationsaxioms in den Softwareentwurfsprozess schwieriger gestaltet, als dies beim Entwurf im technisch-mechanischen Bereich der Fall ist. Hierfür gibt es mehrere Gründe. Ein erstes, wesentliches Problem bei der Beurteilung von Softwareentwürfen ist die mangelnde Messbarkeit. Häufig liegen Kriterien in Form von allgemeinen Qualitätsanforderungen wie Wartbarkeit, Erweiterbarkeit und Wiederverwendbarkeit vor. Deren Erfüllung kann nur selten anhand quantitativ messbarer Merkmalswerte des Softwareentwurfs ermittelt werden. Unterstützung können lediglich Softwaremetriken zur Bewertung der Entwurfsqualität liefern, wie sie später in Abschnitt 6.1 entwickelt werden.
Ein zweiter Problembereich ist die Bestimmung des Informationsgehalts während der Entwurfsphase. Hier scheidet eine Messung am bereits implementierten Entwurf von

189 Vgl. Clapis, Hintersteiner /Software Development/ 274
190 Vgl. Clapis, Hintersteiner /Software Development/ 274
191 Vgl. zu den folgenden Ausführungen Fiege, Stelzer /Modellierung/ 12 ff., 23 ff.

vornherein aus. Es bleibt nur eine Vorhersage oder Schätzung der Erfolgswahrscheinlichkeit des Entwurfs, beispielsweise durch die Auswertung von Entwurfsmetrikwerten. Die Datenbasis für Vorhersagen bzw. Schätzungen in Form von Erfahrungs- oder Vergangenheitswerten ist meist sehr klein oder gar nicht vorhanden. Die Wahl eines geeigneten Verteilungstyps einer Dichtefunktion ist oft willkürlich und selten statistisch fundiert. Aufgrund dieser Schwächen besteht die Gefahr der bewussten oder unbewussten subjektiven Einflussnahme bei der Ermittlung des Informationsgehalts. Um eine solche Verfälschung zu vermeiden, sollte die Bestimmung der Systemspanne des Entwurfs bezüglich einer funktionalen Anforderung ohne Kenntnis der zugehörigen Zielspanne erfolgen und umgekehrt. Weiterhin sollten Ziel- und Systemspanne wenn möglich von verschiedenen Beteiligten unabhängig voneinander festgelegt werden. Die Grenzen beider Spannen ergeben sich dann als Mittelwerte dieser getrennt ermittelten Werte.

Ein drittes Problem besteht in der mathematisch-exakten Formulierung der Berechnungsformeln für die Erfolgswahrscheinlichkeit (vgl. Formel (4) und (8)) und den Informationsgehalt (vgl. Formel (2)). Diese Formeln täuschen eine Genauigkeit vor, die aufgrund der unzureichenden Datenlage in der Praxis selten erreicht werden kann. Ein Vergleich verschiedener SOA-Entwürfe, der allein auf deren Erfolgswahrscheinlichkeit basiert, ist deshalb wenig sinnvoll.

Das Konzept des Informationsgehalts kann, trotz aller genannten Schwierigkeiten, in Verbindung mit anderen Kriterien und Metriken aus dem Software Engineering genutzt werden, um einen vorliegenden SOA-Entwurf zu bewerten und ggf. mit Alternativen zu vergleichen. In Abschnitt 6.1 dieses Berichtes soll ein Vorschlag für ein Messkonzept entwickelt werden, um die Qualität eines SOA-Entwurfs mittels Komplexitätsmetriken zu beurteilen. Laut Suh ist der Informationsgehalt eines Entwurfs eng mit dessen Komplexität verknüpft.[192] Deshalb erscheint es sinnvoll, die im Messkonzept definierten Komplexitätsmetriken in die Berechnung des Informationsgehalts eines SOA-Entwurfs einzubeziehen. Die Verwendung von Softwarekomplexitätsmetriken zur Ermittlung des Informationsgehalts wurde von Pimentel und Stadzisz vorgeschlagen.[193] Es ist allerdings anzumerken, dass die mathematische Herleitung der von den Autoren beschriebenen

[192] Vgl. Suh /Axiomatic Design/ 472

Berechnungsformeln für den Informationsgehalt nicht durchgängig nachvollziehbar ist. Auch die von ihnen verwendete Argumentation ist vermutlich nicht vollständig konform zum Informationsaxiom. Dennoch bietet die Einbeziehung von Komplexitätsmetriken in die Berechnung des Informationsgehalts einen Ansatz zur Übertragung und quantitativen Anwendung des Informationsaxioms auf SOA-Entwürfe.

Zusammenfassend kann festgehalten werden, dass das Informationsaxiom auch für den SOA-Entwurf eine Minimierung des Informationsgehalts fordert. Dies lässt sich durch eine Erhöhung der Erfolgswahrscheinlichkeit bzw. eine Reduzierung der Komplexität des Entwurfs erreichen. Die konkrete Messung des Informationsgehalts kann durch die Verwendung von Komplexitätsmetriken unterstützt werden.

4.6 Manueller Feinentwurf

Mithilfe der in diesem Kapitel vorgestellten angepassten Axiomatic Design-Methodik lässt sich ein erster, vorläufiger Entwurf einer serviceorientierten Architektur erarbeiten. Dieser Grobentwurf ist zunächst als Vorschlag zu verstehen, die entstandenen Services besitzen einen Kandidatenstatus.[194] Axiomatic Design unterstützt die Entwicklung einer SOA hauptsächlich in den Phasen der Analyse und des Grobentwurfs (vgl. Abschnitt 4.2 und Bild B-2). Nach Beendigung des SOA-Entwurfs mit Axiomatic Design muss sich auf der Basis der generierten Ergebnisse eine manuelle Überarbeitung des Entwurfs anschließen. Hierzu bedarf es erfahrener SOA-Designer und -Entwickler, die den vorliegenden Entwurfsvorschlag verfeinern und bei Bedarf erweitern oder ändern.

Die entworfenen Servicekandidaten sollten entsprechend ihren Eigenschaften und Kernfunktionen den verschiedenen Serviceschichten einer SOA aus Abschnitt 3.4 zugeordnet werden. Beispielsweise können Servicekandidaten mit starkem Bezug zu Geschäftsobjektdaten der Schicht der entitätsspezifischen Geschäftsservices zugewiesen werden. Ebenso werden aufgabenspezifische Geschäftsservices, Prozessservices und systemnahe Anwendungsservices identifiziert. Durch diese Zuordnung wird die mit Axiomatic Design generierte Servicehierarchie weiter im Sinne der Ebenenarchitektur aus Bild 3-2 strukturiert.

---

193 Vgl. Pimentel, Stadzisz /Use Case/ 6 ff.
194 Vgl. Erl /Service-Oriented Architecture/ 398 f.

Für jeden erarbeiteten Servicekandidaten muss außerdem die angemessene Berücksichtigung der in Abschnitt 3.5 genannten SOA-Architekturprinzipien geprüft werden.[195] Kopplung und Autonomie bzw. Kohäsion können auf der Grundlage der in Abschnitt 6.1 zu entwickelnden Metriken untersucht werden. Je nach Ergebnis einer solchen metrikgestützten Analyse kann es sinnvoll sein, eng gekoppelte Servicekandidaten zusammenzufassen oder zu überarbeiten sowie Kandidaten mit schlechter Kohäsion geeignet aufzuteilen. Ebenso ist die gewählte Servicegranularität auf ihre Angemessenheit zu überprüfen.[196] Weiterhin ist die Wiederverwendbarkeit der Services auch außerhalb des betrachteten Prozesskontexts sicherzustellen.[197] Beispielsweise sollte prozessspezifische Ablauflogik externalisiert und ggf. durch einen gesonderten Prozessservice gekapselt werden. Außerdem ist es empfehlenswert, auch zukünftige Anforderungen an die Servicekandidaten zu berücksichtigen.[198] Aus diesen Anforderungen resultieren ggf. zusätzliche Serviceoperationen, die den Funktionsumfang eines Service vervollständigen und seine Wiederverwendbarkeit verbessern können.[199]

Im Zuge des Übergangs vom Grob- zum Feinentwurf sind die Schnittstellen der Serviceoperationen zu detaillieren. Hierbei sollte für die Definition der Schnittstellenparameter bzw. Nachrichteninhalte auf bereits vorhandene Datenmodelle der Entwurfsdomäne zurückgegriffen werden.[200] Bei allen Aktivitäten ist stets auch die Einhaltung allgemein üblicher und unternehmensspezifischer Entwurfs- und Modellierungsstandards zu prüfen.[201]

---

195 Vgl. Erl /Service-Oriented Architecture/ 407

196 Vgl. Erl /Service-Oriented Architecture/ 556 f.

197 Vgl. Erl /Service-Oriented Architecture/ 417 f.

198 Vgl. Erl /Service-Oriented Architecture/ 559 und Richter, Haller, Schrey /Serviceorientierte Architektur/ 414

199 Beispielsweise sollten entitätsspezifische Geschäftsservices zumindest über Operationen für das Anlegen, Ändern, Abfragen und Löschen der Daten des repräsentierten Geschäftsobjekts verfügen (vgl. Erl /Service-Oriented Architecture/ 515).

200 Vgl. Erl /Service-Oriented Architecture/ 559

201 Vgl. Erl /Service-Oriented Architecture/ 498

5 Fallbeispiel für den Entwurf einer SOA mit Axiomatic Design

Nachdem Axiomatic Design im vorangegangenen Kapitel speziell für den Entwurf serviceorientierter Architekturen angepasst wurde, soll die entwickelte Methodik nun an einem Beispiel angewendet werden. Hierzu wird zunächst der fachliche Hintergrund des Fallbeispiels erläutert. Anschließend werden alle Schritte der angepassten Entwurfsmethode einzeln an den Beispieldaten vorgeführt. Den Abschluss dieses Kapitels bildet die visuelle Darstellung des erarbeiteten SOA-Entwurfs.

5.1 Hintergrund des Fallbeispiels

Das Szenario für das folgende Fallbeispiel wurde einem Tutorial[202] der Firma Oracle entnommen und in Zusammenarbeit mit einem SOA-Architekten von Oracle, Herrn Bernd Trops, leicht abgeändert sowie erweitert. Die Grundlage für den Entwurf bildet ein Auftragsbearbeitungsprozess in einem Unternehmen. Ziel des Entwurfs ist es, die Abarbeitung der Teilprozesse der Auftragsbearbeitung durch die Services einer SOA zu unterstützen. Hierzu gehört, dass zunächst die eingehenden Auftragsdaten verarbeitet, mit vorhandenen Daten abgeglichen und ggf. vervollständigt werden. Anschließend erfolgt die Auftragsprüfung nach bestimmten Gültigkeits- und Geschäftsregeln. Im Fall eines positiven Prüfergebnisses wird der Auftrag in die Erfüllungsphase übergeleitet. Das betrachtete Unternehmen kann einen Auftrag entweder aus seinem eigenen Lagerbestand (Lager als interner Lieferant) oder durch Bestellung bei einem externen Lieferanten bedienen. Anhand der dabei anfallenden Kosten wird die jeweils günstigste Alternative ausgewählt. Für den Transport der Bestellung zum Auftraggeber wird ein Logistikdienstleister beauftragt. Nach Abschluss dieser Schritte wird der Kunde über den Stand der Bearbeitung informiert und die Auftragsdaten werden verbucht.

[202] Vgl. Oracle /Tutorial/

5.2 Entwurf der SOA mit Axiomatic Design

5.2.1 Ermittlung der Kundenanforderungen

Der erste Schritt im Rahmen des Entwurfs serviceorientierter Architekturen mit Axiomatic Design ist die Ermittlung der Kundenanforderungen an die SOA. Hierzu müssen die Prozessbeteiligten und die Verantwortlichen der betroffenen Fachabteilungen hinzugezogen und befragt werden. Grundlage der Anforderungsanalyse kann eine Beschreibung des Ist-Prozesses, wie in Anhang C.1, oder eines wünschenswerten Soll-Prozesses sein. Die Kundenanforderungen können beispielsweise aus den Zielformulierungen der Teilprozessbeschreibungen extrahiert werden (vgl. Tabellen in Anhang C.1). Das Ergebnis dieses Analyseschritts ist eine Sammlung grober, fachlich-abstrakter und am Prozessziel orientierter Kundenanforderungen wie in Tabelle 5-1.

KA0:	Unterstütze die Bearbeitung eines Kundenauftrags.	
	KA1:	Unterstütze die Verarbeitung und Ergänzung der Daten eines neu eingetroffenen Kundenauftrags.
	KA2:	Unterstütze die Prüfung der Auftragsdaten und die Genehmigung des Auftrags.
	KA3:	Unterstütze die Ermittlung und Beauftragung des günstigsten Lieferanten und eines geeigneten Logistikunternehmens.
	KA4:	Unterstütze den Abschluss der Auftragsbearbeitung.

Tabelle 5-1: Kundenanforderungen an die zu entwerfende SOA

5.2.2 Ableitung der funktionalen Anforderungen

Ausgehend von den in Tabelle 5-1 formulierten Kundenanforderungen wird zunächst die erste Ebene der funktionalen Anforderungen abgeleitet (vgl. Tabelle 5-2). Während des sich anschließenden Zuordnungs- und Dekompositionsprozesses wird diese erste Ebene der funktionalen Anforderungen schrittweise verfeinert. Dies ist für FA_2 in Tabelle 5-2 angedeutet. Als Grundlage für diese fachliche Detaillierung kann vor allem die Prozessdokumentation für das Fallbeispiel in Anhang C.1 genutzt werden. Die untergeordneten FAs können beispielsweise aus den Beschreibungen der Teilprozesse in Tabelle C-1, Tabelle C-2, Tabelle C-3 und Tabelle C-4 extrahiert werden. Eine weitere

Quelle ist die Gesamtdarstellung des Auftragsbearbeitungsprozesses durch die ereignisgesteuerte Prozesskette (EPK) in Bild C-1.

FA0:	Bearbeite Kundenauftrag		
	FA1:	Empfange Kundenauftrag	
	FA2:	Prüfe Kundenauftrag	
		FA21:	Prüfe Plausibilität
		FA22:	Prüfe Gültigkeit der Kredit-karte
		FA23:	Prüfe Genehmigungsfähigkeit
			…
	FA3:	Erfülle Kundenauftrag	
	FA4:	Schließe Kundenauftrag ab	

Tabelle 5-2: Funktionale Anforderungen an die zu entwerfende SOA

Zusätzlich zur Auswertung dieser Dokumente können in der Praxis die Prozessbeteiligten befragt oder bereits existierende Anwendungssysteme zur Unterstützung der Prozessschritte analysiert werden.

5.2.3 Zuordnungs- und Dekompositionsprozess

Nachdem die funktionalen Anforderungen der ersten Dekompositionsebene ermittelt wurden, können ihnen entsprechende Designparameter zugeordnet werden. Jeder Designparameter beschreibt die Daten, die zur Erfüllung der zugehörigen funktionalen Anforderung benötigt bzw. geändert oder erzeugt werden (vgl. Abschnitt 4.3). Auf den oberen Hierarchieebenen werden diese Daten noch recht allgemein formuliert. Auf der untersten Ebene werden jedoch die Blattmodule und damit die Serviceoperationen entworfen. Deshalb müssen die DPs dort detaillierter ausgearbeitet werden, da sie die Schnittstellen der betreffenden Operationen definieren. Die zur Erfüllung der funktionalen Anforderung benötigten Daten, die direkt an der Eingangsschnittstelle übergeben werden, bilden den Inputteil des Designparameters (DP_{in}). Von der Operation erzeugte oder geänderte Daten, die an der Ausgangsschnittstelle bereitgestellt werden, repräsentieren den Outputteil des Designparameters (DP_{out}).

Ist der Abbildungsprozess zwischen funktionaler und physischer Domäne abgeschlossen, müssen für die entstandene Ebene der FA/DP-Paare die Abhängigkeiten der funktionalen Anforderungen untereinander ermittelt und in der Entwurfsmatrix durch Nichtdiagonalelemente dokumentiert werden. Hierbei wird für jede funktionale Anforderung untersucht, zu welchen anderen Designparametern, außer dem eigenen, eine Abhängigkeit besteht und welcher Art diese Abhängigkeit ist (funktionale, Daten- oder Steuerungsabhängigkeit, vgl. Abschnitt 4.4.2). Ist diese Kopplungsanalyse beendet, so ist der Entwurf der betrachteten Ebene der FA/DP-Hierarchie abgeschlossen und es erfolgt der Rücksprung in die funktionale Domäne. Dort werden die funktionalen Anforderungen der letzten Dekompositionsebene ausgewählt, die in den folgenden Schritten weiter verfeinert werden müssen.
Nach diesem iterativen Schema der Zuordnung von DPs zu FAs, Ermittlung von Kopplungsbeziehungen, Rücksprung in die funktionale Domäne und dortiger Dekomposition der FAs vollzieht sich der weitere Entwurf (vgl. Abschnitt 2.3).

5.2.4 Aufstellen der vollständigen Entwurfsmatrix

Nach Abschluss des beschriebenen Zuordnungs- und Dekompositionsprozesses kann die vollständige Entwurfsmatrix aufgestellt werden. Sie ist in Bild C-2 in Anhang C.2 und auszugsweise in Bild 5-1 dargestellt.
Bei der Angabe der Designparameter sind Input- und Outputteil durch einen senkrechten Strich voneinander getrennt. Das Symbol „–“ weist darauf hin, dass diese Schnittstelle keine Datendefinition enthält. Das Symbol „+“ zeigt an, dass neben den Daten der Eingangsschnittstelle weitere Daten benötigt werden, die über Abhängigkeiten in Form von Nichtdiagonalelementen beschrieben sind. Die Art der Abhängigkeit wird durch die Indizes der Nichtdiagonalelemente verdeutlicht (vgl. Legende).
Beispielsweise wird die Anforderung FA_{1131} durch das Modul M_{1131} erfüllt. Die Schnittstelle der entsprechenden Serviceoperation SO_{1131} ist durch DP_{1131} beschrieben. An der Eingangsschnittstelle erhält die Operation die Kundennummer ($DP_{1131,in}$). Sie sucht anhand dieser Nummer nach einem vorhandenen Kundendatenstammsatz und gibt diesen an der Ausgangsschnittstelle an den Aufrufer zurück ($DP_{1131,out}$). Die nachfolgende Operation SO_{1132} ist über eine Steuerungsabhängigkeit mit der Operation SO_{1131} gekoppelt. Dies wird durch das Nichtdiagonalelement $A_{1132,1131}$ angezeigt. Nur wenn die Operation SO_{1131} keine Kundendaten findet oder diese unvollständig sind, wird die Operation SO_{1132} ausgeführt.

Legende:

a) DP-Notation
$DP_{<Nummer>}$: *<Eingangsdaten> | <Ausgangsdaten>*
+ : weitere Eingangsdaten laut Nichtdiagonalelementen (Abhängigkeiten)
— : keine Eingangs- bzw. Ausgangsdaten

b) Nichtdiagonalelemente
X_F : Funktionale Abhängigkeit/Kopplung
X_D : Datenabhängigkeit/-kopplung
X_S : Steuerungsabhängigkeit/-kopplung

				DP_0: Daten zur Bearbeitung des Kundenauftrags									
				DP_1: Daten des Kundenauftrags						DP_2: Daten zur Auftragsprüfung			
				DP_{11}: Kopfdaten		DP_{113}: Kunden-daten							
				DP_{111}: — | Auftragsnummer	DP_{112}: neuer Auftragsstatus | aktualisierter Auftragsstatus	DP_{1131}: Kundennummer | vorhandene Kundendaten	DP_{1132}: neue Kundendaten | Kundennummer	DP_{12}: — | Auftragspositionen	DP_{13}: + | —	DP_{21}: + | Auftragsplausibilität	⋮	DP_{24}: + | Prüfergebnis	**Blatt-module**
FA_0: Bearbeite Kundenauftrag	FA_1: Empfange Kundenauftrag	FA_{11}: Verarbeite Auftragskopf	FA_{111}: Erzeuge Auftragsnummer	**X**									M_{111}
			FA_{112}: Setze Auftragsstatus		**X**							X_D	M_{112}
		FA_{113}: Verar-beite Kunden-daten	FA_{1131}: Suche vorhandene Kundendaten			**X**							M_{1131}
			FA_{1132}: Speichere neue Kundendaten			X_S	**X**						M_{1132}
		FA_{12}: Verarbeite Auftragspositionen						**X**					M_{12}
		FA_{13}: Speichere Auftragsdaten		X_D	X_D	X_D		X_D	**X**				M_{13}
	FA_2: Prüfe Kundenauftrag	FA_{21}: Prüfe Plausibilität						X_D		**X**			M_{21}
		...									...		...
		FA_{24}: Entscheide weitere Auftragsbearbeitung								X_D		**X**	M_{24}

Bild 5-1: Ausschnitt der vollständigen Entwurfsmatrix des Fallbeispiels

Eine weitere Abhängigkeit besteht zwischen Operation SO_{21} und Operation SO_{12}. Es handelt sich um eine Datenabhängigkeit, bei der SO_{21} die Auftragspositionen ($DP_{12,out}$) benötigt, die durch SO_{12} bereitgestellt werden.

5.2.5 Überarbeitung des Entwurfs

Im ersten Entwurfsversuch entsteht u. a. eine Hinkopplung zwischen Operation SO_{112} und SO_{24} (in Bild 5-1 rot hervorgehoben). Sie ist darin begründet, dass durch die Entscheidung über die weitere Auftragsbearbeitung in FA_{24} ein neuer Auftragsstatus generiert wird. Da Operation SO_{112} den Auftragsstatus verwaltet, ist sie an alle Operationen gekoppelt, die diesen Status aktualisieren. Dadurch entsteht die Abhängigkeit zur Operation SO_{24} und außerdem zu den Operationen SO_{314}, SO_{322} und SO_{411}.

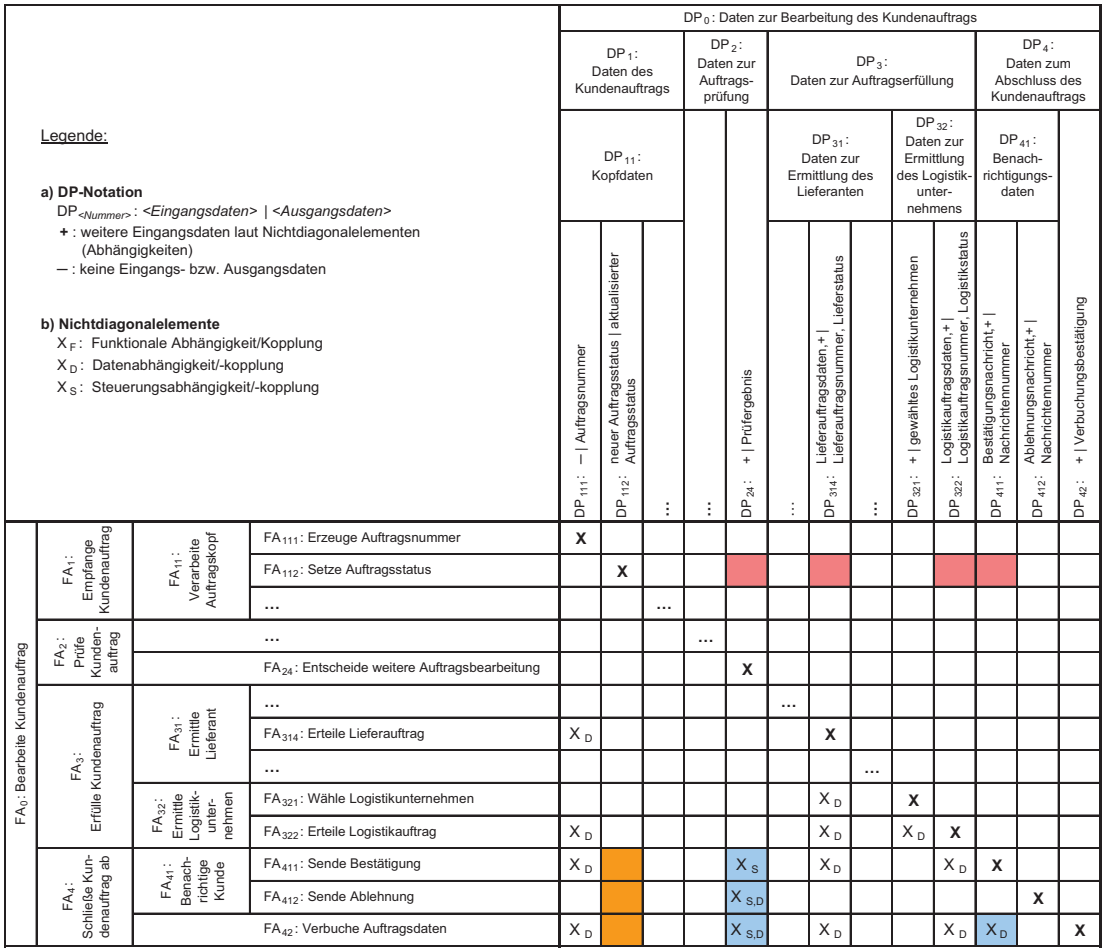


Bild 5-2: Eliminierung von Hinkopplungen

Diese Hinkopplungen werden durch Nichtdiagonalelemente oberhalb der Hauptdiagonale verdeutlicht. Sie bergen die Gefahr der Entstehung von Zyklen im Entwurf und werden deshalb im überarbeiteten Entwurf in Bild C-3 in Anhang C.2 eliminiert. Ein Ausschnitt der überarbeiteten vollständigen Entwurfsmatrix ist in Bild 5-2 dargestellt. Die rot hinterlegten Bereiche der Matrix zeigen die beseitigten Kopplungselemente. Alle Operationen, die den Auftragsstatus beeinflussen (*SO24, SO314, SO322, SO411*) führten im ersten Entwurf zu einer Aktualisierung des Status in *SO112*. Folglich konnte beim Abschluss des Kundenauftrags (*FA4*) dessen aktueller Status direkt über Operation *SO112* abgefragt werden. Durch die Überarbeitung des Entwurfs wurden die Nichtdiagonalelemente oberhalb der Hauptdiagonale beseitigt. Alle Operationen, die bisher den Auftragsstatus direkt aus Operation *SO112* bezogen haben, müssen diesen nun über die Operationen ermitteln, die diesen Status verändern (*SO24, SO314, SO322, SO411*). Die Abhängigkeiten in *FA4* zu *SO112* (orange markierte Bereiche) wurden demzufolge durch andere Nichtdiagonalelemente ersetzt, die in Bild 5-2 blau hinterlegt sind. Durch diese

Änderung konnten die Kopplungselemente oberhalb der Hauptdiagonale entfernt werden. Gleichzeitig ergibt sich jedoch der Nachteil, dass der aktuelle Auftragsstatus nicht mehr zentral bei der Operation SO_{112} abgefragt werden kann.

5.2.6 Ableitung der Servicehierarchie

Durch den iterativen Dekompositions- und Zuordnungsprozess aus Abschnitt 5.2.3 entstehen die FA- und DP-Hierarchie. Aus ihnen kann die Modulhierarchie abgeleitet werden (vgl. Bild 5-3, links). Sie zeigt das Ergebnis der Top-Down-Analyse der Module in ihre Bestandteile.

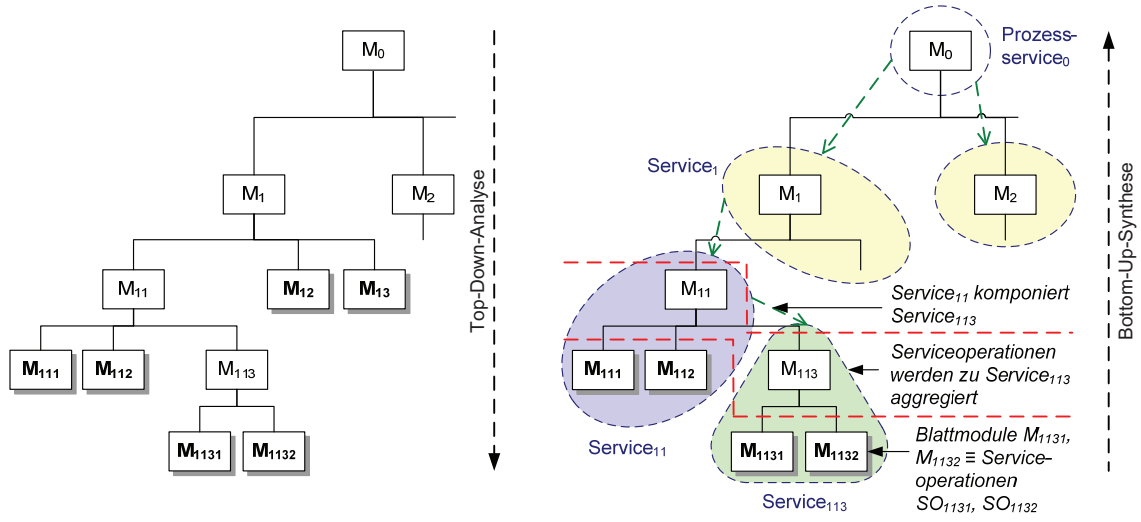


Bild 5-3: Ableitung der Servicehierarchie des SOA-Entwurfs

Durch eine Bottom-Up-Synthese kann aus der Modulhierarchie die Servicehierarchie des SOA-Entwurfs erarbeitet werden (vgl. Abschnitt 4.4.1). Die Blattmodule repräsentieren jeweils die Serviceoperationen. Ihre korrespondierenden Designparameter beschreiben die Operationsschnittstelle. Ausgehend von der untersten Hierarchieebene werden die Blattmodule bzw. Serviceoperationen zu Services aggregiert. Auf den nächsthöheren Ebenen werden die Services entsprechend den zugrunde liegenden Teilprozessen zu Servicekompositionen zusammengestellt. Als Ergebnis dieser Synthese entsteht die Servicehierarchie der zu entwerfenden SOA inklusive der Kompositionsbeziehungen zwischen den Services. Die Struktur der SOA ist im rechten Teil der Bild 5-3 durch die farbigen Flächen dargestellt. Die Kompositionsbeziehungen werden durch die grünen Pfeile repräsentiert. Eine vollständige Übersicht der Servicehierarchie des Fallbeispiels ist in Bild C-4 in Anhang C.2 gegeben. Die Servicehierarchie ist auch in der Entwurfsmatrix in Bild C-3 durch farbige Hinterlegungen angedeutet.

5.3 Darstellung des SOA-Entwurfs mit der UML

Eine Architekturbeschreibung umfasst eine Menge von Modellen in Form von Textspezifikationen oder grafischen Darstellungen, welche die Architektur dokumentieren.[203] Diese Beschreibung lässt sich in unterschiedliche Standpunkte unterteilen. Jeder *Standpunkt* fasst verwandte Betrachtungsaspekte zusammen und definiert Regeln zu ihrer Darstellung. Wird eine vorliegende Softwarearchitektur von einem bestimmten Standpunkt aus beschrieben, so entsteht eine konkrete *Sicht* auf diese Architektur. Hierbei kann u. a. zwischen einer statischen und einer dynamischen Sicht unterschieden werden. Die *statische Sicht* zeigt die Zerlegung der Architektur in ihre Komponenten und deren Beziehungen untereinander. Im Fall einer serviceorientierten Architektur entspricht dies der Darstellung der Servicehierarchie und der Kompositions- und Kopplungsbeziehungen. Die *dynamische Sicht* betrachtet das Systemverhalten zur Laufzeit in Form von Interaktionen und Kontrollflüssen. In einer SOA entspricht dies der dynamischen Interaktion der Services über den Nachrichtenaustausch.

Die in Abschnitt 5.2 entworfene serviceorientierte Architektur für das Fallbeispiel der Auftragsbearbeitung soll in den folgenden zwei Abschnitten mithilfe der Unified Modeling Language (UML) dargestellt werden. Die Architekturbeschreibung erfolgt vom statischen und dynamischen Standpunkt aus und wird mit geeigneten UML-Modellen dokumentiert. Gegenwärtig existiert noch keine etablierte Modellierungssprache für die Darstellung serviceorientierter Architekturen.[204] Deshalb werden in diesem Bericht bereits vorhandene Modelle und Notationselemente der UML verwendet. Meist können sie mit einer entsprechenden Interpretation problemlos für die Repräsentation eines SOA-Entwurfs eingesetzt werden. Teilweise sind jedoch Anpassungen und Erweiterungen der UML-Notation notwendig, um spezielle Aspekte eines SOA-Entwurfs darzustellen. An den jeweiligen Stellen wird gesondert darauf hingewiesen. Die Tabelle B-1 in Anhang B gibt einen Überblick über die verwendete UML-Notation und die korrespondierenden Bestandteile einer serviceorientierten Architektur.

---

203 Vgl. für diesen Absatz Behrens u. a. /Architekturbeschreibung/ 36-39

204 Erste Ansätze für eine Modellierung serviceorientierter Architekturen bieten die Service Component Architecture der Open SOA-Initiative (http://www.osoa.org), die Arbeiten von Baresi, Heckel, Thöne und Varró (vgl. Baresi u. a. /UML-Profile/) sowie ein Request for Proposal der Object Management Group (OMG) (vgl. OMG /UML Profile Services/).

5.3.1 Statische Sicht (Komponentendiagramm)

Für die Darstellung der statischen Sicht auf den SOA-Entwurf wurde das Komponentendiagramm der UML ausgewählt.[205] Es ist insbesondere für die Dokumentation von Softwarearchitekturen während der Entwurfsphase geeignet.[206] Es zeigt die Zergliederung des Gesamtsystems in seine modularen Bestandteile. Zudem werden die Beziehungen zwischen den Systemkomponenten aufgezeigt und ihr Verhalten wird durch die von ihnen realisierten und benötigten Schnittstellen spezifiziert. Diese Betrachtungsweise kommt den serviceorientierten Architekturprinzipien der Modularisierung, der losen Kopplung und des Information Hiding sehr nahe. Außerdem entspricht das UML-Konstrukt der Komponente am besten einem Service einer SOA.[207] Ein Service kapselt, ebenso wie eine Komponente, eine abgeschlossene Funktionalität.

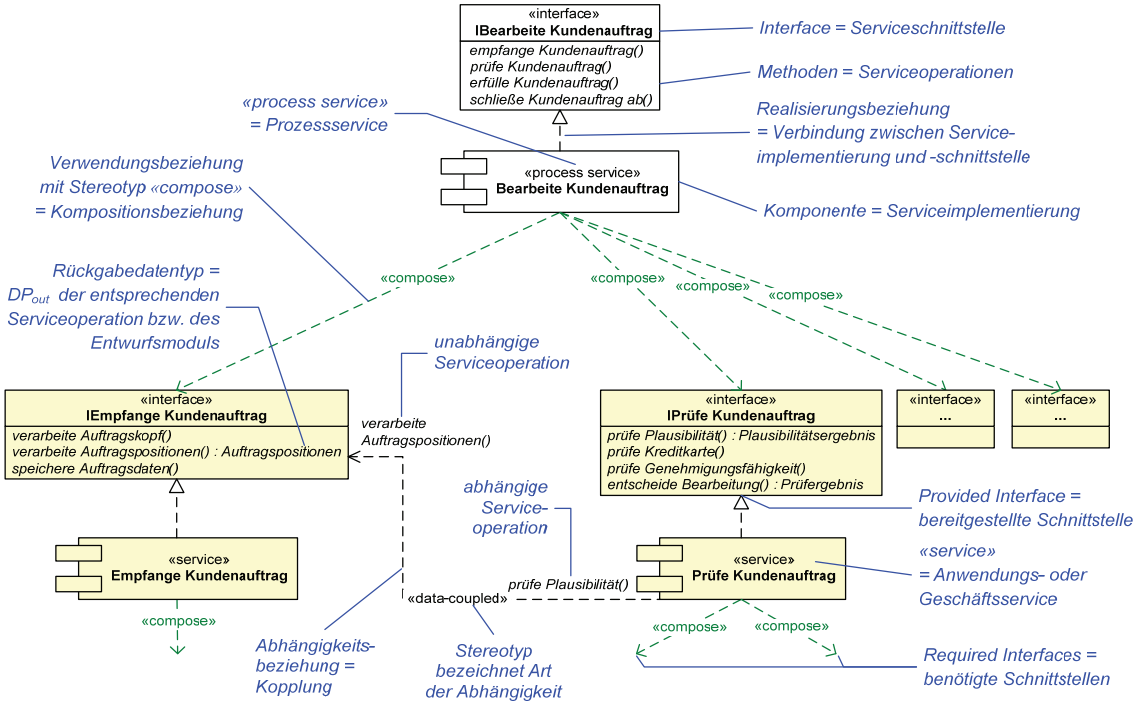


Bild 5-4: Ausschnitt des UML-Komponentendiagramms

Er stellt diese Funktionalität über seine Schnittstelle zur Verfügung und steht mit anderen Services über deren Schnittstellen in Verbindung. Die einzelnen Notationselemente der

205 Vgl. Behrens u. a. /Architekturbeschreibung/ 39
206 Vgl. für die folgenden Sätze Kecher /UML Handbuch/ 145-148
207 Vgl. Baresi u. a. /UML-Profile/ 5

UML zur Darstellung einer statischen Sicht und ihre Interpretation im Rahmen eines SOA-Entwurfs sind in Bild 5-4 dargestellt und werden nachfolgend einzeln beschrieben.

In der Servicehierarchie aus Bild C-4 wurden die Blattmodule der Entwurfsmatrix als Serviceoperationen interpretiert. Sie wurden dann gemäß der Struktur der Modulhierarchie zu Services aggregiert. Die Funktionalität dieser Services wird im Komponentendiagramm der UML durch *Interfaces* repräsentiert.[208] Das Interface-Element definiert die Serviceschnittstelle, welche sich aus den einzelnen Operationen des Service zusammensetzt. Die Benennung der Serviceoperationen orientiert sich an den jeweiligen funktionalen Anforderungen, die sie erfüllen.[209] Die Schnittstelle jeder Serviceoperation wird durch die Designparameter der realisierten funktionalen Anforderung gebildet. Aus Platzgründen werden die Eingangsparameter (DP_{in}) ausgeblendet und es wird nur der Datentyp des Rückgabewerts (DP_{out}) jeder Operation angegeben.[210]

Die eigentliche Serviceimplementierung wird separat von der Serviceschnittstelle durch eine *Komponente* im UML-Diagramm dargestellt. Diese Aufteilung verwirklicht das Prinzip der Trennung von Schnittstelle und Implementierung und ermöglicht die Entkopplung der Services untereinander. Serviceschnittstelle und -implementierung werden jeweils nach der zugrunde liegenden funktionalen Anforderung benannt.[211] Die Stereotypen «service»[212] bzw. «process service» wurden als Erweiterung der UML-Stereotypen definiert. Sie dienen der Unterscheidung der verschiedenen Servicetypen und kategorisieren die jeweilige Komponente bzw. den repräsentierten Service als Anwendungs- oder Geschäftsservice bzw. als Prozessservice (vgl. Abschnitt 3.4).
Die Verbindung zwischen einer Serviceimplementierung (Komponente) und der zugehörigen Serviceschnittstelle (Interface) wird durch eine *Realisierungsbeziehung*[213] angezeigt. Sie drückt aus, dass der entsprechende Service die in der Schnittstelle definierte Funktionalität realisiert und damit eine zur Schnittstelle konforme Implementierung

---

208 Vgl. OMG /UML/ 86

209 Die Namen der UML-Elemente entsprechen weitestgehend den Bezeichnungen der FAs und DPs aus der vollständigen Entwurfsmatrix des Fallbeispiels. Da die UML-Diagramme nur vorläufige Entwurfsergebnisse dokumentieren und keine direkten Vorgaben für die Implementierung darstellen sollen, wurden bei der Benennung der Elemente bewusst auch Leerzeichen und Umlaute übernommen (vgl. Kecher /UML Handbuch/ 35).

210 Vgl. OMG /UML/ 107

211 Zur Unterscheidung von Serviceschnittstelle und -implementierung wird einer Schnittstelle zusätzlich der Buchstabe „I“ für „Interface“ vorangestellt.

212 Vgl. Baresi u. a. /UML-Profile/ 4 f.

213 Vgl. OMG /UML/ 89 (InterfaceRealization) und 87 (Provided Interfaces)

bereitstellt. Jede Komponente ist somit vollständig durch die von ihr selbst angebotene Schnittstelle und die von ihr benötigten Schnittstellen anderer Komponenten definiert.[214]

Neben der reinen Dekomposition des SOA-Entwurfs in einzelne Services müssen auch die Abhängigkeiten und Beziehungen zwischen diesen Services berücksichtigt und dargestellt werden. Für diese Aspekte gibt es keine speziellen UML-Notationselemente. Deshalb werden in diesem Bericht passende, allgemeine Notationselemente der UML benutzt und mittels Stereotypen genauer spezifiziert.[215]

Gemäß der Forderung nach loser Kopplung werden die Beziehungen zwischen Services ausnahmslos über ihre Schnittstellen abgebildet.[216] Der jeweils abhängige Service verwendet die Schnittstelle des unabhängigen Service, um mit diesem zu interagieren.[217] Diese Bezugnahme wird durch *Verwendungsbeziehungen*[218] repräsentiert. Sie werden in der UML durch gestrichelte Pfeile dargestellt. Eine für SOA-Entwürfe spezifische Form dieser Beziehungen sind die *Kompositionsbeziehungen*. Sie setzen die hierarchische Anordnung der Services aus der Modul- und Servicehierarchie (vgl. Bild C-4) im Komponentendiagramm um. Ein übergeordneter Service komponiert einen oder mehrere untergeordnete Services, indem er deren Serviceoperationen in einer bestimmten Reihenfolge aufruft. Die komponierende Serviceoperation des übergeordneten Service trägt dabei den gleichen Namen wie der untergeordnete Service. Die Kompositionsbeziehung wird im UML-Diagramm durch den selbst definierten Stereotyp «compose» bezeichnet. Der Pfeil der Kompositionsbeziehung zeigt vom komponierenden Service zur Schnittstelle des untergeordneten Service. Beispielsweise komponiert in Bild 5-4 die Operation „empfange Kundenauftrag()“ des Service „Bearbeite Kundenauftrag“ die Operationen des untergeordneten Service „Empfange Kundenauftrag“.

Eine zweite Form der Beziehungen zwischen Services bzw. deren Operationen ergibt sich aus den Nichtdiagonalelementen der Entwurfsmatrix. Diese Elemente beschreiben die *Kopplungsbeziehungen* zwischen den Serviceoperationen. Um sie im Komponentendiagramm darzustellen, wird ebenfalls die Verwendungsbeziehung der UML genutzt. Sie zeigt an, dass der abhängige Service die Dienste des unabhängigen Service

214 Vgl. Kecher /UML Handbuch/ 147
215 Vgl. Kecher /UML Handbuch/ 85 und Baresi u. a. /UML-Profile/ 3
216 Vgl. Papazoglou, Yang /Design Methodology/ 55
217 Vgl. OMG /UML/ 87 (Required Interfaces)

benötigt, um seine eigene funktionale Anforderung zu erfüllen. Diese Abhängigkeiten müssen sich in einer späteren Implementierung nicht unbedingt in direkten Aufrufbeziehungen äußern. Die konkrete technische Realisierung wird in der Entwurfsphase noch offen gelassen (vgl. Abschnitt 4.4.2). Im Komponentendiagramm sollen lediglich die Abhängigkeitsbeziehungen aus der Entwurfsmatrix auf der Serviceebene wiedergegeben werden.

Die Kopplungen werden jeweils durch einen gestrichelten Pfeil dargestellt, der vom abhängigen Service zur Schnittstelle des unabhängigen Service zeigt. Um die Abhängigkeit auf Operationsebene genau zu spezifizieren, wurden die nachfolgenden, nicht in der UML enthaltenen Notationselemente eingeführt. Die Serviceoperationen, die an einer Kopplungsbeziehung beteiligt sind, werden an den jeweiligen Pfeilenden notiert. Sie entsprechen den Blattmodulen der Entwurfsmatrix, die durch das repräsentierte Nichtdiagonalelement gekoppelt sind. Die Art der Abhängigkeit wird durch selbst definierte Stereotypen verdeutlicht: «functional-coupled» für eine funktionale Abhängigkeit, «data-coupled» für eine Datenabhängigkeit und «control-coupled» für eine Steuerungsabhängigkeit. In Bild 5-4 besteht beispielsweise eine Datenabhängigkeit zwischen der Operation „prüfe Plausibilität()“ des Service „Prüfe Kundenauftrag“ und der Operation „verarbeite Auftragspositionen()“ des Service „Empfange Kundenauftrag“.

Kopplungsbeziehungen zwischen Operationen desselben Service werden nicht dargestellt. Auch Abhängigkeiten zwischen Serviceoperationen, die bereits durch eine Kompositionsbeziehung oder eine Kette von Kompositionsbeziehungen abgedeckt sind, werden nicht gesondert notiert. Somit erscheinen im Komponentendiagramm nur Abhängigkeiten zwischen Services in unterschiedlichen Kompositionszweigen.

Für das Fallbeispiel des Auftragsbearbeitungsprozesses wurde das vollständige Komponentendiagramm des SOA-Entwurfs in Bild C-5 in Anhang C aufgestellt. Quellen für diese Darstellung sind die vollständige Entwurfsmatrix aus Bild C-3 sowie die Servicehierarchie aus Bild C-4 mit der hierarchischen Struktur des SOA-Entwurfs und den Kompositionsbeziehungen.

---

218 Vgl. OMG /UML/ 136 f. (Usage) und 62 f. (Dependency)

5.3.2 Dynamische Sicht (Kommunikationsdiagramm)

Für die Darstellung der dynamischen Sicht auf den SOA-Entwurf wurde das Kommunikationsdiagramm der UML ausgewählt.[219] Das im vorangegangenen Abschnitt beschriebene Komponentendiagramm zeigt die einzelnen Services des SOA-Entwurfs und ihre statischen Kompositions- und Abhängigkeitsbeziehungen. Das Kommunikationsdiagramm ergänzt diese statische Sicht um die dynamischen Aspekte einer SOA.[220] Als Interaktionsdiagramm modelliert es den fallspezifischen Nachrichtenaustausch zwischen den Services, der zur Umsetzung eines konkreten Prozessablaufs erforderlich ist.

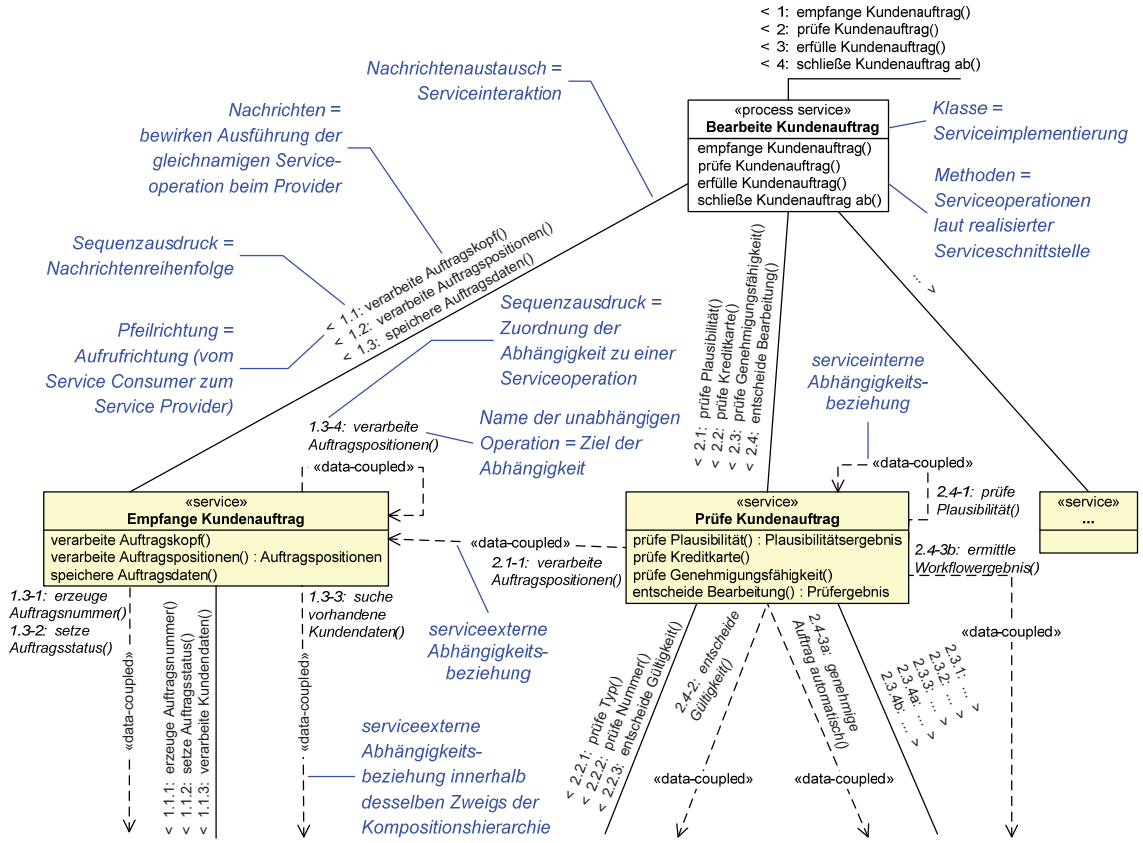


Bild 5-5: Ausschnitt des UML-Kommunikationsdiagramms

Bezogen auf das konkrete Fallbeispiel wird die Kommunikation der Services veranschaulicht, die zur Unterstützung des Auftragsbearbeitungsprozesses erforderlich ist.

219 Vgl. Behrens u. a. /Architekturbeschreibung/ 39

220 Vgl. für die folgenden Sätze Kecher /UML Handbuch/ 383 ff. und 339

Das Kommunikationsdiagramm stellt die Reihenfolge der einzelnen Nachrichten dar und ordnet die Serviceabhängigkeiten aus den Nichtdiagonalelementen der Entwurfsmatrix den jeweiligen Interaktionen zu.

Die Verwendung von Nachrichten entspricht der Forderung nach einer losen Kopplung zwischen den Services und der oftmals nachrichtenorientierten, dokumentenbasierten Kommunikation[221] in einer SOA. Das Kommunikationsdiagramm besitzt ein höheres Abstraktionsniveau als beispielsweise ein UML-Sequenzdiagramm. Es ist deshalb insbesondere für die Darstellung der dynamischen Abläufe und Abhängigkeiten in der Entwurfsphase geeignet. Die einzelnen Notationselemente des Diagramms, ihre Interpretation und Erweiterung zur Darstellung eines SOA-Entwurfs sind in Bild 5-5 dargestellt und werden nachfolgend einzeln erläutert.

Die Komponenten bzw. Services der statischen Struktur (vgl. Bild 5-4) werden im Kommunikationsdiagramm durch Klassen repräsentiert, die die gleichnamigen Komponenten realisieren. Im Gegensatz zum Komponentendiagramm stellen die Klassen hier konkrete Implementierungen der Serviceschnittstellen dar. Deshalb ist die Trennung von Serviceschnittstelle und -implementierung im Kommunikationsdiagramm aufgehoben. Die Operationen der realisierten Serviceschnittstelle werden als Methoden direkt in der jeweiligen Klasse und nicht in separaten Interface-Elementen aufgeführt.

Die Kompositionsbeziehungen aus der statischen Sicht werden in der dynamischen Sicht durch den Nachrichtenaustausch umgesetzt. Dieser ist jeweils mit einer durchgezogenen Kommunikationslinie zwischen den beteiligten Services dargestellt. An jeder Verbindung sind die einzelnen Nachrichten und deren Senderichtung angetragen. Eine eingehende Nachricht bewirkt beim Empfänger, dem Service Provider, die Ausführung der gleichnamigen Serviceoperation. Die Reihenfolge der Nachrichten wird durch deren Sequenznummern festgelegt. Sie orientiert sich an der Reihenfolge der Arbeitsschritte des realisierten Teilprozesses (vgl. auch die EPK in Bild C-1). Alternative Abläufe werden durch Kleinbuchstaben nach der Sequenznummer gekennzeichnet.
In Bild 5-5 sind beispielsweise der Prozessservice „Bearbeite Kundenauftrag“ und der Service „Empfange Kundenauftrag“ durch eine Kommunikationslinie miteinander verbunden. Diese realisiert die entsprechende Kompositionsbeziehung aus Bild 5-4. Mit

der Nachricht 1.1 fordert der Dienstnutzer „Bearbeite Kundenauftrag“ vom Dienstanbieter „Empfange Kundenauftrag“ die Ausführung der Serviceoperation „verarbeite Auftragskopf()“ an. Zur Erfüllung dieser Funktionalität benötigt der Service Provider wiederum die Dienste eines anderen, untergeordneten Service. Er sendet deshalb im Rahmen einer neuen Komposition die Nachrichten 1.1.1, 1.1.2 und 1.1.3 an den ihm untergeordneten Service „Verarbeite Auftragskopf“.

Während die Kompositionsbeziehungen durch den direkten Nachrichtenaustausch abgebildet werden, sind die Abhängigkeiten zwischen Services durch Verwendungsbeziehungen dargestellt. Im Gegensatz zum Komponentendiagramm werden im Kommunikationsdiagramm alle Kopplungselemente aus der Entwurfsmatrix angezeigt. Es existieren deshalb Abhängigkeiten zwischen Operationen desselben Service, Abhängigkeiten zwischen Services desselben Kompositionszweigs und Abhängigkeiten zwischen Services unterschiedlicher Kompositionszweige. Auch hier gilt, dass eine Abhängigkeit nicht unbedingt eine direkte Aufrufbeziehung impliziert, sondern allgemein eine Kopplung der beteiligten Services ausdrücken soll.
Besitzt eine Serviceoperation laut Entwurfsmatrix Abhängigkeiten in Form von Nichtdiagonalelementen in der entsprechenden Matrixzeile, so sind diese im Kommunikationsdiagramm am entsprechenden Service durch Verwendungsbeziehungen vermerkt. Wie auch im Komponentendiagramm sind die Verwendungsbeziehungen durch gestrichelte Pfeile vom abhängigen zum unabhängigen Service dargestellt. Die Art der Abhängigkeit wird durch die Stereotypen «functional-coupled», «data-coupled» und «control-coupled» dokumentiert. Die Zuordnung der Kopplungsbeziehungen zur entsprechenden Serviceoperation erfolgt durch einen Sequenzausdruck. Dieser entspricht der Nachrichtensequenznummer der betrachteten Operation und wird mit einem Bindestrich sowie einer fortlaufenden Nummer ergänzt. Nach diesem Sequenzausdruck wird der Name der unabhängigen Operation angeführt, zu der die Kopplung besteht. Die Angabe dieser Kopplungsbeziehungen sowie ihre Beschreibung durch Sequenznummer und Operationsname erweitert die übliche UML-Notation eines Kommunikationsdiagramms. Sie ist erforderlich, um die in der Entwurfsmatrix definierten Kopplungen zwischen den Serviceoperationen in der dynamischen Interaktion der Services

[221] Vgl. Krafzig, Banke, Slama /Enterprise SOA/ 45 f.

zu berücksichtigen. Mithilfe dieser Angaben können für jede aufgerufene Serviceoperation die jeweiligen Abhängigkeiten laut Entwurfsmatrix zu anderen Operationen aufgelöst und dargestellt werden.

Beispielsweise führt in Bild 5-5 die Nachricht 2.1 an den Service „Prüfe Kundenauftrag“ zur Ausführung seiner Operation „prüfe Plausibilität()“. Laut Entwurfsmatrix besitzt diese Operation eine Kopplung zur Operation „verarbeite Auftragspositionen()“ des Service „Empfange Kundenauftrag“. Diese Abhängigkeit wird durch einen gestrichelten Pfeil dargestellt. Der Ausgangspunkt der Kopplungsbeziehung, die abhängige Operation „prüfe Plausibilität()“, wird durch den Sequenzausdruck 2.1-1 definiert. Das Ziel der Kopplungsbeziehung wird durch den Namen der unabhängigen Operation „verarbeite Auftragspositionen()“ festgelegt. Da es sich um eine Datenabhängigkeit handelt, trägt der Pfeil den Stereotyp «data-coupled».

Das vollständige Kommunikationsdiagramm für das Fallbeispiel ist in Bild C-6 in Anhang C dargestellt. Es zeigt die Komposition der Services in Form des Nachrichtenaustauschs und die Abhängigkeiten in der entworfenen SOA, wenn die einzelnen Teilprozesse und Prozessschritte der Auftragsbearbeitung umgesetzt werden.

6 Bewertung des Beitrags von Axiomatic Design im Entwurf serviceorientierter Architekturen

In den vorangegangenen Kapiteln 2 und 4 wurde die Entwurfsmethodik Axiomatic Design vorgestellt und für den Entwurf serviceorientierter Architekturen angepasst. In Kapitel 5 wurde die Anwendung dieser Methode anhand eines Fallbeispiels demonstriert. Bisher blieb jedoch ungeklärt, welchen konkreten Beitrag Axiomatic Design im Rahmen des SOA-Entwurfs liefert. Deshalb soll in diesem Kapitel vor allem die Qualität der Entwurfsergebnisse im Hinblick auf die Architekturprinzipien serviceorientierter Architekturen kritisch bewertet werden. Zu diesem Zweck wird in Abschnitt 6.1 ein Konzept zur Messung der Entwurfsqualität erarbeitet. In diesem Zusammenhang werden Komplexitätsmetriken definiert, welche die Einschätzung des Nutzens und der Auswirkungen des Einsatzes von Axiomatic Design unterstützen sollen. Die theoretische oder empirische Validierung der entwickelten Komplexitätsmetriken ist nicht Gegenstand dieses Berichtes. Ebenso wird keine vollständige Bewertung des mit Axiomatic Design erarbeiteten SOA-Entwurfs vorgenommen. Dieser Aspekt wird in einem nachfolgenden Bericht gesondert betrachtet. In Abschnitt 6.2 dieses Kapitels wird abschließend auf die Vorzüge, Schwächen und Grenzen der Anwendung von Axiomatic Design im Entwurf serviceorientierter Architekturen eingegangen.

6.1 Messkonzept für die Bewertung der Qualität von SOA-Entwürfen

6.1.1 Einführung in die Bewertung der Entwurfsqualität

Ziel des nachfolgend zu entwickelnden Messkonzepts ist es, die Qualität einer mit Axiomatic Design entworfenen SOA messbar zu machen. Qualität bezeichnet im Softwareumfeld die „Gesamtheit der Merkmale eines Softwareprodukts, die sich auf dessen Eignung beziehen, festgelegte oder vorausgesetzte Erfordernisse zu erfüllen“[222]. Für einige ausgewählte Qualitätsmerkmale sollen nachfolgend geeignete Metriken entwickelt werden. Eine Metrik ist ein Qualitätsindikator und „definiert, wie eine Kenngröße eines

[222] DIN /Bewerten von Softwareprodukten. DIN 66272/ 3

Software-Produkts … gemessen wird“[223]. In diesem Kapitel wird als Softwareprodukt der erarbeitete SOA-Entwurf betrachtet. Da es sich hierbei um ein Ergebnis der Entwurfsphase handelt, sind die entwickelten Maße den Entwurfsmetriken[224] zuzuordnen. Die Werte der entwickelten Metriken sollen Rückschlüsse auf die Erfüllung folgender, in Abschnitt 3.5 definierten Qualitätsmerkmale einer SOA erlauben: lose Kopplung, hohe Autonomie und angemessene Granularität der Services. Aufgrund ihrer abstrakten Natur ist die Autonomie der Services nicht direkt messbar. Die Berücksichtigung der Autonomie beim Zuschneiden eines Service führt jedoch dazu, dass dieser einen abgeschlossenen, funktionalen Aspekt kapselt.[225] Seine Operationen sind bezüglich dieses Aspekts zusammengehörig bzw. kohärent. Folglich kann aus der Kohäsion eines Service auf den Grad seiner Autonomie geschlossen werden. Deshalb soll, statt der abstrakten Autonomie, die mit ihr verbundene Kohäsion der Services untersucht werden.

Durch das zu erarbeitende Messkonzept soll die Komplexität eines SOA-Entwurfs bewertet werden. Dies schlägt eine Brücke zur Beurteilung der Entwurfskomplexität im Rahmen der Anwendung des Informationsaxioms von Axiomatic Design (vgl. Abschnitte 2.4.2 und 4.5.2). Der Informationsgehalt eines Entwurfs bewertet dessen Erfolgswahrscheinlichkeit und steht in enger Verbindung zu dessen Komplexität.[226] Die in Axiomatic Design allgemein formulierte Entwurfskomplexität korrespondiert mit der im Software Engineering verwendeten Komplexität eines Softwareentwurfs[227]. Deshalb können die im Folgenden zu entwickelnden Komplexitätsmetriken verwendet werden, um die Komplexität bzw. den Informationsgehalt eines SOA-Entwurfs zu ermitteln.[228] Sie können aber auch parallel zum Unabhängigkeits- und Informationsaxiom von Axiomatic Design zur Bewertung eines Entwurfs hinzugezogen werden.

Die Komplexität einer Softwarearchitektur wird maßgeblich durch die verwendeten Abstraktionen und die sachgerechte Modularisierung des Entwurfs bestimmt.[229] Ein Softwaremodul sollte zusammengehörige Funktionalität zu einer logischen Einheit

---

223 Balzert /Software-Qualitätssicherung/ 225
224 Vgl. Ebert /Qualitätsmanagement/ 63
225 Vgl. Erl /Service-Oriented Architecture/ 303 f.
226 Vgl. Suh /Axiomatic Design/ 472
227 Vgl. Henderson-Sellers /Object-Oriented Metrics/ 56 f.
228 Vgl. Pimentel, Stadzisz /Use Case/ 6 ff. und die Anmerkungen zum Ansatz der Autoren auf Seite 50.
229 Vgl. Balzert /Software-Entwicklung/ 1024

aggregieren.[230] Die Gesamtkomplexität des Entwurfs einer Softwarearchitektur ergibt sich aus mehreren Faktoren. Unter einem lokalen Aspekt wird die *intramodulare Komplexität* betrachtet. Sie schätzt den Aufbau der Module und damit ihre Kohäsion ein. Unter einem zweiten, strukturellen Aspekt wird die *intermodulare Komplexität* betrachtet. Hierbei werden die Verbindungen der Module untereinander und damit ihre Kopplung bewertet. Ein weiterer Aspekt, die Granularität der Module, nimmt eine Zwischenstellung ein. Die Granularität wird zwar durch den Zuschnitt und die innere Struktur eines Moduls bestimmt, besitzt aber über die Modulschnittstelle auch eine Wirkung auf die Beziehungen des Moduls zu seiner Umgebung.

Bezogen auf den Entwurf serviceorientierter Architekturen werden zur Beurteilung der Komplexität und damit der Qualität der Entwurfsergebnisse die Serviceoperationen als Module und die Services als Zusammenfassung dieser Module zu Komponenten betrachtet. Der Zuschnitt und die innere Struktur der Services und Serviceoperationen sowie ihre Beziehungen untereinander sind wesentliche Indikatoren zur Bewertung der Qualitätsmerkmale[231]. Für jeden der drei genannten Komplexitätsaspekte – Kopplung, Kohäsion und Granularität – werden in den folgenden Abschnitten geeignete Metriken für SOA-Entwürfe entwickelt. Zuerst werden jeweils die theoretischen Grundlagen des zu messenden Qualitätsaspekts erläutert. Anschließend erfolgt die Herleitung und Definition der Metriken. Zum Schluss wird ihre Anwendung und Interpretation an einem Ausschnitt des in Kapitel 5 erstellten SOA-Entwurfs demonstriert.

6.1.2 Kopplung

Theoretische Grundlagen der Kopplung

Stevens, Myers und Constantine haben 1974 die Kopplung als Maß für die Stärke der Verbindungen zwischen zwei Softwaremodulen beschrieben.[232] In dem vorliegenden Bericht wird die Kopplung allgemein als Stärke der Beziehungen eines Moduls zu seiner Systemumgebung, also zu allen anderen Modulen, betrachtet.[233] Eine hohe Kopplung im Entwurf wirkt sich negativ auf die Änderbarkeit und Wartbarkeit der implementierten

230 Vgl. Balzert /Software-Qualitätssicherung/ 572
231 Vgl. Balzert /Software-Qualitätssicherung/ 258
232 Vgl. Stevens, Myers, Constantine /Structured Design/ 117
233 Vgl. Briand, Morasca, Basili /Measures for Design/ 732

Software aus.[234] Sie führt zur unkontrollierten Fortpflanzung von Fehlern über Modulgrenzen hinaus und erhöht die Komplexität des Entwurfs und der Implementierung. Aufgrund der hierarchischen Dekomposition beim Entwurf einer SOA mit Axiomatic Design sind gewisse Abhängigkeiten zwischen den entstehenden Modulen notwendig und nicht vollständig zu vermeiden. Für die Gestaltung des SOA-Entwurfs ist dennoch eine möglichst niedrige Kopplung der Services untereinander anzustreben.[235]

Zur differenzierten Analyse der Kopplung wurden in der Literatur verschiedene Abstufungen beschrieben. Mit steigender Kopplungsstärke sind sie in Tabelle 6-1 aufgeführt.

Kopplungsgrad	**Eigenschaften**
Keine Kopplung (No Coupling)	Zwischen den betrachteten Modulen besteht keine Beziehung.
Datenkopplung (Data Coupling)	Die gekoppelten Module sind über Datenflüsse verbunden, wobei nur elementare Datenparameter ausgetauscht werden.
Datenstrukturkopplung (Stamp Coupling)	Die gekoppelten Module sind über Datenflüsse verbunden. Als Parameter werden zusammengesetzte Datenstrukturen ausgetauscht.
Steuerungskopplung (Control Coupling)	Zwischen den gekoppelten Modulen werden Steuerinformationen in Form von Kontrollparametern (z. B. Flags) ausgetauscht. Dadurch wird der interne Ablauf des abhängigen Moduls steuernd beeinflusst.
Globale Kopplung (Common Coupling)	Die gekoppelten Module haben Zugriff auf einen gemeinsamen globalen Datenbereich.
Inhaltsbezogene Kopplung (Content Coupling)	Ein Modul nimmt direkten Bezug auf das Innere eines anderen Moduls (Durchbrechen des Black Box-Prinzips).

Tabelle 6-1: Unterscheidung verschiedener Kopplungsgrade[236]

Für die Bewertung der Kopplung innerhalb eines SOA-Entwurfs sind hauptsächlich die Datenkopplung und Datenstrukturkopplung in Form der Datenabhängigkeiten sowie die Steuerungskopplung in Form der Steuerungsabhängigkeiten zwischen Serviceoperationen

234 Vgl. für die folgenden Sätze Darcy u. a. /Structural Complexity/ 986 und Vogel u. a. /Software-Architektur/ 116
235 Vgl. Papazoglou, Yang /Design Methodology/ 58 und Humm, Voß, Hess /Regeln/ 401

relevant. Globale und inhaltsbezogene Kopplung sollten in einer SOA bei Befolgung der Architekturprinzipien der Serviceorientierung und Verwendung moderner Programmierparadigmen nicht auftreten. Services kommunizieren ausschließlich über ihre Schnittstellen[237], so dass die Verwendung globaler Datenbereiche oder die Bezugnahme auf die Implementierungsdetails eines anderen Service unterbunden werden.

Im Umfeld der Kopplungsmetriken wurden für fast alle Programmierparadigmen sehr intensive Forschungen und empirische Studien durchgeführt. Eine Übersicht über einige Metriken zur Bewertung der Kopplung und die entsprechenden Literaturstellen bietet Tabelle 6-2.

Traditionelle Kopplungsmetriken	
Fan-in/Fan-out	Henry, Kafura /Software Systems' Structure/ 102 f.
Relative Intermodule Complexity	Henderson-Sellers /Object-Oriented Metrics/ 86
Coupling Between Modules	Fenton, Pfleeger /Software Metrics/ 310 f.
Spezielle Kopplungsmetriken für objektorientierte Software	
Coupling Between Objects (CBO)	Chidamber, Kemerer /Metrics Suite/ 486
Message Passing Coupling (MPC)	Briand, Daly, Wüst /Coupling Measurement/ 101
Response For a Class (RFC)	Chidamber, Kemerer /Metrics Suite/ 487 f.
Data Abstraction Coupling (DAC)	Briand, Daly, Wüst /Coupling Measurement/ 102

Tabelle 6-2: Bereits existierende Kopplungsmetriken

Die bekannteste Kopplungsmetrik für objektorientierte Software ist Coupling Between Objects (CBO). Sie entstammt der Metriksammlung von Chidamber und Kemerer.[238] Das Maß CBO ermittelt die Anzahl der Klassen, mit denen die betrachtete Klasse in Beziehung steht. Eine solche Beziehung ist gegeben, wenn eine Klasse die Methoden oder Attribute der anderen benutzt. Da ein Service keine eigenen Attribute zur Repräsentation von Daten besitzt, ist diese Metrik nicht vollständig auf SOA-Entwürfe anwendbar. Auch andere verbreitete Kopplungsmetriken aus Tabelle 6-2 sind nicht einsetzbar, da sie entweder

236 Vgl. für die einzelnen Kopplungsgrade Page-Jones /Systemdesign/ 74, 77, 80, 85 und 89
237 Vgl. Papazoglou, Yang /Design Methodology/ 55

spezielle objektorientierte Konzepte wie Vererbung und Polymorphie berücksichtigen oder auf Quellcode- und Implementierungsdetails basieren, die während der Entwurfsphase noch nicht vorliegen.

Eine allgemein anwendbare und viel zitierte Kopplungsmetrik ist die Fan-in/Fan-out-Metrik von Henry und Kafura.[239] Sie bewertet die von einem Modul ausgehenden und in das Modul eingehenden Informationsflüsse und schließt daraus auf die Kopplung des Moduls zu seiner Umgebung. Auf der Grundlage dieser Metrik werden im folgenden Abschnitt Kopplungsmaße für SOA-Entwürfe entwickelt.

Definition von Kopplungsmetriken für SOA-Entwürfe

Die nachfolgend entwickelten Metriken zur Bewertung der Kopplung in einem SOA-Entwurf basieren auf der *Fan-in/Fan-out-Metrik* von Henry und Kafura.[240] Die Fan-in/Fan-out-Metrik betrachtet den Informationsfluss und damit die Kommunikationsbeziehungen zwischen Softwaremodulen. Der *Fan-out* ist die Zahl der vom betrachteten Modul ausgehenden Informationsflüsse und der *Fan-in* die Anzahl der in das Modul eingehenden Informationsflüsse. Ein Informationsfluss kann z. B. durch die Parameterübergabe bei einem Modulaufruf entstehen. Die Kopplungskomplexität C eines Moduls M_i wurde von Henry und Kafura folgendermaßen definiert:[241]

$$C(M_i) = Codelänge(M_i) \cdot \left(fanin(M_i) \cdot fanout(M_i)\right)^2 \qquad (9)$$

Laut empirischen Untersuchungen ist nur die Höhe des Fan-out eines Moduls für dessen Kopplung relevant.[242] Deshalb wird in diesem Bericht für die Beurteilung der Kopplung in einem SOA-Entwurf nur der Fan-out der Module bzw. Serviceoperationen betrachtet. Der Fan-out wird im SOA-Entwurf als die Menge von Kopplungen verstanden, die von einer Serviceoperation ausgehen. Die Abhängigkeiten einer Serviceoperation SO_i sind in der Entwurfsmatrix von Axiomatic Design durch die Nichtdiagonalelemente A_{ij} ($j=1,\ldots, n$ und $j \neq i$) in der Zeile des entsprechenden Entwurfsmoduls M_i definiert. Zur Bewertung der Kopplung einer Serviceoperation zu ihrer Umgebung werden deshalb die Nichtdiagonalelemente der jeweiligen Operation in der Entwurfsmatrix untersucht. Jedes

238 Vgl. für die zwei folgenden Sätze Chidamber, Kemerer /Metrics Suite/ 486
239 Vgl. für diesen und den folgenden Satz Henry, Kafura /Software Systems' Structure/ 102
240 Vgl. für diesen Absatz Henry, Kafura /Software Systems' Structure/ 102-104
241 Vgl. Henry, Kafura /Software Systems' Structure/ 103

dieser Kopplungselemente wird hinsichtlich zweier Merkmale beurteilt: die Reichweite der Abhängigkeit und die Art der Abhängigkeit. Die Ausprägungen dieser zwei Merkmale für ein Nichtdiagonalelement bestimmen die Stärke der repräsentierten Kopplungsbeziehung. Die möglichen Merkmalswerte sind in Tabelle 6-3 aufgeführt.

Merkmal 1: Reichweite der Abhängigkeit			
Merkmalsausprägungen	Stärke der Kopplung	Gewicht	
serviceintern (si)	*schwach* (Kopplung ist i. d. R. sinnvoll und notwendig, leicht überschaubar)	g_{si}	1
serviceextern, innerhalb des Kompositionszweigs (seK)	*mittel* (innerhalb des Kompositionszweigs bestehen stets Abhängigkeiten)	g_{seK}	2
serviceextern (se)	*stark* (Kopplung von logisch unabhängigen Services)	g_{se}	6
Merkmal 2: Art der Abhängigkeit			
Merkmalsausprägungen	Stärke der Kopplung	Gewicht	
Datenabhängigkeit (D)	*schwach* (nur Kopplung über Daten; impliziert nicht unbedingt einen direkten Aufruf)	g_D	1
funktionale Abhängigkeit (F)	*mittel* (Service und Schnittstelle der aufgerufenen Operation müssen bekannt sein)	g_F	2
Steuerungsabhängigkeit (S)	*stark* (Beeinflussung der Ablauflogik durch Kontrollparameter)	g_S	5

Tabelle 6-3: Kopplungsmerkmale der Nichtdiagonalelemente und Ausprägungen

Hinsichtlich der Reichweite einer Kopplung wird unterschieden zwischen serviceinternen Abhängigkeiten zwischen Operationen desselben Service (si), Abhängigkeiten zwischen Operationen verschiedener Services innerhalb desselben Zweigs der Kompositionshierarchie (seK) und serviceexternen Abhängigkeiten zwischen Operationen

[242] Vgl. Henderson-Sellers /Object-Oriented Metrics/ 86

verschiedener Services in getrennten Kompositionszweigen (se) (vgl. auch Bild 5-5). Die Ausprägungen für das Merkmal der Abhängigkeitsart wurden bereits in Abschnitt 4.4.2 eingehend erläutert.

Jede Merkmalsausprägung aus Tabelle 6-3 wird gemäß ihrer Auswirkung auf die Kopplungsstärke mit einem individuellen Gewicht bewertet. Eine hohe Gewichtung zeigt an, dass die entsprechende Ausprägung zu einer starken Kopplung beiträgt. Die konkrete Festlegung der Gewichte muss entsprechend den Entwurfszielen hinsichtlich des zulässigen Ausmaßes der Kopplung erfolgen. Hohe Gewichtungen führen zu höheren Kopplungsmaßen und fördern einen restriktiven Umgang mit bzw. eine Vermeidung der Kopplung.

Die Kopplungswirkung eines Nichtdiagonalelements wird durch die Multiplikation der zwei Gewichte berechnet, die seinen konkreten Merkmalsausprägungen zugeordnet sind. Zur näheren Analyse der Kopplung der Bestandteile des betrachteten SOA-Entwurfs werden folgende Mengen definiert (vgl. Abschnitt 4.4.1):

S	Menge aller Services des betrachteten SOA-Entwurfs
$SO(S_h)$	Menge aller Serviceoperationen des Service S_h

Weiterhin werden die Nichtdiagonalelemente einer Serviceoperation SO_i entsprechend ihrer Merkmalswerte zu Fan-out-Mengen *FO* gruppiert:

$FO(SO_i)$	Menge der Fan-out-Elemente/Nichtdiagonalelemente/Abhängigkeiten von Serviceoperation/Modul *i* $FO(SO_i) = \{ A_{ij} \mid j = 1,\ldots,n,\ j \neq i \}$
$FO_{si,D}(SO_i)$	Menge serviceinterner Datenabhängigkeiten
$FO_{si,F}(SO_i)$	Menge serviceinterner funktionaler Abhängigkeiten
$FO_{si,S}(SO_i)$	Menge serviceinterner Steuerungsabhängigkeiten
$FO_{seK,D}(SO_i)$	Menge serviceexterner, im Kompositionszweig befindlicher Datenabhängigkeiten
$FO_{seK,F}(SO_i)$	Menge serviceexterner, im Kompositionszweig befindlicher funktionaler Abhängigkeiten
$FO_{seK,S}(SO_i)$	Menge serviceexterner, im Kompositionszweig befindlicher Steuerungsabhängigkeiten

$FO_{se,D}(SO_i)$ Menge serviceexterner Datenabhängigkeiten

$FO_{se,F}(SO_i)$ Menge serviceexterner funktionaler Abhängigkeiten

$FO_{se,S}(SO_i)$ Menge serviceexterner Steuerungsabhängigkeiten

Die Kopplung einer Serviceoperation wird ermittelt, indem alle ihre Abhängigkeiten zu anderen Serviceoperationen gewichtet und aufaddiert werden. Somit ergibt sich die *gewichtete Serviceoperationskopplung (GSOK)* für die betrachtete Serviceoperation SO_i, wenn die Mengenelemente mit ihren spezifischen Gewichten bewertet werden:

$$\begin{aligned} &GSOK(SO_i) = \\ &|FO_{si,D}(SO_i)| \cdot g_{si} \cdot g_D + |FO_{si,F}(SO_i)| \cdot g_{si} \cdot g_F + |FO_{si,S}(SO_i)| \cdot g_{si} \cdot g_S + \\ &|FO_{seK,D}(SO_i)| \cdot g_{seK} \cdot g_D + |FO_{seK,F}(SO_i)| \cdot g_{seK} \cdot g_F + |FO_{seK,S}(SO_i)| \cdot g_{seK} \cdot g_S + \\ &|FO_{se,D}(SO_i)| \cdot g_{se} \cdot g_D + |FO_{se,F}(SO_i)| \cdot g_{se} \cdot g_F + |FO_{se,S}(SO_i)| \cdot g_{se} \cdot g_S \end{aligned} \quad (10)$$

Die *Servicekopplung (SK)* beschreibt die Gesamtkopplung eines Service S_h und berechnet sich durch die Addition aller Serviceoperationskopplungswerte seiner Operationen:

$$SK(S_h) = \sum_{SO_i \in SO(S_h)} GSOK(SO_i) \quad (11)$$

Die *durchschnittliche Serviceoperationskopplung (DSOK)* repräsentiert die durchschnittliche Kopplung der Operationen des betrachteten Service. Sie setzt die Servicekopplung ins Verhältnis zur Anzahl der Serviceoperationen:

$$DSOK(S_h) = \frac{SK(S_h)}{|SO(S_h)|} = \frac{\sum_{SO_i \in SO(S_h)} GSOK(SO_i)}{|SO(S_h)|} \quad (12)$$

Für den zu untersuchenden SOA-Entwurf kann auch die durchschnittliche Gesamtkopplung bestimmt werden. Diese *durchschnittliche Servicekopplung (DSK)* ist der Mittelwert der Servicekopplung (*SK*) in Bezug auf alle Services des Entwurfs:

$$DSK = \frac{\sum_{S_h \in S} SK(S_h)}{|S|} \quad (13)$$

Die in den Formeln (10) bis (13) definierten Metriken ermöglichen die Analyse der Kopplung auf unterschiedlichen Aggregationsstufen des SOA-Entwurfs. Das Maß *GSOK* beschreibt die Kopplung einzelner Serviceoperationen. Hohe Werte[243] weisen auf eine starke Kopplung zur Umgebung hin. Die Maße *SK* und *DSOK* untersuchen die Kopplung eines einzelnen Service, indem sie die Kopplungswerte seiner Operationen aggregieren. Hohe Werte von *SK* oder *DSOK* weisen auf problematische Servicekandidaten hin, die besonders stark an andere Services gekoppelt sind. Die Metrik *DSK* ermöglicht schließlich, eine Gesamtbewertung des SOA-Entwurfs. Sie kann insbesondere zum Vergleich verschiedener Alternativentwürfe herangezogen werden. Es ist eine möglichst geringe durchschnittliche Kopplung der Services anzustreben.

Eine gewisse Kopplung ist aufgrund der Aufgabenteilung in einer SOA und der damit verbundenen Nutzung der Dienste anderer Serviceanbieter notwendig. Ein hoher Wert eines Kopplungsmaßes weist jedoch auf eine große Anzahl solcher Beziehungen und damit eine unerwünschte Abhängigkeit der Serviceoperation bzw. des Service von der Umgebung hin. Dies verringert die Wiederverwendbarkeit des Service oder seiner Operationen in anderen Kompositionen. Die Analyse der Kopplungsmaße zeigt somit, ähnlich wie die Fan-in/Fan-out-Metrik, kritische Punkte in der SOA auf.[244] Mögliche Ursachen für eine starke Kopplung könnten eine unangemessene Verfeinerung[245] während der Dekomposition oder eine fehlende Abstraktionsschicht[246] zwischen dem betrachteten Service und den von ihm benutzten Services sein. Eine solche Abstraktion kann z. B. in Form eines zwischengeschalteten, komponierenden Prozessservice eingeführt werden.

Anwendung der definierten Kopplungsmetriken

Die Gewichtung der Nichtdiagonalelemente und die Berechnung der Kopplungsmaße soll nun auszugsweise anhand der Entwurfsmatrix des Fallbeispiels aus Abschnitt 5.2 demonstriert werden. Hierzu wurde der Service „Genehmige Auftrag manuell“ (S_{235}) ausgewählt. Er besitzt zwei Serviceoperationen SO_{2351} und SO_{2352}. Für jede dieser

243 Die genaue Schwelle für die Einstufung eines Kopplungswertes als „hoch“ hängt von dem konkreten Entwurfskontext und den jeweiligen Anforderungen hinsichtlich Wiederverwendung, Granularität und Wartbarkeit der Services ab.

244 Vgl. Henry, Kafura /Software Systems’ Structure/ 110

245 Vgl. Dumke /Maß/ 75

246 Vgl. Henry, Kafura /Software Systems’ Structure/ 105

Operationen müssen zuerst die Nichtdiagonalelemente analysiert und gewichtet werden, um den Wert *GSOK* zu bestimmen.

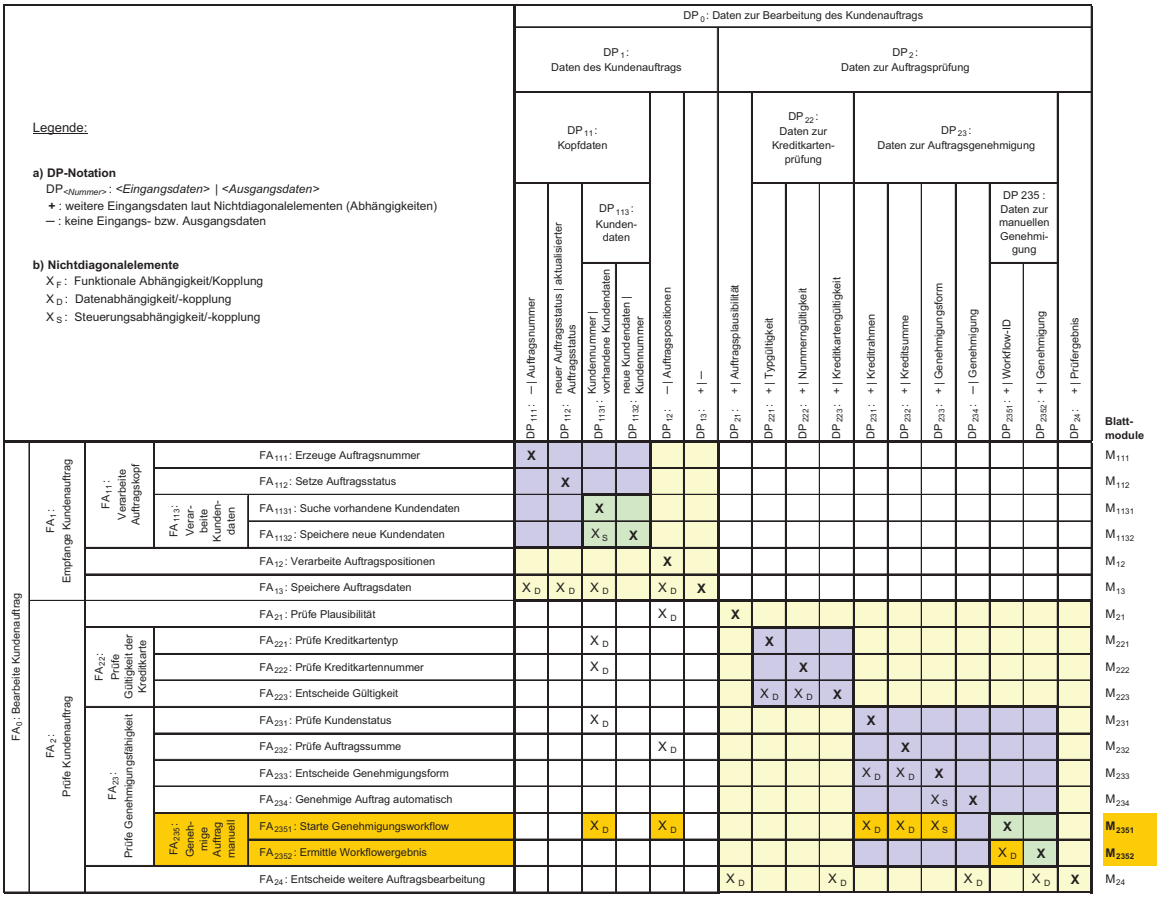


Bild 6-1: Berechnung der Kopplungsmaße anhand der Entwurfsmatrix

Wie aus Bild 6-1 ersichtlich ist, besitzt die Serviceoperation „Starte Genehmigungsworkflow“ (M_{2351}) fünf Nichtdiagonalelemente. Zwei davon sind Datenabhängigkeiten zu Operationen von Services außerhalb des Kompositionszweigs (se,D). Drei Kopplungselemente beziehen sich auf einen Service innerhalb desselben Kompositionszweigs (seK). Es handelt sich hierbei um zwei Datenabhängigkeiten (seK,D) und eine Steuerungsabhängigkeit (seK,S). Diese Nichtdiagonalelemente werden entsprechend ihren Merkmalswerten zu den oben beschriebenen Mengen zugeordnet:

$$
\begin{aligned}
FO(SO_{2351}) &= \{A_{2351,1131}, A_{2351,12}, A_{2351,231}, A_{2351,232}, A_{2351,233}\} &\rightarrow |FO(SO_{2351})| &= 5\\
FO_{se,D}(SO_{2351}) &= \{A_{2351,1131}, A_{2351,12}\} &\rightarrow |FO_{se,D}(SO_{2351})| &= 2\\
FO_{seK,D}(SO_{2351}) &= \{A_{2351,231}, A_{2351,232}\} &\rightarrow |FO_{seK,D}(SO_{2351})| &= 2\\
FO_{seK,S}(SO_{2351}) &= \{A_{2351,233}\} &\rightarrow |FO_{seK,S}(SO_{2351})| &= 1
\end{aligned}
$$

$$FO_{si,D} = FO_{si,F} = FO_{si,S} = FO_{seK,F} = FO_{se,F} = FO_{se,S} = \varnothing$$

Die Berechnung des Kopplungsmaßes *GSOK* für die Operation SO_{2351} ergibt sich aus der Betrachtung der Mächtigkeit dieser Mengen gemäß Formel (10):

$$\begin{aligned} & GSOK(SO_{2351}) \\ &= |FO_{se,D}(SO_{2351})| \cdot g_{se} \cdot g_D + |FO_{seK,D}(SO_{2351})| \cdot g_{seK} \cdot g_D + |FO_{seK,S}(SO_{2351})| \cdot g_{seK} \cdot g_S \\ &= 2 \cdot 6 \cdot 1 + 2 \cdot 2 \cdot 1 + 1 \cdot 2 \cdot 5 = 26 \end{aligned}$$

Die Operation „Ermittle Workflowergebnis“ (M_{2352}) besitzt nur eine Datenabhängigkeit, die sich auf die Operation SO_{2351} innerhalb desselben Service (si,D) bezieht. Die Serviceoperationskopplung berechnet sich wie folgt:

$$FO(SO_{2352}) = FO_{si,D}(SO_{2352}) = \{A_{2352,2351}\}$$
$$GSOK(SO_{2352}) = |FO_{si,D}(SO_{2352})| \cdot g_{si} \cdot g_D = 1 \cdot 1 \cdot 1 = 1$$

Somit ergibt sich das Gesamtkopplungsmaß *SK* für den Service „Genehmige Auftrag manuell“ nach Formel (11) als Summe der beiden Serviceoperationskopplungsmaße:

$$SK(S_{235}) = GSOK(SO_{2351}) + GSOK(SO_{2352}) = 26 + 1 = 27$$

Die durchschnittliche Serviceoperationskopplung beträgt gemäß Formel (12) folglich:

$$DSOK(S_{235}) = \frac{SK(S_{235})}{|SO(S_{235})|} = \frac{27}{|\{SO_{2351}, SO_{2352}\}|} = \frac{27}{2} = 13{,}5$$

Die Auswertung der Metriken ergibt, dass die Serviceoperation SO_{2351} eine relativ hohe Kopplung und die Serviceoperation SO_{2352} eine sehr geringe Kopplung aufweisen. Aufgrund dieser unterschiedlich hohen Kopplungswerte ergibt sich für den Service S_{235} eine insgesamt mittlere durchschnittliche Kopplungsstärke. Eine vollständige Übersicht über die Werte der Kopplungsmetriken und deren Berechnung für den gesamten SOA-Entwurf des Fallbeispiels ist in Bild C-7 in Anhang C.4 gegeben.

6.1.3 Kohäsion

Theoretische Grundlagen der Kohäsion

Die Kohäsion eines Softwaremoduls wurde von Stevens, Myers und Constantine als gegenseitige Bindung der Modulbestandteile definiert.[247] Liegt eine hohe Kohäsion vor, so sind die Modulelemente eng miteinander verbunden (kohärent). Dies bewirkt i. d. R. eine

geringe Komplexität des Moduls sowie eine gute Verständlichkeit und Wartbarkeit.[248] Kopplung und Kohäsion stehen in enger Wechselwirkung und sollten deshalb nicht getrennt voneinander betrachtet werden.[249] Eine hohe Kohäsion ist meist mit einer geringen Kopplung des betrachteten Moduls zu seiner Umgebung verbunden. Wie auch die Kopplung wird die Kohäsion je nach Stärke in verschiedene Stufen eingeteilt. Diese Kohäsionsgrade sind in Tabelle 6-4 mit abnehmender Kohäsion aufgeführt.

Kohäsionsgrad	**Eigenschaften**
Funktionale Kohäsion (Functional Cohesion)	Alle Modulelemente wirken zur Erfüllung einer Funktion zusammen.
Sequentielle Kohäsion (Sequential Cohesion)	Die Ausgabedaten eines Elements sind Eingabedaten des nächsten Elements.
Kommunizierende Kohäsion (Communicational Cohesion)	Die Modulelemente benutzen die gleiche Menge von Ein- oder Ausgabedaten.
Prozedurale Kohäsion (Procedural Cohesion)	Alle Modulelemente sind an der Realisierung eines Prozesses beteiligt.
Zeitliche Kohäsion (Temporal Cohesion)	Die Modulelemente stehen in keiner Beziehung zueinander, werden aber im selben Zeitraum benutzt.
Logische Kohäsion (Logical Cohesion)	Die Modulelemente erfüllen verschiedene Funktionen, die nur logisch miteinander verbunden sind (gleiche Problemkategorie).
Zufällige Kohäsion (Coincidental Cohesion)	Die Elemente des Moduls sind untereinander nicht verbunden.

Tabelle 6-4: Unterscheidung verschiedener Kohäsionsgrade[250]

Die Definition von Kohäsionsmetriken für SOA-Entwürfe im nächsten Abschnitt beschränkt sich auf die Betrachtung der funktionalen Kohäsion. Als höchste Kohäsionsstufe stellt sie das erstrebenswerte Optimalziel beim Zuschnitt von Services dar. Es soll folglich untersucht werden, inwieweit die mit Axiomatic Design entworfenen Services die Anforderungen der funktionalen Kohäsion erfüllen und abgeschlossene, in

247 Vgl. Stevens, Myers, Constantine /Structured Design/ 121
248 Vgl. Darcy u. a. /Structural Complexity/ 986
249 Vgl. für diesen und den folgenden Satz Vogel u. a. /Software-Architektur/ 118
250 Vgl. für die einzelnen Kohäsionsgrade Page-Jones /Systemdesign/ 97, 98, 99, 102, 105, 107 f. und 110

sich kohärente Funktionen kapseln. Hierzu wird die Stärke des funktionalen Zusammenhalts der Operationen der Services bewertet.[251]

Die Kohäsion konnte im Softwarebereich lange Zeit nur durch subjektive Werturteile von Experten eingeschätzt werden.[252] So wird beispielsweise bei der lexikalischen Analyse versucht, die Funktion des betrachteten Moduls durch einen Satz zu beschreiben. Die Struktur dieses Satzes soll den Grad der Modulkohäsion reflektieren.[253] Page-Jones hat außerdem einen Entscheidungsbaum für die Ermittlung der Kohäsion entwickelt.[254]
Für die objektive Messung der Kohäsion existieren relativ wenige Metriken. Tabelle 6-5 gibt eine Übersicht über einige dieser Kohäsionsmetriken und die entsprechenden Literaturstellen.

Traditionelle Kohäsionsmetriken	
Cohesion Ratio	Fenton, Pfleeger /Software Metrics/ 313
Strong Functional Cohesion (SFC) Weak Functional Cohesion (WFC) Adhesiveness (A)	Bieman, Kang /Design-Level Cohesion/ 113 und Ott, Bieman /Program Slices/ 7
Spezielle Kohäsionsmetriken für objektorientierte Software	
Lack of Cohesion in Methods (LCOM)	Chidamber, Kemerer /Metrics Suite/ 488 f.
Tight Class Cohesion (TCC) Loose Class Cohesion (LCC)	Bieman, Kang /Cohesion and Reuse/ 261
Ratio of Cohesive Interactions (RCI)	Briand, Morasca, Basili /Measures for Design/ 729 f. und Briand, Daly, Wüst /Cohesion Measurement/ 80 f.

Tabelle 6-5: Bereits existierende Kohäsionsmetriken

Zur Bewertung objektorientierter Entwürfe wird überwiegend das Maß Lack of Cohesion in Methods (LCOM) aus der Metriksammlung von Chidamber und Kemerer eingesetzt. Es untersucht, in welcher Intensität die Methoden einer Klasse dieselben Instanzvariablen

251 Vgl. Papazoglou, Yang /Design Methodology/ 59
252 Vgl. Ott, Bieman /Program Slices/ 2
253 Vgl. Page-Jones /Systemdesign/ 112 f. und Stevens, Myers, Constantine /Structured Design/ 124
254 Vgl. Page-Jones /Systemdesign/ 114

verwenden.[255] Da Services keine Attributvariablen besitzen, ist diese Metrik nicht für die Bewertung einer SOA übertragbar.[256] Einige weitere, allgemein anwendbare Metriken basieren auf der Auswertung von Programm-Slices. Ein Slice ist eine, von irrelevanten Details befreite, Repräsentation eines Programmfragments.[257] Der Zusammenhalt der Programmfragmente bzw. Slices wird durch die Verwendung gemeinsamer Programmdaten hergestellt. Auf der Grundlage dieser Analysemethode werden im nächsten Abschnitt Kohäsionsmaße für SOA-Entwürfe entwickelt.

Definition von Kohäsionsmetriken für SOA-Entwürfe

Slice-basierte Metriken stellen einen möglichen Ansatz für eine objektive Kohäsionsmessung dar. Die ersten Metriken dieses Typs analysierten hierfür den Quellcode des zu beurteilenden Softwareartefakts. Zur Ableitung Slice-basierter Entwurfsmetriken muss sich diese Analyse nun auf Informationen beschränken, die während der Entwurfsphase vorliegen.[258] Eine Slice-basierte Entwurfsmetrik betrachtet deshalb nur die Schnittstellen der Softwarekomponenten[259] mit ihren Eingangs- und Ausgangsparametern. Die Schnittstellenparameter bilden so genannte *Data Token*. Für jeden Ausgangsparameter wird ein so genannter *Data Slice* ermittelt. Dieser enthält für den zugrunde liegenden Ausgangsparameter alle Data Token (Parameter), die seinen Wert beeinflussen.

Das Konzept der Data Slices geht von der Überlegung aus, dass die Aufgabe von Softwarekomponenten darin besteht, bestimmte Dienste bereitzustellen und dabei Daten zu generieren.[260] Zur Ausführung der angebotenen Funktionen verarbeiten die Softwarekomponenten ihre Eingangsparameter. Die Ergebnisse der Funktionsausführung werden über die Ausgangsparameter an die Umgebung bekannt gegeben. Somit repräsentiert jeder Ausgangsparameter bzw. sein korrespondierender Data Slice eine Funktionalität der betrachteten Komponente. Die höchste Kohäsionsstufe, die funktionale

255 Vgl. Chidamber, Kemerer /Metrics Suite/ 488
256 Vgl. Perepletchikov, Ryan, Frampton /Paradigms/ 438
257 Vgl. Meyers, Binkley /Slice-Based Cohesion/ 1
258 Vgl. für die folgenden Sätze Bieman, Kang /Design-Level Cohesion/ 113
259 In der zitierten Literatur wird bei der Darstellung der Kohäsion allgemein der Modulbegriff verwendet. Ein Modul ist in diesem Bericht i. d. R. mit einem Entwurfsmodul bzw. der korrespondierenden Serviceoperation assoziiert. Zur Erhaltung der begrifflichen Konsistenz wird deshalb im Folgenden statt von Modulen von Komponenten gesprochen. Dies soll verdeutlichen, dass Services (Komponenten) und nicht ihre Serviceoperationen (Module) Gegenstand der Kohäsionsanalyse sind.

Kohäsion, liegt vor, wenn die Funktionen der Komponente eng miteinander verbunden sind. Um feststellen zu können, wie stark die funktionale Kohäsion einer Komponente ist, müssen die Abhängigkeiten ihrer Parameter analysiert werden. Für jeden Ausgangsparameter werden alle anderen Eingangs- und Ausgangsparameter (Data Token) ermittelt, die zu ihm in Verbindung stehen. Sie bilden den Data Slice des betrachteten Ausgangsparameters (vgl. Bild 6-2). Eine Verbindung zwischen zwei Parametern besteht, wenn ein Parameter den Wert des anderen Parameters beeinflusst.[261]

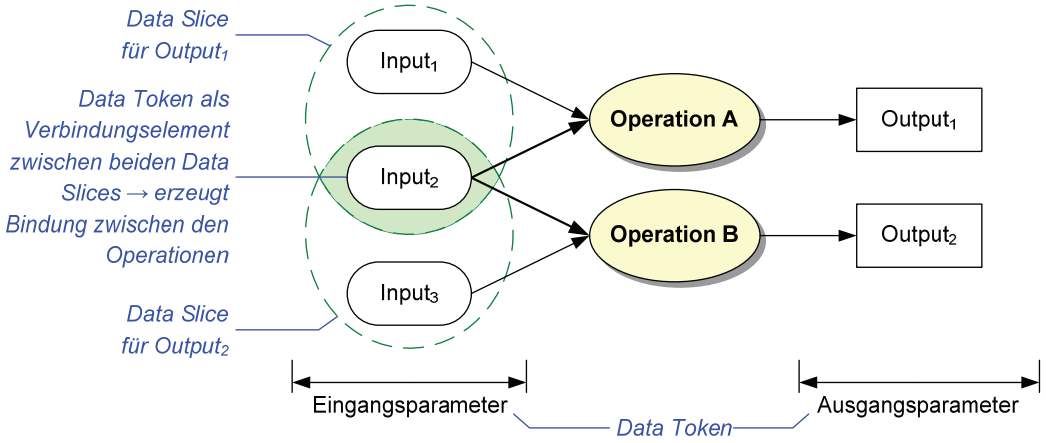


Bild 6-2: Identifizierung von Data Token und Data Slices

Ein Data Token kann mit mehreren Ausgangsparametern in Beziehung stehen und somit in mehreren Data Slices vorkommen.[262] Er wirkt dann als Verbindungselement zwischen diesen Data Slices (vgl. Data Token „$Input_2$“ in Bild 6-2). Die Stärke der Bindungswirkung eines Token hängt von dessen Verbundenheit (Connectedness) ab, die sich danach bemisst, in wie vielen Data Slices er auftritt. Je häufiger die Data Slices einer Komponente durch gemeinsame Data Token verbunden sind, desto stärker ist auch die Bindung der repräsentierten Funktionen und damit die funktionale Kohäsion der betrachteten Komponente.

Um die Kohäsion einer Softwarekomponente zu ermitteln, wird ein *Parameterabhängigkeitsgraph* (Input/Output Dependency Graph) aufgestellt.[263] Die Parameter bilden die Knoten des Graphen. Beeinflusst ein Parameter den Wert eines

260 Vgl. für diesen Absatz Ott, Bieman /Program Slices/ 2 f.
261 Vgl. Ott, Bieman /Program Slices/ 8
262 Vgl. für diesen Absatz Ott, Bieman /Program Slices/ 6 und Bieman, Kang /Design-Level Cohesion/ 115
263 Vgl. für diesen Absatz Bieman, Kang /Design-Level Cohesion/ 112 f.

anderen Ausgangsparameters, so sind die jeweiligen Knoten durch eine Kante verbunden. Dieser Graph kann auch in Form einer Tabelle wiedergegeben werden.
Aus dem Abhängigkeitsgraphen bzw. der Abhängigkeitstabelle der Softwarekomponente werden für ihre Parameter gewisse Eigenschaften ermittelt. Ein Parameter gilt als *isoliert*, wenn er nur zu einem Ausgangsparameter (Data Slice) eine Beziehung aufweist.[264] Besitzt ein Parameter hingegen Verbindungen zu allen Ausgangsparametern der Komponente, so ist er *essenziell*. Verfügt die Komponente nur über einen Ausgangsparameter, so sind alle Parameter essenziell bzw. nicht isoliert. Anhand dieser Parametereigenschaften lassen sich entsprechende Kohäsionsmaße berechnen.

Das soeben dargestellte, Slice-basierte Konzept zur Kohäsionsbewertung lässt sich mit geringfügigen Anpassungen auf SOA-Entwürfe anwenden. Im Fokus steht im Folgenden die Messung der Kohäsion von Services. Ein Service entspricht einer zu bewertenden Softwarekomponente. Seine Serviceoperationen und deren Schnittstellendefinitionen bilden die Komponentenelemente. Die Schnittstellenparameter der Serviceoperationen entsprechen den Data Token und bilden die Grundlage für die folgende Kohäsionsanalyse. Die Eingangsparameter einer Serviceoperation werden durch den entsprechenden Eingangsteil DP_{in} des zugehörigen Designparameters aus der Entwurfsmatrix repräsentiert. Außerdem müssen auch die Nichtdiagonalelemente der Operation in Form von Daten- oder Steuerungsabhängigkeiten zu den Eingangsparametern hinzugezählt werden, da sie in gewisser Weise Eingangsdaten darstellen. Die Ausgangsparameter einer Serviceoperation werden durch den entsprechenden Ausgangsteil DP_{out} des zugehörigen Designparameters gebildet. Zur Bewertung eines Service werden die Eingangsdaten seiner Operationen identifiziert und ihre Beziehungen zu den Ausgangsdaten mithilfe des Abhängigkeitsgraphen analysiert. Aus diesem Graphen bzw. einer entsprechenden Tabelle lassen sich folgende Werte ermitteln[265]:

T_h Anzahl aller Eingangs-/Ausgangsparameter (DP_{in}, Daten- und Steuerungsabhängigkeiten/DP_{out}) des Service h

O_h Anzahl aller Ausgangsparameter (DP_{out}) des Service h

264 Vgl. für die folgenden Sätze Bieman, Kang /Design-Level Cohesion/ 113
265 Vgl. für die folgenden Variablen und Formeln Ott, Bieman /Program Slices/ 8 f. und Bieman, Kang /Design-Level Cohesion/ 115

N_h Anzahl nicht isolierter Parameter des Service *h*

E_h Anzahl essenzieller Parameter des Service *h*

V_i Anzahl von Ausgangsparametern, zu denen Parameter *i* eine Verbindung besitzt

Aus diesen Werten lassen sich schließlich die nachfolgenden Kohäsionsmaße berechnen. Die *schwache Kohäsion* (Loose Cohesion, *LC* bzw. Weak Functional Cohesion) beschreibt die relative Zahl nicht isolierter Parameter eines Service:[266]

$$LC(S_h) = \frac{N_h}{T_h} \tag{14}$$

Die *starke Kohäsion* (Tight Cohesion, *TC* bzw. Strong Functional Cohesion) ermittelt die relative Anzahl essenzieller Parameter eines Service:[267]

$$TC(S_h) = \frac{E_h}{T_h} \tag{15}$$

Die Verteilung der isolierten und essenziellen Parameter beeinflusst, wie stark die Ausgangsparameter eines Service und damit seine verschiedenen Funktionen miteinander verbunden sind.[268] Eine solche Verbundenheit der Funktionen wird durch die Zahl der Ausgangsparameter beeinflusst, mit der jeder Parameter durchschnittlich in Beziehung steht.[269] Die Verbundenheit der Serviceoperationen eines Service wird durch die *Servicekohäsion* (Service Cohesion, *SC* bzw. Adhesiveness) repräsentiert:

$$SC(S_h) = \begin{cases} \dfrac{\sum_{i=1}^{T_h}(V_i - 1)}{T_h \cdot (O_h - 1)} & \text{für } O_h > 1 \\ 1 & \text{sonst} \end{cases} \tag{16}$$

Nicht isolierte Parameter besitzen eine schwächere Kohäsionswirkung als essenzielle Parameter.[270] Deshalb weist ein hoher Wert von *LC* nur auf eine gute funktionale Kohäsion hin. Besitzt hingegen das Maß *TC* einen hohen Wert, zeugt dies von einer stark

266 Vgl. Ott, Bieman /Program Slices/ 9 und Bieman, Kang /Design-Level Cohesion/ 115
267 Vgl. Ott, Bieman /Program Slices/ 9 und Bieman, Kang /Design-Level Cohesion/ 115
268 Vgl. Ott, Bieman /Program Slices/ 6 f.
269 Vgl. für diesen Satz und Formel (16) Bieman, Kang /Design-Level Cohesion/ 115
270 Vgl. Ott, Bieman /Program Slices/ 7

ausgeprägten funktionalen Kohäsion innerhalb des betrachteten Service. Der Wert *SC* verbindet beide Maße zu einer Gesamtbeurteilung der Kohäsion. Fällt er hoch aus, spricht dies für eine insgesamt gute Servicekohäsion.
Für den gesamten SOA-Entwurf kann schließlich die *durchschnittliche Servicekohäsion (DSC)* berechnet werden:

$$DSC = \frac{\sum_{S_h \in S} SC(S_h)}{|S|} \tag{17}$$

Ebenso wie die durchschnittliche Servicekopplung (*DSK*) dient auch die Metrik *DSC* der Gesamtbewertung eines Entwurfs und dem Vergleich verschiedener Entwurfsalternativen.

Anwendung der entwickelten Kohäsionsmetriken

Die Aufstellung des Parameterabhängigkeitsgraphen, die Ermittlung der Parametereigenschaften und die Berechnung der Kohäsionsmaße soll nun anhand eines Service, des in Abschnitt 5.2 entwickelten Fallbeispiels, demonstriert werden. Zur Vereinfachung der Analyse bietet sich auch hier die Verwendung der mit Axiomatic Design erstellten vollständigen Entwurfsmatrix an. Im Weiteren wird der Service „Ermittle Lieferant“ (S_{31}) betrachtet. Er besitzt fünf Serviceoperationen, von denen jedoch nur die zwei Operationen SO_{313} und SO_{314} für die Ermittlung der Servicekohäsion relevant sind. Die anderen Operationen (SO_{311}, SO_{312}, SO_{315}) stellen primär keine eigene Funktionalität zur Verfügung, da sie lediglich untergeordnete Services komponieren. Die Operation „wähle Lieferant aus()“ (SO_{313}) verfügt über keine direkten Eingangsparameter (DP_{in}), aber über drei Nichtdiagonalelemente. Da es sich um Datenabhängigkeiten handelt, werden die Ausgangsparameter (DP_{out}) der durch die Kopplungselemente referenzierten Serviceoperationen zu Eingangsdaten der betrachteten Operation SO_{313}. Somit verarbeitet die Operation SO_{313} die Auftragspositionen (Nichtdiagonalelement $A_{313,12}$), die Bereitstellungskosten ($A_{313,3112}$) und das oder die Lieferantenangebote ($A_{313,3122}$). Die Operation wählt anhand dieser Daten den günstigsten Lieferanten aus und gibt diesen als Ausgangsparameter (DP_{out}) zurück.
Die zweite Serviceoperation „erteile Lieferauftrag()“ (SO_{314}) besitzt die Lieferauftragsdaten als direkten Eingangsparameter. Weiterhin benötigt diese Operation über Datenabhängigkeiten folgende Daten: die Auftragsnummer ($A_{314,111}$), die Auftragspositionen ($A_{314,12}$), den aktuellen Lagerbestand ($A_{314,3111}$), das Lieferantenangebot ($A_{314,3122}$) sowie den Ausgangsparameter der Operation „wähle Lieferant aus()“, den

gewählten Lieferanten ($A_{314,313}$). Die betrachtete Operation generiert mit diesen Daten einen Lieferauftrag und gibt dessen Nummer sowie den Lieferstatus als Ausgangsparameter zurück. Auf der Basis der Auflistung dieser Daten kann der Abhängigkeitsgraph für den Service „Ermittle Lieferant“ bzw. dessen Operationen aufgestellt werden. Er ist in Bild 6-3 angegeben. Eingangsparameter sind durch abgerundete und Ausgangsparameter durch normale Rechtecke dargestellt.

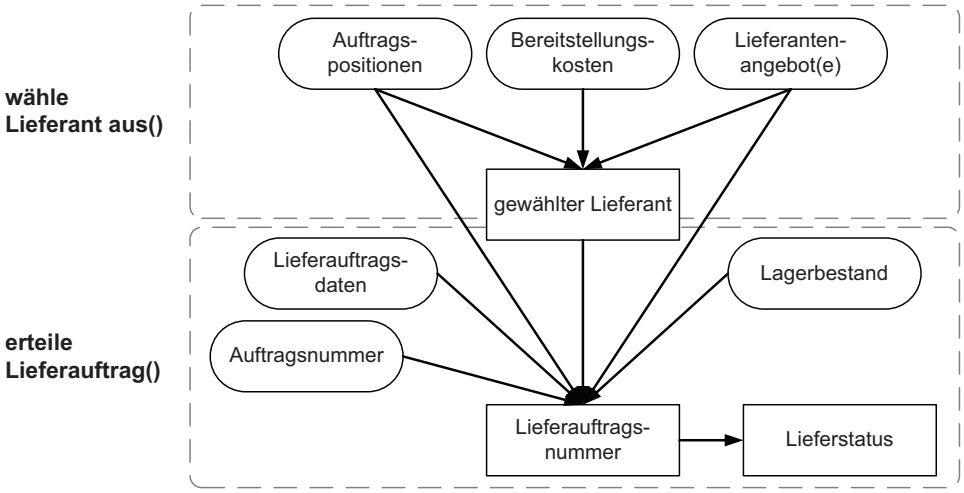


Bild 6-3: Abhängigkeitsgraph des Service „Ermittle Lieferant“

Anhand dieses Abhängigkeitsgraphen kann die Tabelle 6-6 erstellt werden. Sie dokumentiert die Beziehungen zwischen Eingangs- und Ausgangsparametern.

	Eingangs-/Ausgangsparameter (DP_{in}, DP_{out}, Daten- und Steuerungsabhängigkeiten)	**Ausgangsparameter** (DP_{out})			**Anzahl verbundener Ausgangsparameter (V_i)**	**nicht isolierter Parameter?**	**essenzieller Parameter?**
		1	2	3			
		gewählter Lieferant	Lieferauftragsnummer	Lieferstatus			
1	Auftragspositionen	1	1	1	3	X	X
2	Bereitstellungskosten	1	1	1	3	X	X
3	Lieferantenangebot(e)	1	1	1	3	X	X
4	gewählter Lieferant	1	1	1	3	X	X
5	Lieferauftragsdaten	*0*	1	1	*2*	X	*nein*
6	Auftragsnummer	*0*	1	1	*2*	X	*nein*
7	Lagerbestand	*0*	1	1	*2*	X	*nein*
8	Lieferauftragsnummer	1	1	1	3	X	X
9	Lieferstatus	1	1	1	3	X	X

Tabelle 6-6: Abhängigkeitstabelle des Service „Ermittle Lieferant“

Der Wert 1 im Inhaltsbereich der Tabelle weist auf eine direkte oder indirekte Beziehung zwischen den entsprechenden Parametern im Abhängigkeitsgraphen hin. Aus den Daten der Abhängigkeitstabelle des Service geht hervor, dass kein Parameter isoliert ist und sechs

Parameter essenziell sind. Aus diesen Angaben lassen sich die Kohäsionsmaße gemäß den Formeln (14), (15) und (16) berechnen:

$$T_{31} = 9, \quad O_{31} = 3, \quad N_{31} = 9, \quad E_{31} = 6$$

$$LC(S_{31}) = \frac{N_{31}}{T_{31}} = \frac{9}{9} = 1$$

$$TC(S_{31}) = \frac{E_{31}}{T_{31}} = \frac{6}{9} \approx 0{,}67$$

$$SC(S_{31}) = \frac{\sum_{i=1}^{T}(V_i - 1)}{T_{31} \cdot (O_{31} - 1)} = \frac{\sum_{i=1}^{9}(V_i - 1)}{9 \cdot (3 - 1)} = \frac{15}{18} \approx 0{,}83$$

Hinsichtlich der schwachen funktionalen Kohäsion (*LC*) hat der Service „Ermittle Lieferant“ den Maximalwert von 1 erreicht. Man kann deshalb prinzipiell von einer befriedigenden Kohäsion ausgehen. Weiterhin weist das Ergebnis von 0,67 für die starke Kohäsion (*TC*) auf eine recht gute Bindung innerhalb des Service hin. Der Wert der Servicekohäsion (*SC*) von 0,83 liegt nahe am Höchstwert von 1. Somit ergibt sich für den Service zusammenfassend eine gute Gesamtbewertung der Kohäsion. Da sehr viele Parameter des betrachteten Service zur Erzeugung seiner Ausgangsparameter beitragen, können die Bestandteile des Service als zusammengehörig bzw. kohärent eingeschätzt werden.

Eine vollständige Übersicht über die Werte der Kohäsionsmetriken für den gesamten SOA-Entwurf des Fallbeispiels ist in Bild C-7 in Anhang C.4 gegeben.

6.1.4 Servicegranularität

Theoretische Grundlagen der Servicegranularität

Der Begriff der Granularität beschreibt im allgemeinen Sprachgebrauch die Körnigkeit[271] von Stoffen. Die Definition der Granularität von Software ist wissenschaftlich noch wenig durchdrungen. Im Umfeld serviceorientierter Architekturen beschreibt die *Servicegranularität* den Umfang der Funktionalität, die ein Service bzw. seine Operationen kapselt.[272] Im Rahmen des SOA-Entwurfs mit Axiomatic Design wird eine fachlich-

[271] Vgl. Wermke, Kunkel-Razum, Scholze-Stubenrecht /Fremdwörterbuch/ 378

[272] Vgl. Feuerlicht /Service Granularity/ 315 und Whitehead /Component-Based Development/ 22

funktionale Dekomposition als Abstraktionskriterium verwendet. Diese Modularisierung führt zu einer Verteilung der Gesamtfunktionalität auf einzelne Services.[273] Die Granularität der entstehenden Services ist abhängig vom Abstraktionsniveau der von ihnen realisierten Aufgaben (vgl. Bild 6-4).[274] Eine grobe Servicegranularität entspricht einem hohen Abstraktionsniveau und einem großen Funktionsumfang. Werden die fachlichen Aufgaben hingegen stärker zergliedert, führt dies zu einer Konkretisierung[275] der Funktionalität und zu einem niedrigeren Abstraktionsniveau. Es entstehen feingranulare Services mit einem kleinen, abgegrenzten Aufgabenbereich.

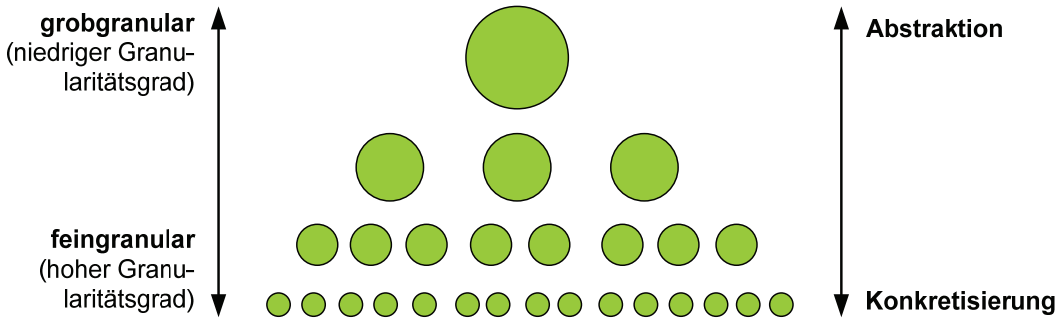


Bild 6-4: Spektrum der Granularität

Grobgranulare Services stellen umfangreiche Funktionalität in Form ihrer Serviceoperationen zur Verfügung. Die Nutzung grobgranularer Services erfordert nur wenige Interaktionen mit dem Dienstanbieter. Dies reduziert den notwendigen Nachrichtenaustausch über Netzwerke und führt zu einer guten Performance.[276] Auch das Wiederverwendungspotenzial ist aufgrund des breiten Einsatzspektrums i. d. R. höher als bei den meist spezialisierten feingranularen Services. Der große funktionale Umfang grobgranularer Services führt häufig zu einer geringen Servicekopplung.[277] Der hohe Abstraktionsgrad birgt jedoch auch die Gefahr einer schwachen Kohäsion der Operationen des Service.[278] Die Serviceschnittstellen sind aufgrund des Funktionsumfangs meist recht komplex.[279] Die generalisierten Serviceoperationen[280] sind oft wenig flexibel. Sie erfordern,

273 Vgl. Friedrich u. a. /Operating Systems/ 56
274 Vgl. Fiege, Stelzer /Analyse/ 4 und Balzert /Software-Qualitätssicherung/ 559
275 Vgl. Balzert /Software-Qualitätssicherung/ 559
276 Vgl. Feuerlicht /Service Granularity/ 316
277 Vgl. Huhns, Singh /Service-Oriented Computing/ 79
278 Vgl. De Jonge /Build-Level Components/ 589 und Feuerlicht /Service Granularity/ 319
279 Vgl. Hopkins /Component Primer/ 29
280 Vgl. Marks, Bell /Planning and Implementation/ 124

dass, auch wenn nur eine Teilfunktion benötigt wird, trotzdem die gesamte Operation ausgeführt werden muss und ggf. überflüssige Daten transferiert werden.[281] Außerdem kann die grobe Granularität Sicherheitsprobleme bereiten, da Zugriffsbeschränkungen meist nur auf der Ebene der Services und nicht für einzelne Operationen festgelegt werden können.[282]

Feingranulare Services verfügen i. d. R. über eine einfache Schnittstelle und sind flexibel in Servicekompositionen einsetzbar.[283] Die Realisierung eines vollständigen fachlichen Teilprozesses erfordert, im Vergleich zur Nutzung grobgranularer Services, die Kombination vieler feingranularer Services. Dies verursacht einen hohen Aufwand für Servicekomposition und -kommunikation.[284] Folglich kann es zu Performanceproblemen kommen, da für die vielen Serviceaufrufe entsprechende Anfrage- und Antwortnachrichten versendet sowie ggf. Verschlüsselung und Authentifizierung durchgeführt werden müssen. Die große Anzahl feingranularer Services führt nicht zuletzt auch zu Schwierigkeiten hinsichtlich effizienter Verwaltung und Monitoring der SOA.[285]

Die Granularität ist ein relatives Merkmal eines Service bzw. einer Serviceoperation und kann deshalb nur im Hinblick auf den konkret vorliegenden Entwurfskontext bestimmt werden.[286] Der Entwurfskontext ergibt sich aus dem Entwurfsgegenstand und -umfeld sowie den Entwurfszielen, insbesondere den Anforderungen bezüglich Wiederverwendbarkeit, Performance, Flexibilität usw. Für die Wahl eines angemessenen Granularitätsgrades gibt es dementsprechend keine allgemein gültigen Regeln oder Handlungsanweisungen.[287] In der Praxis ist stets eine detaillierte Analyse unter Berücksichtigung des Entwurfskontexts erforderlich.

Es gibt allerdings gewisse Empfehlungen hinsichtlich der Wahl einer geeigneten Servicegranularität.[288] So sollten die Kernfunktionen eines Entwurfsbereichs eher durch feingranulare Services realisiert werden. Dies entspricht der Spezialisierung und den hohen Flexibilitätsanforderungen dieser Funktionen. Hingegen können standardisierte und

---

281 Vgl. Erl /Service-Oriented Architecture/ 557 und Feuerlicht /Service Granularity/ 319
282 Vgl. McGovern u. a. /Java Web Services/ 54
283 Vgl. Feuerlicht /Service Granularity/ 316
284 Vgl. De Jonge /Build-Level Components/ 589 und Feuerlicht /Service Granularity/ 317
285 Vgl. Szyperski /Component Software/ 150
286 Vgl. Krafzig, Banke, Slama /Enterprise SOA/ 205
287 Vgl. für diesen und den folgenden Satz Szyperski /Component Software/ 139 und 150
288 Vgl. für die folgenden Sätze Reldin, Sundling /Explaining Service Granularity/ 109

universell verwendete Kontextfunktionen meist durch grobgranulare Services abgebildet werden.[289] Um die Vorteile grober und feiner Granularität zu kombinieren, können *multigranulare Services* entwickelt werden.[290] Die gewünschte Funktionalität wird hierbei zunächst durch viele, eher feingranulare Services umgesetzt. Anschließend werden gemäß dem Fassaden-Entwurfsmuster[291] grobgranulare Services entwickelt, die die Dienste der feingranularen Services komponieren und somit kapseln. Dadurch kann für die allgemeine, insbesondere externe Nutzung der Funktionalität eine grobgranulare und stabile Schnittstelle zur Verfügung gestellt werden. Für spezielle Anforderungen ist parallel der Zugriff auf die feingranularen Services möglich. Nachteil dieser Kompromisslösung ist der wesentlich höhere Aufwand für Management und Wartung der SOA.

Neben der Begriffsbestimmung bereitet auch die Ermittlung oder gar Messung der Granularität von Software in der Praxis erhebliche Schwierigkeiten. In den nachfolgenden Abschnitten soll dennoch der Versuch unternommen werden, eine Annäherung an diese Problematik zu erreichen. Zunächst werden hierzu Metriken für die objektive Messung der Granularität vorgeschlagen. Resultierend aus deren Schwachstellen soll anschließend ein qualitatives Bewertungskonzept entwickelt werden.

Definition von Granularitätsmetriken für SOA-Entwürfe

Der nachfolgend entwickelte Ansatz für die Messung der Servicegranularität basiert auf der These, dass sich der Funktionsumfang eines Service in der Komplexität der von ihm verwendeten Daten widerspiegelt.[292] Folglich sollte die strukturelle Komplexität der genutzten Daten[293] mit dem Abstraktionsgrad der gekapselten Funktionalität und somit mit der Servicegranularität korrespondieren. Verarbeitet und erzeugt ein Service komplexe und stark strukturierte Daten bzw. Geschäftsobjekte aus unterschiedlichen fachlichen Domänen, so weist dies auf einen großen Funktionsumfang und damit eine grobe Granularität hin.[294] Hingegen spricht die Verwendung einfacher und spezieller Parameter eher für einen kleinen Funktionsumfang und folglich eine feine Servicegranularität.

289 Vgl. Marks, Bell /Planning and Implementation/ 125
290 Vgl. für die folgenden Sätze dieses Absatzes McGovern u. a. /Java Web Services/ 53-55
291 Vgl. Gamma u. a. /Entwurfsmuster/ 212-222
292 Vgl. Feuerlicht /Service Granularity/ 317
293 Vgl. Jones /Software Measurement/ 322 (Data Complexity, Structural Complexity)
294 Vgl. für diesen und den folgenden Satz McGovern u. a. /Java Web Services/ 53

Zur Berechnung der Servicegranularität müssen, gemäß der erläuterten These, zunächst alle Daten bestimmt werden, die von den Operationen des betrachteten Service verarbeitet werden. Für jede Serviceoperation SO_i ergibt sich somit eine Menge $D(SO_i)$ von Daten D_t, die von dieser Operation entweder verarbeitet, geändert oder bereitgestellt werden:

$$D(SO_i) = \{D_t \mid SO_i \text{ verwendet } D_t\} \tag{18}$$

Wurde der Entwurf der SOA mit Axiomatic Design durchgeführt, so können die von einer Serviceoperation verarbeiteten Daten aus der Entwurfsmatrix ermittelt werden. Sie ergeben sich aus dem Eingangsteil des Designparameters (DP_{in}), der die Operationsschnittstelle beschreibt sowie aus den Daten- und Steuerungsabhängigkeiten der betrachteten Operation. Die von der Serviceoperation erzeugten oder geänderten Daten entsprechen dem Ausgangsteil des zugehörigen Designparameters (DP_{out}). Da die Designparameter im Rahmen des Grobentwurfs formuliert werden, ist ihr Aufbau nur oberflächlich beschrieben. Für eine genauere Analyse muss ihre Datenstruktur durch eine manuelle Ausarbeitung oder die Verwendung bereits existierender Datenmodelle verfeinert werden. Ausgehend von der zugrunde liegenden funktionalen Anforderung sollten alle von einer Serviceoperation verarbeiteten oder erzeugten Daten und deren Struktur bis auf die Ebene der Datenelemente ermittelt werden.

Die Bestimmung der Komplexität der von einem Service genutzten Daten basiert auf einer eingehenden Analyse ihrer Struktur und Elemente. Zur quantitativen Beurteilung der strukturellen Komplexität eines Datums kann auf die Bewertungsvorschriften der Function Point-Methode zurückgegriffen werden.[295] Das Function Point-Verfahren dient der Aufwandsschätzung von IT-Projekten anhand des Funktionsumfangs einer Anwendung aus Nutzersicht.[296] Zur Bemessung des Funktionsumfangs werden neben den erforderlichen Transaktionen (Transaktions-Function Points) auch die von der Anwendung verwendeten Daten (Daten-Function Points) hinsichtlich ihrer Komplexität bewertet.[297] Die Komplexität der Daten wird durch zwei Faktoren bestimmt: die Anzahl von Datenelementen und die Anzahl von Elementgruppen.[298] Ein *Datenelement* (Data Element Type, DET) ist ein nicht

295 Vgl. Whitehead /Component-Based Development/ 22
296 Vgl. Bundschuh, Fabry /Aufwandschätzung/ 179
297 Vgl. Bundschuh, Fabry /Aufwandschätzung/ 237
298 Vgl. Bundschuh, Fabry /Aufwandschätzung/ 241. In Cardoso /Data-Flow Complexity/ 69-71 wird ein ähnliches Konzept für die Bewertung der Datenkomplexität vorgeschlagen. Dazu werden auf der Feinentwurfsebene die

rekursives Datenfeld.[299] Eine *Elementgruppe* (Record Element Type, RET) ist eine Untergruppe, die wiederum aus Datenelementen besteht. Die Anzahl der Datenelemente charakterisiert die Breite und die Zahl der Elementgruppen die Strukturierung eines Datums. Durch die Analyse der Struktur eines Datums D_t und das Auszählen seiner Datenelemente und Elementgruppen erhält man folgende zwei Werte:

$DET(D_t)$ Anzahl atomarer Datenelemente im Datum D_t

$RET(D_t)$ Anzahl Elementgruppen im Datum D_t [300]

Anhand dieser Werte wird die strukturelle Datenkomplexität $DK(D_t)$ des Datums D_t ermittelt. Hierfür werden die Zählregeln der Function Point-Methode der International Function Point Users Group (IFPUG) herangezogen. Diese Regeln sind in Tabelle 6-7 dargestellt. Sie unterscheiden zwischen externen Daten, die von der betrachteten Serviceoperation zwar referenziert, aber extern verwaltet werden und internen Daten, die von der Serviceoperation selbst gepflegt werden, also in ihrem Autonomiebereich liegen.[301] Die Function Point-Zählregeln legen Grenzen für die Werte von $DET(D_t)$ und $RET(D_t)$ fest, nach denen bestimmte Punktwerte (ungewichtete Function Points) für die Komplexität DK des Datums D_t vergeben werden. Der erste Punktwert in einer Tabellenzelle gilt für externe Daten und der zweite Wert für in der Operation gepflegte Daten.

Datenkomplexität $DK(D_t)$		**Anzahl Datenelemente,** $DET(D_t)$								
		1 – 19			**20 – 50**			**> 50**		
		extern	/	intern	extern	/	intern	extern	/	intern
Anzahl Elementgruppen, $RET(D_t)$	**1**	5	/	7	5	/	7	7	/	10
	2 – 5	5	/	7	7	/	10	10	/	15
	> 5	7	/	10	10	/	15	10	/	15

Tabelle 6-7: Datenkomplexität in Function Points nach IFPUG, Version 4.1[302]

Anhand von Tabelle 6-7 wird für jedes Datum D_t, das von der betrachteten Serviceoperation SO_i verwendet wird, ein Komplexitätswert $DK(D_t)$ ermittelt. Die

Datenstrukturen in Form von XML-Schemata untersucht und zwischen primitiven und komplexen Datentypen unterschieden.

299 Vgl. für die folgenden Sätze Bundschuh, Fabry /Aufwandschätzung/ 241-243

300 Wenn eine Datenstruktur keine Untergruppen enthält, wird der Wert RET trotzdem mit 1 belegt (vgl. Bundschuh, Fabry /Aufwandschätzung/ 241).

301 Vgl. Bundschuh, Fabry /Aufwandschätzung/ 242 (Internal Logical File und External Interface File)

302 Quelle: Bundschuh, Fabry /Aufwandschätzung/ 243

Serviceoperationsgranularität (SOG) berechnet sich aus der Summe dieser Datenkomplexitäten:

$$SOG(SO_i) = \sum_{D_t \in D(SO_i)} DK(D_t) \tag{19}$$

Die *Servicegranularität (SG)* eines Service S_h ergibt sich durch Aufsummierung der Granularitätswerte aller seiner Operationen:

$$SG(S_h) = \sum_{SO_i \in SO(S_h)} SOG(SO_i) \tag{20}$$

Die *durchschnittliche Serviceoperationsgranularität (DSOG)* repräsentiert die durchschnittliche Granularität der Operationen des betrachteten Service. Sie setzt die Servicegranularität ins Verhältnis zur Anzahl der Serviceoperationen:

$$DSOG(S_h) = \frac{SG(S_h)}{|\,SO(S_h)\,|} = \frac{\sum_{SO_i \in SO(S_h)} SOG(SO_i)}{|\,SO(S_h)\,|} \tag{21}$$

Für den zu untersuchenden SOA-Entwurf kann auch die *durchschnittliche Servicegranularität (DSG)* ermittelt werden. Diese Metrik bestimmt den Mittelwert der Servicegranularität (*SG*) in Bezug auf alle Services des Entwurfs:

$$DSG = \frac{\sum_{S_h \in S} SG(S_h)}{|\,S\,|} \tag{22}$$

Alle dargestellten Metriken bemessen den Funktionsumfang und somit die Granularität einer Serviceoperation oder eines Service anhand der Komplexität der verwendeten Daten. Je nach Datenkomplexität ergeben sich somit unterschiedlich hohe Metrikwerte. Je höher die Metrikwerte sind, desto gröber ist die Granularität der betrachteten Serviceoperationen bzw. Services. Dieser Zusammenhang ist für die Metriken *SOG* und *SG* in Bild 6-5 verdeutlicht. Die dargestellte Funktion kann, wie beim Function Point-Verfahren, auf der Grundlage von Erfahrungswerten oder Vergangenheitsdaten ähnlicher Projekte ermittelt und fortgeschrieben werden.[303]

[303] Vgl. Balzert /Software-Entwicklung/ 85

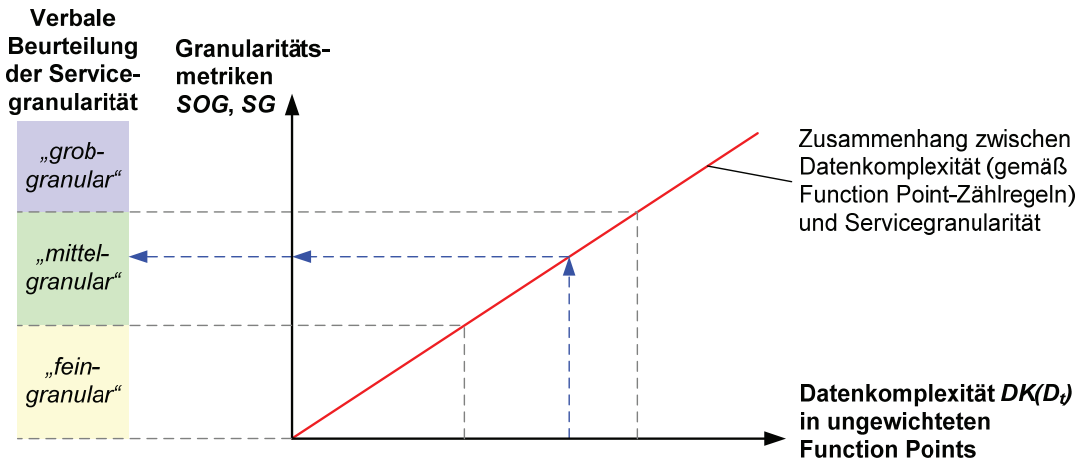


Bild 6-5: Zusammenhang zwischen Datenkomplexität und Granularität

Den mithilfe der Datenkomplexität berechneten Metrikwerten kann eine verbale Beurteilung des Granularitätsgrades zugeordnet werden. Hierzu wird in Bild 6-5 zwischen feiner, mittlerer und grober Granularität unterschieden.

In Abschnitt 0 wurde betont, dass die Granularität ein relatives Merkmal von Services bzw. Serviceoperationen ist. Folglich muss die verbale Einschätzung der Granularität stets im Hinblick auf den zugrunde liegenden Entwurfskontext erfolgen. Die genaue Festlegung der Grenzen zwischen den verschiedenen Granularitätsgraden ist deshalb vom Entwurfskontext und den Entwurfszielen abhängig. Zur Anpassung des Messkonzepts an die konkret vorliegenden Entwurfsbedingungen können in Tabelle 6-7 die Klassengrenzen für die Größen *DET* und *RET* sowie die vergebenen Punktwerte für die Datenkomplexität variiert werden.[304] Weiterhin kann der funktionale Zusammenhang zwischen Datenkomplexität und Granularität verändert werden. Beispielsweise kann in Bild 6-5 auch ein nichtlinearer Funktionsverlauf angenommen werden, der sich dann entsprechend in Änderungen an Formel (19) niederschlägt. Außerdem können die Anzahl der verbal formulierten Granularitätsgrade angepasst und die Position der Grenzen zwischen den verschiedenen Graden individuell verschoben werden.

Anwendung der entwickelten Granularitätsmetriken

Die Berechnung der im vorangegangenen Abschnitt definierten Metriken soll auszugsweise am Fallbeispiel aus Kapitel 5 veranschaulicht werden. Hierzu wurde der

[304] Vgl. Balzert /Software-Entwicklung/ 87 f.

Service „Prüfe eigene Liefermöglichkeit“ (S_{311}) ausgewählt. Er besitzt zwei Serviceoperationen, die nachfolgend hinsichtlich der Komplexität der verwendeten Daten untersucht werden.

Die Operation „ermittle Lagerbestand()“ (SO_{3111}) referenziert die Daten der gewünschten Auftragsposition über eine Datenabhängigkeit und ist aufgrund einer Steuerungsabhängigkeit an das Ergebnis der Auftragsprüfung gebunden. Ist die Suche nach Beständen zur Auftragsposition in den verfügbaren Lagern abgeschlossen, werden die ermittelten Lagerbestände als Ausgangsparameter ($DP_{3111,out}$) zurückgegeben. Die genannten Daten wurden in Bild 6-6 beispielhaft bis auf die Ebene der Datenelemente verfeinert.

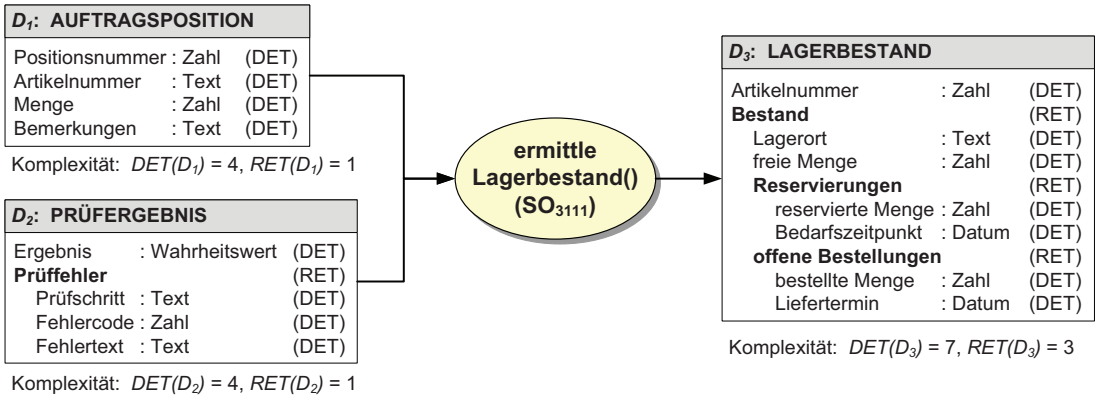


Bild 6-6: Von Serviceoperation SO_{3111} verwendete Daten und deren Struktur

Die von der Serviceoperation SO_{3111} verwendeten Daten müssen hinsichtlich ihrer strukturellen Komplexität untersucht werden. Diese Analyse ergibt, dass das Datum „Auftragsposition“ (D_1) aus vier Datenelementen besteht ($DET(D_1) = 4$) und keine Untergruppe aufweist ($RET(D_1) = 1$). Die „Auftragsposition“ ist ein externes Datum, das von der Serviceoperation „verarbeite Auftragspositionen()“ verwaltet wird. Laut Tabelle 6-7 ergibt sich somit für dieses Datum ein Komplexitätswert von 5 ($DK(D_1) = 5$). Das externe Datum „Prüfergebnis“ (D_2) wird ebenfalls mit einem Punktwert von 5 bewertet ($DK(D_2) = 5$). Der Ausgangsparameter „Lagerbestand“ (D_3) ist stärker strukturiert. Er verfügt über drei Untergruppen und sieben Datenelemente. Es handelt sich zudem um ein intern gepflegtes Datum, so dass ihm laut Tabelle 6-7 ein Punktwert von 7 zugeordnet wird ($DK(D_3) = 7$).

Die zweite Operation des betrachteten Service ist „berechne Bereitstellungskosten()“ (SO_{3112}). Sie verarbeitet aufgrund von Datenabhängigkeiten die Daten der Auftragsposition und der von Operation SO_{3111} ermittelten Lagerbestände. Für alle übergebenen

Lagerbestände werden die Bereitstellungskosten berechnet und als Ausgangsparameter zurückgegeben. Diese Zusammenhänge sind in Bild 6-7 dargestellt.

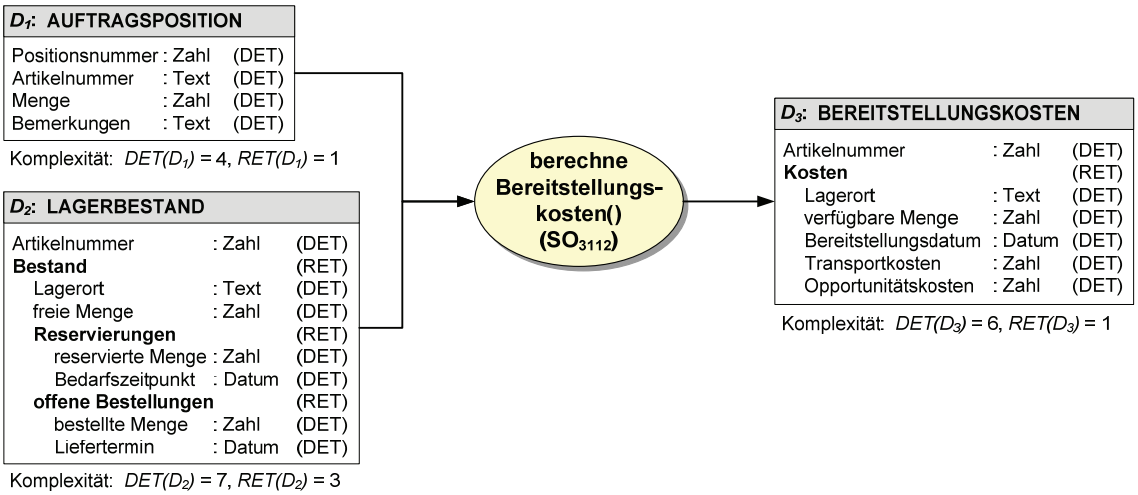


Bild 6-7: Von Serviceoperation SO_{3112} verwendete Daten und deren Struktur

Für die verarbeiteten Daten „Auftragsposition“ und „Lagerbestand“ wurde die Datenstruktur bereits in Bild 6-6 analysiert. Dem Datum „Auftragsposition“ wird erneut ein Komplexitätswert von 5 zugeordnet ($DK(D_1) = 5$). Der „Lagerbestand“ ist für die Operation „berechne Bereitstellungskosten()“ ein externes Datum, das von Operation SO_{3111} bereitgestellt wird. Folglich ergibt sich ein Punktwert von 5 für die Datenkomplexität ($DK(D_2) = 5$). Der Ausgangsparameter „Bereitstellungskosten“ besteht aus sechs Datenelementen ($DET(D_3) = 6$) und einer Untergruppe ($RET(D_3) = 1$). Da es sich um ein internes Datum handelt, wird ihm gemäß Tabelle 6-7 ein Komplexitätswert von 7 zugeordnet ($DK(D_3) = 7$).

Für jede der zwei Serviceoperationen wurden die von ihnen verwendeten Daten ermittelt und hinsichtlich ihrer Komplexität bewertet. Auf der Grundlage dieser Ergebnisse kann die Granularität der Operationen gemäß Formel (19) berechnet werden:

$$\begin{aligned} SOG(SO_{3111}) &= DK(\text{"Auftragsposition"}) + DK(\text{"Prüfergebnis"}) + DK(\text{"Lagerbestand"}) \\ &= 5 + 5 + 7 = 17 \\ SOG(SO_{3112}) &= DK(\text{"Auftragsposition"}) + DK(\text{"Lagerbestand"}) + DK(\text{"Bereitstellungskosten"}) \\ &= 5 + 5 + 7 = 17 \end{aligned}$$

Die Granularität des betrachteten Service „Prüfe eigene Liefermöglichkeit“ (S_{311}) ergibt sich nach Formel (20) durch die Aufsummierung der Serviceoperationsgranularitäten:

$$SG(S_{311}) = \sum_{SO_i \in SO(S_{311})} SOG(SO_i) = SOG(SO_{3111}) + SOG(SO_{3112}) = 17 + 17 = 34$$

Schließlich kann auch die durchschnittliche Granularität der Serviceoperationen entsprechend Formel (21) ermittelt werden:

$$DSOG(S_{311}) = \frac{SG(S_{311})}{|\,SO(S_{311})\,|} = \frac{34}{|\,\{SO_{3111}, SO_{3112}\}\,|} = \frac{34}{2} = 17$$

Auf der Grundlage der nun vorliegenden Metrikwerte kann eine am spezifischen Entwurfskontext orientierte verbale Einschätzung der Servicegranularität erfolgen. Für eine vollständige Beurteilung müssten nun individuelle Klassengrenzen gemäß Bild 6-5 definiert werden. Aufgrund des Beispielcharakters soll diese Auswertung jedoch abgekürzt werden: Der betrachtete Service S_{311} wird als fein- bis mittelgranular eingestuft. Seine Operationen SO_{3111} und SO_{3112} besitzen einen homogenen Granularitätsgrad.

Schwachstellen der entwickelten Granularitätsmetriken

Das soeben geschilderte Vorgehen zur Bewertung der Servicegranularität besitzt einige, nicht unerhebliche Schwächen. Es basiert auf der Grundannahme von einem direkten Zusammenhang zwischen der Granularität eines Service und der Komplexität der von ihm verwendeten Daten. Diese These ist nicht allgemein gültig. Es lassen sich durchaus Gegenbeispiele von Services finden, die, trotz einfach strukturierter Eingangs- und Ausgangsdaten, umfangreiche Funktionen ausführen und demzufolge eine grobe Granularität aufweisen.[305]

Außerdem lässt sich die geschilderte Bewertung nur dann sinnvoll anwenden, wenn die Serviceschnittstelle und die Serviceimplementierung minimal entworfen wurden. Das heißt, dass an der Eingangsschnittstelle nur die von der Serviceoperation tatsächlich benötigten Daten übergeben werden dürfen und die Operation selbst keine überflüssigen Daten abruft. An der Ausgangsschnittstelle darf die Serviceoperation nur die Daten zur Verfügung stellen, die laut funktionaler Anforderung notwendig sind. Es muss also sichergestellt werden, dass ein Service keine unbenötigten oder redundanten Daten verarbeitet oder bereitstellt.

---

[305] Eine Serviceoperation, die beispielsweise nur eine Kundennummer als Eingangsparameter verarbeitet und einen wenig komplexen Datensatz zurückliefert, sollte nach der beschriebenen Ausgangsthese eine feine Granularität aufweisen. Dies trifft zu, falls die betrachtete Operation z. B. nur die Kundenstammdaten zur Kundennummer aus einer Datenbank abruft (vgl. Operation SO_{1131} des Fallbeispiels). Allerdings kann eine Operation auf der Grundlage der Kundennummer auch eine umfangreiche und komplizierte Bonitätsanalyse für den betreffenden Kunden durchführen. In diesem Fall besitzt sie, trotz geringer Komplexität der Ein- und Ausgangsdaten, eine mittlere bis grobe Granularität.

Aus den genannten Gründen sind die entwickelten Granularitätsmetriken nur begrenzt für die Einschätzung der Servicegranularität einsetzbar. Sie können lediglich erste Hinweise zum Granularitätsgrad geben und sind nicht geeignet, eine endgültige Aussage diesbezüglich zu treffen. Es ist anzunehmen, dass keine einzelne Metrik existiert, mit deren Hilfe die Servicegranularität zuverlässig und rein objektiv ermittelt werden könnte. Deshalb soll im folgenden Abschnitt ein qualitatives Bewertungsverfahren entwickelt werden, um das oben geschilderte Vorgehen geeignet zu ergänzen.

Vorschlag für eine qualitative Granularitätsbewertung

In diesem Abschnitt soll ein Konzept für die Bewertung der Servicegranularität entwickelt werden, das sich auf verschiedene qualitative Kriterien stützt. Einen Überblick über das zugrunde liegende Bewertungsmodell gibt Bild 6-8.

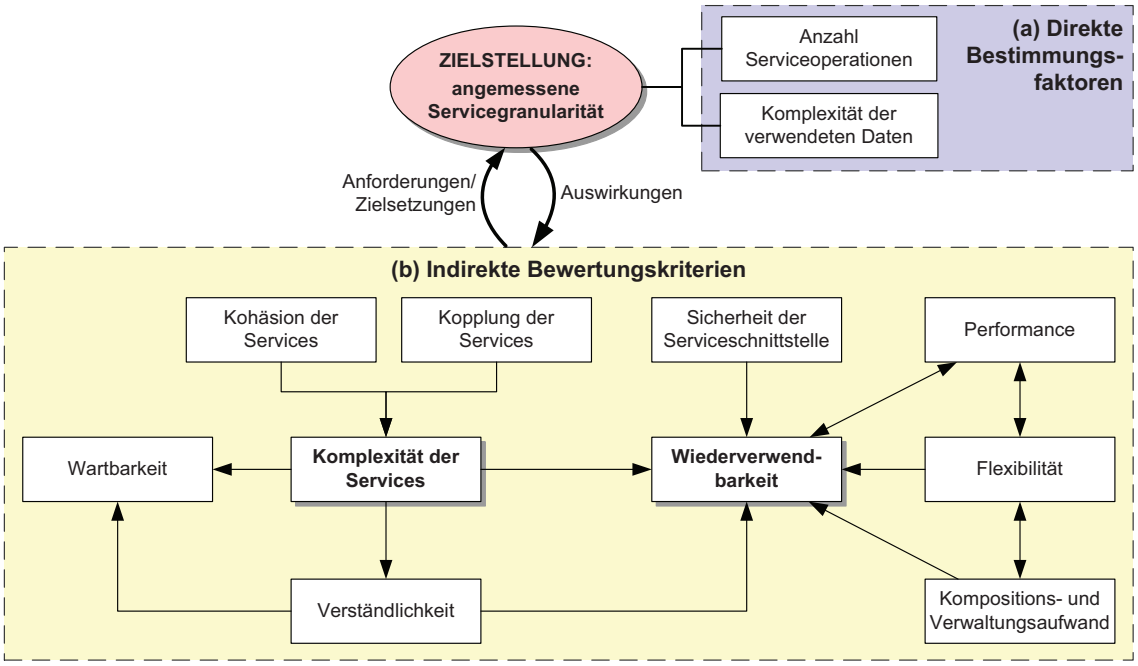


Bild 6-8: Multikriterielles Modell zur Granularitätsbewertung

Im violett hinterlegten Bereich ist angedeutet, dass der Granularitätsgrad eines Service anhand von direkten Bestimmungsfaktoren geschätzt werden kann. Dieser Ansatz wurde im vorangegangenen Abschnitt verfolgt. Dabei wurde festgestellt, dass eine direkte Messung der Granularität nicht zuverlässig möglich ist. Ursächlich hierfür ist der Umstand, dass es kaum quantitative Merkmale zur objektiven Ermittlung der Granularität von Software gibt.

Im Folgenden soll ein qualitativer Bewertungsansatz verfolgt werden. Er stützt sich auf die Analyse indirekter Kriterien, die mit der Servicegranularität in Wechselwirkung stehen, dargestellt durch den gelben Bereich in Bild 6-8. Diese Kriterien wurden durch eine Literaturrecherche[306] und die Auswertung von Arbeitstreffen mit SOA-Beratern[307] zusammengetragen. Sie beeinflussen einerseits die Wahl einer angemessenen Granularität, indem sie hierfür bestimmte Anforderungen und Rahmenbedingungen formulieren. So wird ein Service u. a. auch im Hinblick auf gute Wiederverwendbarkeit, geringe Kopplung und hohe Kohäsion sowie die erforderliche Performance zugeschnitten und entworfen. Umgekehrt wirkt sich die gewählte Servicegranularität wiederum auf diese Kriterien aus. Durch die Definition des Funktionsumfangs eines Service werden dessen fachliche Grenzen abgesteckt, wodurch die Servicekohäsion und die Kopplung zu anderen Services bestimmt werden. Die Granularität eines Service beeinflusst außerdem, ob dieser einfach, sicher, flexibel und mit geringem Aufwand genutzt und wieder verwendet werden kann.

Das soeben beschriebene Modell stellt ein *multikriterielles Bewertungsproblem* dar.[308] Das übergeordnete Ziel besteht in der Bestimmung der Granularität eines Service und der Beurteilung ihrer Angemessenheit in Bezug auf die Entwurfsziele. Die Servicegranularität soll indirekt anhand von mehreren verschiedenen Kriterien (vgl. Bild 6-8) ermittelt und bewertet werden. Die verwendeten Kriterien sind durch ein Beziehungsnetz untereinander verbunden. Bei den Verbindungen, die in Bild 6-8 durch Pfeile dargestellt wurden, kann es sich um einseitige oder wechselseitige Einflüsse bzw. Abhängigkeiten handeln. Beispielsweise bestimmen Kohäsion und Kopplung die Komplexität eines Service, während sich Flexibilität und Kompositionsaufwand gegenseitig beeinflussen. Außerdem können sich die Bewertungskriterien hinsichtlich der Zielerreichung indifferent, komplementär oder konfliktär zueinander verhalten.[309] So wirkt z. B. eine hohe Flexibilität positiv auf die Wiederverwendbarkeit eines Service, wohingegen sich Flexibilität und Performance von Services meist konkurrierend gegenüber stehen.

---

306 Vgl. insbesondere Feuerlicht /Service Granularity/ 315 f., Whitehead /Component-Based Development/ 22 f., Marks, Bell /Planning and Implementation/ 124 f., Szyperski /Component Software/ 149 f., De Jonge /Build-Level Components/ 589 und Hopkins /Component Primer/ 29

307 Vor allem Gespräche mit dem SOA-Architekten Bernd Trops der Firma Oracle und ein Treffen mit Beratern von IBM.

308 Vgl. für die folgenden Sätze Roth /Lösungsverfahren/ 24 f.

309 Vgl. Rommelfanger, Eickemeier /Entscheidungstheorie/ 142

Zur Bewertung der Servicegranularität wird die Verwendung der *Scoring-Methode* als Vertreter der multiattributiven Verfahren[310] vorgeschlagen. Die Scoring-Methode ist ein additives Zielgewichtungsverfahren, das eine Gesamtbewertung durch die Aggregation gewichteter Ausprägungen verschiedener Bewertungskriterien erzielt.[311] Sie kann nach dem in der Tabelle 6-8 dargestellten Schema durchgeführt werden.

(1) Bewertungskriterium	(2) Gewicht	(3) Ausprägung	(4) Zielbeitrag
$BK_k \quad (k = 1 \ldots K)$	$g_k \quad (0 < g_k < 1)$	$a_k \quad (0 \le a_k \le 10)$	$z_k = g_k \cdot a_k$
1 Wiederverwendbarkeit	…	…	…
2 Performance	…	…	…
…	…	…	…
	$\sum_{k=1}^{K} g_k = 1$		$Z_{Granularität} = \sum_{k=1}^{K} z_k$

Tabelle 6-8: Multikriterielles Bewertungsschema der Scoring-Methode

In einem ersten Schritt sind aus den nichtfunktionalen Qualitätsanforderungen der Kunden alle für die Granularitätsbewertung relevanten Kriterien zusammenzustellen und aufzulisten. Anschließend ist für jedes Kriterium ein Gewicht g_k festzulegen, das dessen relative Bedeutung repräsentiert.[312] Die Kriteriengewichte beschreiben, welche Relevanz bestimmte Aspekte hinsichtlich der Servicegranularität besitzen. Die Gewichte lassen sich aus den Entwurfszielen ableiten und sollten für jeden Entwurf, ggf. sogar für jede Serviceebene (vgl. Abschnitt 3.4), individuell bestimmt werden.

Für die Festlegung der Kriteriengewichte gibt es zahlreiche Verfahren. Ein sehr einfaches Vorgehen stellt das Rangsummenverfahren[313] dar, das in Tabelle 6-9 beispielhaft beschrieben ist. Vor der Berechnung der Kriteriengewichte müssen die Bewertungskriterien zunächst in einer Rangfolge angeordnet werden.[314] Hierzu werden alle Kriterien paarweise miteinander verglichen und die Ergebnisse in die Matrix der Tabelle 6-9 eingetragen. Der Wert 1 in Zeile *x* und Spalte *y* bedeutet, dass Kriterium *x* wichtiger als Kriterium *y* ist. Der Eintrag ½ steht für eine gleichgroße Relevanz der

310 Vgl. Weber /Mehrkriterielle Entscheidungen/ 11
311 Vgl. Roth /Lösungsverfahren/ 63
312 Vgl. Roth /Lösungsverfahren/ 63
313 Vgl. Weber /Mehrkriterielle Entscheidungen/ 51-54

verglichenen Kriterien. Der Rang r_k eines jeden Kriteriums ergibt sich nach absteigendem Wert der jeweiligen Zeilensummen. Das Rangsummenverfahren ordnet gemäß der Formel in Tabelle 6-9 jedem Kriterium entsprechend dessen Rang ein Gewicht zu.[315] Dabei sinken die Gewichte mit abnehmendem Rang proportional. In Tabelle 6-9 wurde beispielsweise die Flexibilität für wichtiger als alle anderen Kriterien befunden und besitzt deshalb den höchsten Rang. Dementsprechend ist das Gewicht dieses Kriteriums mit rund 0,33 am größten.

Bewertungskriterien		**Paarvergleichsmatrix**					**Zeilen-summe**	**Rang**	**Gewicht**
k	$BK_k \quad (K=5)$	1	2	3	4	5		r_k	$g_k = \frac{K - r_k + 1}{K(K+1)/2}$
1	Kopplung	–	½			½	1	4	$2/15 \approx 0{,}13$
2	Kohäsion	½	–			1	1,5	3	$3/15 = 0{,}20$
3	Wiederverwendbarkeit	1	1	–		1	3	2	$4/15 \approx 0{,}27$
4	Flexibilität	1	1	1	–	1	4	1	$5/15 \approx 0{,}33$
5	Performance	½				–	0,5	5	$1/15 \approx 0{,}07$

Tabelle 6-9: Ermittlung der Kriteriengewichte nach dem Rangsummenverfahren

Bei der Festlegung der Gewichtung sollten nach Möglichkeit die Interdependenzen zwischen den Bewertungskriterien berücksichtigt werden.[316] So können durch die Vergabe unterschiedlicher Gewichte Kriterienkonflikte aufgelöst werden. Die Beziehungen im Kriteriensystem können beim Rangsummenverfahren nur implizit durch den Bewertenden während des paarweisen Kriterienvergleichs berücksichtigt werden. Komplexere Bewertungsverfahren, wie der Analytic Hierarchy Process (AHP) nach Saaty[317], verarbeiten bei der Festlegung der Gewichte explizit die Abhängigkeiten innerhalb eines hierarchischen Zielsystems.

Nachdem nun für jedes Kriterium ein Gewicht festgelegt wurde, wird im dritten Schritt des Scoring-Verfahrens der betrachtete Service hinsichtlich jedes Kriteriums bewertet. Dies ergibt die Ausprägungen a_k der einzelnen Kriterien, die in Form von dimensionslosen Punktwerten (Scores) vergeben werden.[318] Hierfür kann z. B. eine Bewertungsskala von 0

314 Vgl. für die folgenden Sätze Weber /Mehrkriterielle Entscheidungen/ 51 f.
315 Vgl. für diesen und den folgenden Satz Weber /Mehrkriterielle Entscheidungen/ 54
316 Vgl. Roth /Lösungsverfahren/ 67
317 Vgl. Saaty /Analytic Hierarchy Process/
318 Vgl. für die folgenden Sätze Roth /Lösungsverfahren/ 73

(sehr schlecht) bis 10 (sehr gut) verwendet werden.[319] Die Bewertung wird meist durch eine subjektive Einschätzung vorgenommen. Hierbei können die Punktwerte direkt vergeben werden oder verbale Beurteilungen werden entsprechend einer Zuordnungsvorschrift in numerische Punktwerte übertragen.[320] Es können aber auch quantitative Ausprägungen verwendet werden. So können beispielsweise für Kopplung und Kohäsion die Werte der in den Abschnitten 6.1.2 und 6.1.3 entwickelten Metriken herangezogen werden. Solche quantitativen Ausprägungen müssen durch Klassenbildung oder eine geeignete Skalierung bzw. Normierung in Punktwerte übertragen werden.[321]

Im vierten Schritt des Scoring-Verfahrens wird der Zielbeitrag z_k bezüglich jedes Kriteriums durch die Multiplikation des Kriteriengewichts mit der jeweiligen Ausprägung errechnet.[322] Die Gesamtbewertung Z des betrachteten Service hinsichtlich aller Kriterien ergibt sich schließlich durch die Aufsummierung aller Zielbeiträge. Dieser Gesamtwert gibt Hinweise darauf, ob sich die gewählte Granularität positiv oder negativ auf die untersuchten Kriterien auswirkt. Die Granularität kann als angemessen angesehen werden, wenn die mit den Kriterien verbundenen Zielvorstellungen erreicht wurden. Sollten bei wichtigen Kriterien unbefriedigende Punktwerte aufgetreten sein, so ist der Zuschnitt des Service ggf. zu überarbeiten.

Das beschriebene Bewertungsverfahren auf der Grundlage der Scoring-Methode besitzt den Vorteil, dass es einfach anzuwenden ist und nachvollziehbare Ergebnisse liefert.[323] Es stellt aber gleichzeitig hohe Anforderungen an den Bewertenden. Dieser muss zunächst die für die Einschätzung der Granularität relevanten Kriterien aus den Entwurfszielen ableiten. Anschließend sollte er seine Präferenzen bezüglich dieser Kriterien unter Berücksichtigung der Kriterienbeziehungen durch numerische Gewichte ausdrücken können. Schließlich muss er in der Lage sein, die Ausprägung der Kriterien für den betrachteten Entwurf durch Punktwerte zu beurteilen.

Trotz dieser Anforderungen trägt die Beschäftigung mit den Einflussfaktoren und Auswirkungen der gewählten Servicegranularität zur gedanklichen Durchdringung dieser Problematik bei. Als Ergebnis steht weniger der genaue Gesamtpunktwert im Vordergrund,

---

319 Vgl. Weber /Mehrkriterielle Entscheidungen/ 56
320 Vgl. Roth /Lösungsverfahren/ 74 f.
321 Vgl. Weber /Mehrkriterielle Entscheidungen/ 55 f. und Roth /Lösungsverfahren/ 74
322 Vgl. für diesen und den folgenden Satz Roth /Lösungsverfahren/ 73

sondern dessen Entstehung und das Verständnis der Vor- und Nachteile der unterschiedlichen Granularitätsgrade von Services.[324]

6.2 Allgemeine Bewertung der Anwendung von AD im Entwurf von SOA

In den vorangegangenen Abschnitten wurden Metriken und ein Bewertungsverfahren entwickelt, um den Beitrag von Axiomatic Design fassbar zu machen. Im Folgenden soll sich nun eine allgemeine Betrachtung der Vor- und Nachteile sowie Grenzen der Anwendung von Axiomatic Design für den Entwurf serviceorientierter Architekturen anschließen.

6.2.1 Vorteile der Anwendung von Axiomatic Design

Das dem Axiomatic Design zugrunde liegende Domänenkonzept verfolgt eine konsequente Trennung von Anforderungen an den Entwurf (FAs) und möglichen Lösungen (DPs). Diese Abgrenzung erfordert zu Beginn des Entwurfsprozesses die explizite Formulierung der Entwurfsziele, was in der Praxis sonst oft vernachlässigt wird.[325] Die daraus resultierende konsequente Ausrichtung des weiteren Entwurfs an den formulierten Anforderungen führt i. d. R. zu einem zielkonformen und minimalen Ergebnis.[326] Die während des Entwurfs mit Axiomatic Design durchgeführte hierarchische Dekomposition ist ein bewährtes Strukturierungsmittel des Software Engineering.[327] Sie ermöglicht eine schrittweise Verfeinerung von Anforderungen und Lösungen. Diese fortschreitende Detaillierung führt zu einer Verteilung der dem Entwurfsgegenstand zugrunde liegenden Komplexität auf die verschiedenen Dekompositionsebenen. Auf jeder Ebene entstehen somit Entwurfsbestandteile mit einer überschaubaren Komplexität. Die verschiedenen Abstraktionsebenen erhöhen zudem die Verständlichkeit des Gesamtentwurfs.[328] Bezogen auf den Entwurf serviceorientierter Architekturen ergibt sich aus dieser Strukturierung eine gute Übersicht über die Servicehierarchie inklusive Kompositionsbeziehungen und Abhängigkeiten zwischen den Services.

---

323 Vgl. für diesen Absatz Roth /Lösungsverfahren/ 83
324 Vgl. Szyperski /Component Software/ 139
325 Vgl. Suh /Axiomatic Design/ 5 und Suh /Principles/ 32
326 Vgl. Suh /Principles/ 39
327 Vgl. Riebisch /Architektur- und Komponentenentwicklung/ 74 f.
328 Vgl. Balzert /Software-Qualitätssicherung/ 559 ff.

Das Unabhängigkeits- und Informationsaxiom von Axiomatic Design unterstützen den Designer bei den zu treffenden Entwurfsentscheidungen.[329] Durch die Befolgung dieser Axiome werden insbesondere die Kopplung und Kohäsion der Entwurfsbestandteile positiv beeinflusst. Das Unabhängigkeitsaxiom fordert die Minimierung der Abhängigkeiten und die Vermeidung zyklischer Kopplungen zwischen den Entwurfselementen. Im Kontext serviceorientierter Architekturen führt die Einhaltung dieses Axioms zu lose gekoppelten, weitgehend autonomen bzw. kohärenten Services und wohl definierten, azyklischen Beziehungen zwischen diesen Services. Die vollständige Entwurfsmatrix von Axiomatic Design dokumentiert alle vorhandenen Abhängigkeiten und ermöglicht somit die Analyse von Kopplung und Kohäsion im SOA-Entwurf. Hierdurch können potenzielle Schwachstellen frühzeitig identifiziert und überarbeitet werden.

Durch den Entwurf einer SOA mittels Axiomatic Design wird auch eine angemessene Servicegranularität gefördert. Während des Entwurfsprozesses wird jeder Zweig der Dekompositionshierarchie solange verfeinert, bis die Blattelemente genügend Details für eine spätere Umsetzung des Entwurfs beinhalten.[330] Die Tiefe der Dekomposition ist somit abhängig von der erforderlichen fachlichen Konkretisierung des Entwurfs. Aus den Blattmodulen der Dekompositionshierarchie lassen sich die Serviceoperationen ableiten und zu Services zusammenfassen. Je nach Dekompositionstiefe der entsprechenden Blattmodule und Abstraktionsgrad der verfeinerten funktionalen Anforderung entstehen Services unterschiedlicher Granularität. Der Funktionsumfang der Services und somit ihre Granularität sind folglich direkt vom Abstraktionsgrad der korrespondierenden FAs abhängig. Die Servicegranularität ist angemessen, da sie sich nach dem Abstraktionsgrad der zugrunde liegenden fachlichen Anforderung richtet.

Andere Aspekte des Entwurfs mit Axiomatic Design beeinflussen ebenfalls die Granularität der entstehenden Services. So führt die Berücksichtigung des Unabhängigkeitsaxioms, wie oben beschrieben, zur Begrenzung der Kopplung, zur Vermeidung von Zyklen und zu einer guten Kohäsion der Services. Die Befolgung der Forderung nach einem minimalen Entwurf[331] ermöglicht zudem die Erkennung und

329 Vgl. Suh /Axiomatic Design/ 5 f.
330 Vgl. Suh /Axiomatic Design/ 29
331 Vgl. Suh /Principles/ 52 f. (Korollar 2)

Vermeidung von Redundanzen im Entwurf. Jeder dieser Aspekte trägt zu einer sinnvollen und angemessenen Granularität der entworfenen Services bei.

Aus eigener Erfahrung im Entwurf mit Axiomatic Design habe ich insbesondere die Trennung von Entwurfszielen und -lösungen als vorteilhaft empfunden. Die explizite Formulierung der funktionalen Anforderungen „zwingt“ zur intensiven Beschäftigung mit dem Entwurfsgegenstand und den gestellten Zielen. Zudem unterstützt die klare Abfolge von Entwurfsschritten in Axiomatic Design speziell die Durchdringung und Ordnung eines unstrukturierten Entwurfsumfelds. Die vollständige Entwurfsmatrix bietet als Resultat des Entwurfsprozesses einen umfassenden und tiefgehenden Einblick in das Entwurfsergebnis.

6.2.2 Probleme und Grenzen der Anwendung von Axiomatic Design

Der Entwurf mit Axiomatic Design erfordert vom Designer eine gewisse Disziplin in der Einhaltung der vorgegebenen Entwurfsschritte. Die Befolgung dieser Schritte ist häufig mit einem erhöhten Zeit- und Ressourcenbedarf verbunden.[332] Für jede generierte Dekompositionsebene müssen entsprechende Entwurfsgleichungen oder -matrizen aufgestellt[333] und die getroffenen Entscheidungen dokumentiert werden[334]. Diese Aktivitäten und der mit ihnen verbundene Aufwand sind der Preis für die im vorangegangenen Abschnitt genannten Vorteile und die Verbesserung der Entwurfsqualität. Es ist zu erwarten, dass sich aufgrund der geringeren Anzahl von Fehlern und Schwachstellen im Entwurf der anfängliche Mehraufwand meist recht schnell durch die erzielten Einsparungen in späteren Entwicklungsphasen rentiert.[335]
Das strenge formale Vorgehen mit Axiomatic Design könnte den Eindruck erwecken, dass die Kreativität und die Flexibilität der Designer eingeschränkt werden. Doch trotz der methodischen Unterstützung und Strukturierung durch Axiomatic Design wird bei der Generierung von Lösungen weiterhin auf die Erfahrung und Innovationskraft der am Entwurf beteiligten Personen gesetzt.[336]

Hinsichtlich der Anwendung von Axiomatic Design muss angemerkt werden, dass nicht alle Entwurfsschritte durchgehend von AD selbst unterstützt werden. Beispielsweise muss

---

332 Vgl. Suh /Axiomatic Design/ 111
333 Vgl. Suh /Axiomatic Design/ 30
334 Vgl. Suh /Axiomatic Design/ 101
335 Vgl. Suh /Axiomatic Design/ 111

für die Ermittlung der Kundenanforderungen auf andere Methoden, wie das Quality Function Deployment, zurückgegriffen werden.[337]

Die dargestellte Methodik darf nicht als selbstständiges Vorgehensmodell für die Entwicklung einer SOA betrachtet werden. Axiomatic Design unterstützt vornehmlich die Analyse und den Grobentwurf einer SOA (vgl. Bild B-2). Es kann und sollte deshalb in bestehende Vorgehensmodelle, wie die in Abschnitt 4.1 beschriebenen, eingebettet werden.[338] Dort kann AD insbesondere die Phasen der serviceorientierten Analyse und des serviceorientierten Entwurfs bis hin zur Definition der Servicehierarchie und der Serviceschnittstellen unterstützen. Die Detaillierung und Überarbeitung der Entwurfsergebnisse im Rahmen des Feinentwurfs werden dann wieder durch Phasen und Werkzeuge traditioneller Softwarevorgehensmodelle abgebildet.

Bei meiner eigenen Entwurfstätigkeit mit Axiomatic Design ist mir insbesondere der hohe Dokumentationsaufwand aufgefallen, der nur durch den Einsatz geeigneter Software[339] angemessen reduziert werden kann. Die Entwurfsmatrix wird speziell bei umfangreichen Entwürfen leicht unübersichtlich und unhandlich. Größere Änderungen an einer solchen Matrix sind ohne computergestützte Werkzeuge kaum beherrschbar. Trotz der bestehenden Softwareunterstützung liegen die wesentlichen Faktoren für den erfolgreichen Abschluss eines Entwurfsvorhabens im Einflussbereich des Designers. Diesbezüglich habe ich es als Herausforderung empfunden, stets eine überschneidungsfreie Dekomposition der funktionalen Anforderungen und der korrespondierenden Designparameter sicherzustellen. Auch die Ermittlung aller vorhandenen Abhängigkeiten zwischen den Entwurfsbestandteilen wird mit zunehmendem Umfang der Entwurfsmatrix schwieriger.

Zusammenfassend kann festgehalten werden, dass Axiomatic Design seinen Beitrag zum SOA-Entwurf liefert, indem es den Entwurfsprozess strukturiert und dem Designer methodische Unterstützung sowie Entscheidungshilfen an die Hand gibt. Die genannten Vorzüge von Axiomatic Design führen i. d. R. zu Entwurfsergebnissen mit überschaubarer Komplexität und tragen zur Erfüllung der Architekturziele serviceorientierter Architekturen bei. Die Befolgung der Entwurfsschritte und Axiome verursacht jedoch

336 Vgl. Suh /Axiomatic Design/ 5 und 31
337 Vgl. Suh /Axiomatic Design/ 14
338 Vgl. Suh /Axiomatic Design/ 266

einen gewissen Mehraufwand bei der Erarbeitung und Dokumentation der Entwurfsergebnisse.

[339] Das Softwarepaket Acclaro DFSS (http://www.dfss-software.com/) unterstützt neben anderen Methoden insbesondere die Entwurfsschritte von Axiomatic Design.

7 Schlussbemerkungen und Ausblick

In diesem Bericht wurde die Entwurfsmethodik Axiomatic Design vorgestellt und für den Entwurf serviceorientierter Architekturen angepasst. Hierzu wurden die Bestandteile und Axiome der Methode geeignet für den Architekturentwurf übertragen bzw. interpretiert. Im Rahmen der Anwendung dieser Methode an einem Fallbeispiel wurde außerdem eine Möglichkeit zur Darstellung von SOA-Entwürfen mit der UML vorgestellt. Im letzten Teil dieses Berichtes wurde ein Messkonzept entwickelt, das verschiedene Komplexitätsmetriken zur quantitativen Bewertung von Kopplung, Kohäsion und Granularität von Services definiert. Zusätzlich wurde ein multikriterielles Bewertungsverfahren zur Einschätzung der Servicegranularität vorgeschlagen.
Im Ergebnis dieses Berichtes sollte deutlich werden, dass Axiomatic Design, im Vergleich zu anderen Methoden für den SOA-Entwurf, ein sehr strukturiertes Vorgehen während der Analyse- und Entwurfsphase ermöglicht. Zudem unterstützt Axiomatic Design explizit die Erreichung einiger Architekturziele von serviceorientierten Architekturen. Mit dem Unabhängigkeits- und Informationsaxiom bietet es außerdem wertvolle Entscheidungshilfen und Gestaltungsrichtlinien bei Entwurfsentscheidungen. Mithilfe der Metriken des entwickelten Messkonzepts ist eine quantitativ fundierte Analyse der Qualität der mit Axiomatic Design generierten SOA-Entwürfe möglich. Auf der Grundlage der Analyseergebnisse kann der Beitrag der Anwendung von Axiomatic Design im Entwurf serviceorientierter Architekturen besser bewertet werden.

Während der Analyse des Beitrags von Axiomatic Design wurden bereits einige Schwachstellen der Entwurfsmethode erläutert. Hinsichtlich der Übertragung von AD für den Entwurf serviceorientierter Architekturen ist insbesondere die Messung des Informationsgehalts eines SOA-Entwurfs mit Schwierigkeiten verbunden. Mit der Integration von Komplexitätsmetriken in die Berechnung des Informationsgehalts wurde bereits ein Vorschlag zur Lösung dieses Problems unterbreitet, den es weiter zu untersuchen gilt.
Zusätzlich gibt es an einer Reihe anderer Stellen Verbesserungsmöglichkeiten und damit Ansatzpunkte für zukünftige Forschungstätigkeiten. Aus Sicht der Praxis stellt insbesondere die gegenwärtige Beschränkung von Axiomatic Design auf den Top-Down-Entwurf ein Hindernis für die Anwendung dar. Die explizite Berücksichtigung einer bereits

existierenden Systemlandschaft oder einer Sammlung von bestehenden Services ist momentan noch nicht möglich. Es muss deshalb nach Wegen gesucht werden, vorhandene Lösungen im Rahmen eines Bottom-Up-Vorgehens in den Entwurf mit Axiomatic Design einzubinden.

Für die Unterstützung von Entwurfsentscheidungen erscheint es außerdem hilfreich, in Ergänzung zum Unabhängigkeits- und Informationsaxiom, weitere Axiome oder Theoreme zu entwickeln. Diese sollten spezielle Richtlinien oder Empfehlungen für die Gestaltung serviceorientierter Architekturen formulieren. Um den mit Axiomatic Design verbundenen erhöhten Entwurfs- und Dokumentationsaufwand zu reduzieren, wäre es zweckdienlich, aus der erarbeiteten Entwurfsmatrix automatisch die entsprechenden UML-Diagramme des SOA-Entwurfs generieren zu können.

Im Hinblick auf das erarbeitete Messkonzept steht eine empirische oder theoretische Validierung der entwickelten Komplexitätsmetriken für SOA-Entwürfe noch aus. In einem nachfolgenden Bericht sollen diese Metriken zum Vergleich von Axiomatic Design mit einer anderen SOA-Entwurfsmethodik eingesetzt werden.

Literaturverzeichnis

Balzert /Software-Entwicklung/

Helmut Balzert: Lehrbuch der Software-Technik (Band 1): Software-Entwicklung. 2. Aufl., Heidelberg - Berlin 2001.

Balzert /Software-Qualitätssicherung/

Helmut Balzert: Lehrbuch der Software-Technik (Band 2): Software-Management, Software-Qualitätssicherung, Unternehmensmodellierung. Heidelberg - Berlin 1998.

Baresi u. a. /UML-Profile/

Luciano Baresi, Reiko Heckel, Sebastian Thöne, Dániel Varró: A UML-Profile for Service-Oriented Architectures. http://www.oasis-open.org/archives/semantic-ex/200610/pdf00000.pdf, Paderborn 2003, Abruf: 2007-04-21.

Bass, Clements, Kazman /Architecture/

Len Bass, Paul Clements, Rick Kazman: Software Architecture in Practice. Reading, Massachusetts u. a. 1998.

Behrens u. a. /Architekturbeschreibung/

Jan Behrens, Simon Giesecke, Henning Jost, Jasminka Matevska, Ulf Schreier: Architekturbeschreibung. In: Ralf Reussner, Wilhelm Hasselbring (Hrsg.): Handbuch der Software-Architektur. Heidelberg 2006, S. 35-64.

Benatallah u. a. /Composition/

Boualem Benatallah, Remco M. Dijkman, Marlon Dumas, Zakaria Maamar: Service Composition: Concepts, Techniques, Tools and Trends. In: Zoran Stojanovic, Ajantha Dahanayake (Hrsg.): Service-Oriented Software System Engineering: Challenges and Practices. Hershey, USA u. a. 2005, S. 48-66.

Bieberstein u. a. /SOA Compass/

Norbert Bieberstein, Sanjay Bose, Marc Fiammante, Keith Jones, Rawn Shah: Service-Oriented Architecture Compass: Business Value, Planning, and Enterprise Roadmap. Upper Saddle River u. a. 2006.

Bieman, Kang /Cohesion and Reuse/

James M. Bieman, Byung-Kyoo Kang: Cohesion and Reuse in an Object-Oriented System. In: Mansur H. Samadzadeh, Mansour K. Zand (Hrsg.): Proceedings of the 1995 Symposium on Software Reusability (SSR'95). Seattle 1995, S. 259-262.

Bieman, Kang /Design-Level Cohesion/

James M. Bieman, Byung-Kyoo Kang: Measuring Design-Level Cohesion. In: IEEE Transactions on Software Engineering. Band 24, Nr. 2, 1998, S. 111-124.

Bosch /Statistik-Taschenbuch/

Karl Bosch: Statistik-Taschenbuch. 3. Aufl., München - Wien 1998.

Briand, Daly, Wüst /Cohesion Measurement/

Lionel C. Briand, John W. Daly, Jürgen K. Wüst: A Unified Framework for Cohesion Measurement in Object-Oriented Systems. In: Empirical Software Engineering. Band 3, Nr. 1, 1998, S. 65-117.

Briand, Daly, Wüst /Coupling Measurement/

Lionel C. Briand, John W. Daly, Jürgen K. Wüst: A Unified Framework for Coupling Measurement in Object-Oriented Systems. In: IEEE Transactions on Software Engineering. Band 25, Nr. 1, 1999, S. 91-121.

Briand, Morasca, Basili /Measures for Design/

Lionel C. Briand, Sandro Morasca, Victor R. Basili: Defining and Validating Measures for Object-Based High-Level Design. In: IEEE Transactions on Software Engineering. Band 25, Nr. 5, 1999, S. 722-743.

Brown /Coupling/

Christopher A. Brown: Kinds of Coupling and Approaches to Deal With Them. In: Proceedings of ICAD2006, 4th International Conference on Axiomatic Design. Firenze 2006.

Bundschuh, Fabry /Aufwandschätzung/

Manfred Bundschuh, Axel Fabry: Aufwandschätzung von IT-Projekten. Bonn 2000.

Cardoso /Data-Flow Complexity/

Jorge Cardoso: About the Data-Flow Complexity of Web Processes. In: 6th International Workshop on Business Process Modeling, Development, and Support: Business Processes and Support Systems: Design for Flexibility. The 17th Conference on Advanced Information Systems Engineering (CAiSE'05). Porto 2005, S. 67-74.

Chidamber, Kemerer /Metrics Suite/

Shyam R. Chidamber, Chris F. Kemerer: A Metrics Suite for Object Oriented Design. In: IEEE Transactions on Software Engineering, Band 20, Nr. 6, 1994, S. 476-493.

Clapis, Hintersteiner /Software Development/

Paul J. Clapis, Jason D. Hintersteiner: Enhancing Object-Oriented Software Development Through Axiomatic Design. In: Proceedings of ICAD2000, First International Conference on Axiomatic Design. Cambridge 2000, S. 272-277.

Clark, Holton /Graphentheorie Grundlagen/

John Clark, Derek A. Holton: Graphentheorie: Grundlagen und Anwendungen. Heidelberg - Berlin - Oxford 1994.

Darcy u. a. /Structural Complexity/

David P. Darcy, Chris F. Kemerer, Sandra A. Slaughter, James E. Tomayko: The Structural Complexity of Software: An Experimental Text. In: IEEE Transactions on Software Engineering. Band 31, Nr. 11, 2005, S. 982-995.

De Jonge /Build-Level Components/

Merijn de Jonge: Build-Level Components. In: IEEE Transactions on Software Engineering. Band 31, Nr. 7, 2005, S. 588-600.

DIN /Bewerten von Softwareprodukten. DIN 66272/

DIN (Hrsg.): Bewerten von Softwareprodukten: Qualitätsmerkmale und Leitfaden zu ihrer Verwendung. DIN 66272. o. O. 1994.

Dostal u. a. /Web Services/

Wolfgang Dostal, Mario Jeckle, Ingo Melzer, Barbara Zengler: Service-orientierte Architekturen mit Web Services: Konzepte – Standards – Praxis. München 2005.

Dumke /Maß/

Reiner Dumke: Softwareentwicklung nach Maß: Schätzen – Messen – Bewerten. Braunschweig - Wiesbaden 1992.

Ebert /Qualitätsmanagement/

Christof Ebert: Metriken im Qualitätsmanagement. In: Christof Ebert, Reiner Dumke (Hrsg.): Software-Metriken in der Praxis: Einführung und Anwendung von Software-Metriken in der industriellen Praxis. Berlin u. a. 1996, S. 45-83.

Erl /Service-Oriented Architecture/

Thomas Erl: Service-Oriented Architecture: Concepts, Technology, and Design. Upper Saddle River u. a. 2005.

Fenton, Pfleeger /Software Metrics/

Norman E. Fenton, Shari Lawrence Pfleeger: Software Metrics: A Rigorous and Practical Approach. 2. Aufl., London u. a. 1997.

Feuerlicht /Service Granularity/

George Feuerlicht: Service Granularity Considerations Based on Data Properties of Interface Parameters. In: International Journal of Computer Systems Science & Engineering. Band 21, Nr. 4, 2006, S. 315-320.

Fiege, Stelzer /Analyse/

René Fiege, Dirk Stelzer: Analyse des Beitrages von Axiomatic Design zum Entwurf Serviceorientierter Architekturen. In: Udo Bankhofer, Peter Gmilkowsky, Volker Nissen, Dirk Stelzer (Hrsg.): Ilmenauer Beiträge zur Wirtschaftsinformatik. Nr. 2006-06, Ilmenau 2006.

Fiege, Stelzer /Modellierung/

René Fiege, Dirk Stelzer: Modellierung Serviceorientierter Architekturen mit Axiomatic Design – Analyse des Beitrages zur Verbesserung der Entwurfsqualität.. In: Udo Bankhofer, Volker Nissen, Dirk Stelzer, Steffen Straßburger (Hrsg.): Ilmenauer Beiträge zur Wirtschaftsinformatik. Nr. 2007-04, Ilmenau 2007.

Friedrich u. a. /Operating Systems/

L. Fernando Friedrich, John Stankovic, Marty Humphrey, Michael Marley, John Haskins Jr.: A Survey of Configurable, Component-Based Operating Systems for Embedded Applications. In: IEEE Micro. Band 21, Nr. 3, 2001, S. 54-68.

Gamma u. a. /Entwurfsmuster/

Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides: Entwurfsmuster: Elemente wiederverwendbarer objektorientierter Software. München u. a. 2001.

Henderson-Sellers /Object-Oriented Metrics/

Brian Henderson-Sellers: Object-Oriented Metrics: Measures of Complexity. Upper Saddle River 1996.

Henry, Kafura /Software Systems' Structure/

Sallie Henry, Dennis Kafura: The Evaluation of Software Systems' Structure Using Quantitative Software Metrics. In: Martin Shepperd (Hrsg.): Software Engineering Metrics (Band 1): Measures and Validations. London u. a. 1993, S. 99-111.

Hopkins /Component Primer/

Jon Hopkins: Component Primer. In: Communications of the ACM. Band 43, Nr. 10, 2000, S. 27-30.

Huhns, Singh /Service-Oriented Computing/

Michael Huhns, Munindar P. Singh: Service-Oriented Computing: Key Concepts and Principles. In: IEEE Internet Computing. Band 9, Nr. 1, 2005, S. 75-81.

Humm, Voß, Hess /Regeln/

Bernhard Humm, Markus Voß, Andreas Hess: Regeln für serviceorientierte Architekturen hoher Qualität. In: Informatik Spektrum. Band 29, Nr. 6, 2006, S. 395-411.

IEEE /Architectural Description. IEEE 1471-2000/

IEEE (Hrsg.): Recommended Practice for Architectural Description of Software-Intensive Systems: Description. IEEE Standard 1471-2000. o. O. 2000.

Jones /Software Measurement/

Capers Jones: Applied Software Measurement: Assuring Productivity and Quality. 2. Aufl., New York u. a. 1996.

Kecher /UML Handbuch/

Christoph Kecher: UML 2.0: Das umfassende Handbuch. 2. Aufl., Bonn 2006.

Krafzig, Banke, Slama /Enterprise SOA/

Dirk Krafzig, Karl Banke, Dirk Slama: Enterprise SOA: Service-Oriented Architecture Best Practices. Upper Saddle River 2005.

Kurniawan, Zhang, Tseng /Customers/

Sri Hartati Kurniawan, Mei Zhang, Mitchell M. Tseng: Connecting Customers in Axiomatic Design. In: Proceedings of ICAD2004, The Third International Conference on Axiomatic Design. Seoul 2004.

Lee /Complexity/

Taesik Lee: Complexity Theory in Axiomatic Design. Dissertation vorgelegt am Department of Mechanical Engineering des Massachusetts Institute of Technology. o. O. 2003.

Lee /Lecture Note/

Taesik Lee: Supplementary Lecture Note on Information Content. http://ocw.mit.edu/NR/rdonlyres/Mechanical-Engineering/2-882Spring-2005/08183F7F-9A9E-4284-ACE1-AD2DA668A4E4/0/lec228_supplemnt.pdf, o. O. 2005, Abruf: 2007-03-28.

Lee, Jeziorek /Off-Diagonal Term/

Taesik Lee, Peter N. Jeziorek: Understanding the Value of Eliminating an Off-Diagonal Term in a Design Matrix. In: Proceedings of ICAD2006, 4th International Conference on Axiomatic Design. Firenze 2006.

Lee, Suh /Design/

Taesik Lee, Nam P. Suh: Design of Systems (Lecture Note). http://ocw.mit.edu/NR/rdonlyres/Mechanical-Engineering/2-882Spring-2005/060DB164-18F2-4F6B-B353-8343227AF8D7/0/lec309.pdf, o. O. 2005, Abruf: 2007-03-27.

Ludewig, Lichter /Software Engineering/

Jochen Ludewig, Horst Lichter: Software Engineering: Grundlagen, Menschen, Prozesse, Techniken. Heidelberg 2007.

Marks, Bell /Planning and Implementation/

Eric A. Marks, Michael Bell: Service-Oriented Architecture: A Planning and Implementation Guide for Business and Technology. Hoboken 2006.

McGovern u. a. /Java Web Services/

James McGovern, Sameer Tyagi, Michael E. Stevens, Sunil Mathew: Java Web Services Architecture. San Francisco u. a. 2003.

Melvin /Axiomatic System Design/

Jason W. Melvin: Axiomatic System Design: Chemical Mechanical Polishing Machine Case Study. Dissertation vorgelegt am Department of Mechanical Engineering des Massachusetts Institute of Technology. o. O. 2003.

Melvin, Suh /Rearrangement/

Jason W. Melvin, Nam P. Suh: Beyond the Hierarchy: System-wide Rearrangement as a Tool to Eliminate Iteration. In: Proceedings of ICAD2002, Second International Conference on Axiomatic Design. Cambridge 2002.

Meyers, Binkley /Slice-Based Cohesion/

Timothy M. Meyers, David Binkley: Slice-Based Cohesion Metrics and Software Intervention. http://www.cs.loyola.edu/~binkley/papers/wcre04-metrics.ps, Baltimore 2004, Abruf: 2007-06-23.

OASIS /Reference Model/

OASIS (Hrsg.): Reference Model for Service Oriented Architecture 1.0. http://www.oasis-open.org/committees/download.php/19679/soa-rm-cs.pdf, o. O. 2006, Abruf: 2007-04-03.

OMG /UML/

Object Management Group (Hrsg.): Unified Modeling Language: Superstructure. Version 2.1.1. o. O. 2007.

OMG /UML Profile Services/

Object Management Group (Hrsg.): UML Profile and Metamodel for Services (UPMS): Request For Proposal. http://www.omg.org/cgi-bin/apps/doc?soa/06-09-09.pdf, Needham, USA 2006, Abruf: 2007-06-15.

Oracle /Tutorial/

Oracle (Hrsg.): Oracle SOA Suite: Tutorial, Release 3. http://download-uk.oracle.com/docs/cd/B31017_01/core.1013/b28937.pdf, o. O. 2006, Abruf: 2007-05-09.

Ott, Bieman /Program Slices/

Linda M. Ott, James M. Bieman: Program Slices as an Abstraction for Cohesion Measurement. http://www.cs.colostate.edu/~bieman/Pubs/istPreprint98.pdf, o. O. 1998, Abruf: 2007-06-05.

o. V. /Technology/

o. V.: Axiomatic Design Technology. http://www.axiomaticdesign.com/-technology/axiomatic.asp, o. O. 2006, Abruf: 2007-05-07.

Page-Jones /Systemdesign/

Meilir Page-Jones: Strukturiertes Systemdesign: Ein praktischer Leitfaden. München - Wien - London 1995.

Papazoglou, Yang /Design Methodology/

Mike P. Papazoglou, Jian Yang: Design Methodology for Web Services and Business Processes. In: Alejandro Buchmann, Fabio Casati, Ludger Fiege, Mei-Chun Hsu, Ming-Chien Shan (Hrsg.): Technologies for E-Services: Third International Workshop, Proceedings. Berlin u. a. 2002, S. 54-64.

Perepletchikov, Ryan, Frampton /Paradigms/

Mikhail Perepletchikov, Caspar Ryan, Keith Frampton: Comparing the Impact of Service-Oriented and Object-Oriented Paradigms on the Structural Properties of Software. In: Robert Meersman u. a. (Hrsg.): On the Move to Meaningful Internet Systems 2005: OTM Workshops. Heidelberg 2005, S. 431-441.

Pimentel, Stadzisz /Use Case/

Andrey R. Pimentel, Paulo C. Stadzisz: A Use Case Based Object-Oriented Software Design Approach Using the Axiomatic Design Theory. In: Proceedings of ICAD2006, 4th International Conference on Axiomatic Design. Firenze 2006.

Precht, Kraft, Bachmaier /Statistik/

Manfred Precht, Roland Kraft, Martin Bachmaier: Angewandte Statistik 1. 6. Aufl., München - Wien 1999.

Reldin, Sundling /Explaining Service Granularity/

Pierre Reldin, Peter Sundling: Explaining SOA Service Granularity: How IT-Strategy Shapes Services. Masterarbeit vorgelegt am Department of Management and Engineering der Linköping Universität. Stockholm 2007.

Reussner, Hasselbring /Handbuch Software-Architektur/

Ralf Reussner, Wilhelm Hasselbring (Hrsg.): Handbuch der Software-Architektur. Heidelberg 2006.

Richter, Haller, Schrey /Serviceorientierte Architektur/

Jan-Peter Richter, Harald Haller, Peter Schrey: Serviceorientierte Architektur. In: Informatik Spektrum. Band 28, Nr. 5, 2005, S. 413-416.

Riebisch /Architektur- und Komponentenentwicklung/

Matthias Riebisch: Prozess der Architektur- und Komponentenentwicklung. In: Ralf Reussner, Wilhelm Hasselbring (Hrsg.): Handbuch der Software-Architektur. Heidelberg 2006, S. 65-88.

Rommelfanger, Eickemeier /Entscheidungstheorie/

Heinrich J. Rommelfanger, Susanne H. Eickemeier: Entscheidungstheorie: Klassische Konzepte und Fuzzy-Erweiterungen. Berlin u. a. 2002.

Roock, Lippert /Refactorings/

Stefan Roock, Martin Lippert: Refactorings in großen Softwareprojekten: Komplexe Restrukturierungen erfolgreich durchführen. Heidelberg 2004.

Roth /Lösungsverfahren/

Britta Roth: Lösungsverfahren für mehrkriterielle Entscheidungsprobleme: Klassische Verfahren, Neuronale Netze und Fuzzy Logic. Frankfurt am Main u. a. 1998.

Saaty /Analytic Hierarchy Process/

Thomas L. Saaty: Decision Making for Leaders: The Analytic Hierarchy Process for Decisions in a Complex World. 3. Aufl., Pittsburgh 2001.

Schmelzer, Sesselmann /Geschäftsprozessmanagement/

Hermann J. Schmelzer, Wolfgang Sesselmann: Geschäftsprozessmanagement in der Praxis: Kunden zufrieden stellen – Produktivität steigern – Wert erhöhen. 5. Aufl., München - Wien 2006.

Schumann, Gerisch /Software-Entwurf/

Jörg Schumann, Manfred Gerisch: Software-Entwurf: Prinzipien, Methoden, Arbeitsschritte, Rechnerunterstützung. Berlin 1984.

Stevens, Myers, Constantine /Structured Design/

Wayne P. Stevens, Glenford J. Myers, Larry L. Constantine: Structured Design. In: IBM Systems Journal. Band 13, Nr. 2, 1974, S. 115-139.

Suh /Axiomatic Design/

Nam P. Suh: Axiomatic Design: Advances and Applications. New York - Oxford 2001.

Suh /Principles/

Nam P. Suh: The Principles of Design. New York - Oxford 1990.

Szyperski /Component Software/

Clemens Szyperski: Component Software: Beyond Object-Oriented Programming. 2. Aufl., London u. a. 2002.

Tittmann /Graphentheorie/

Peter Tittmann: Graphentheorie: Eine anwendungsorientierte Einführung. München - Wien 2003.

Vogel u. a. /Software-Architektur/

Oliver Vogel, Ingo Arnold, Arif Chughtai, Edmund Ihler, Uwe Mehlig, Thomas Neumann, Markus Völter, Uwe Zdun: Software-Architektur: Grundlagen – Konzepte – Praxis. München 2005.

Wanner, Jäger /Wiederverwendung/

Gerhard Wanner, Klaus-Dieter Jäger: Wiederverwendung von Softwarekomponenten durch entkoppelte Schnittstellen. In: OBJEKTspektrum. Nr. 1, 1999, S. 30-33.

Weber /Mehrkriterielle Entscheidungen/

Karl Weber: Mehrkriterielle Entscheidungen. München - Wien 1993.

Wermke, Kunkel-Razum, Scholze-Stubenrecht /Fremdwörterbuch/

Matthias Wermke, Kathrin Kunkel-Razum, Werner Scholze-Stubenrecht (Hrsg.): Duden Fremdwörterbuch (Band 5). 8. Aufl., Mannheim u. a. 2005.

Whitehead /Component-Based Development/

Katharine Whitehead: Component-Based Development: Principles and Planning for Business Systems. London u. a. 2002.

Winter, Schelp /Dienstorientierte Architekturgestaltung/

Robert Winter, Joachim Schelp: Dienstorientierte Architekturgestaltung auf unterschiedlichen Abstraktionsebenen. In: Ralf Reussner, Wilhelm Hasselbring (Hrsg.): Handbuch der Software-Architektur. Heidelberg 2006, S. 229-242.

Anhang A Strukturierung und Architekturprinzipien von SOA

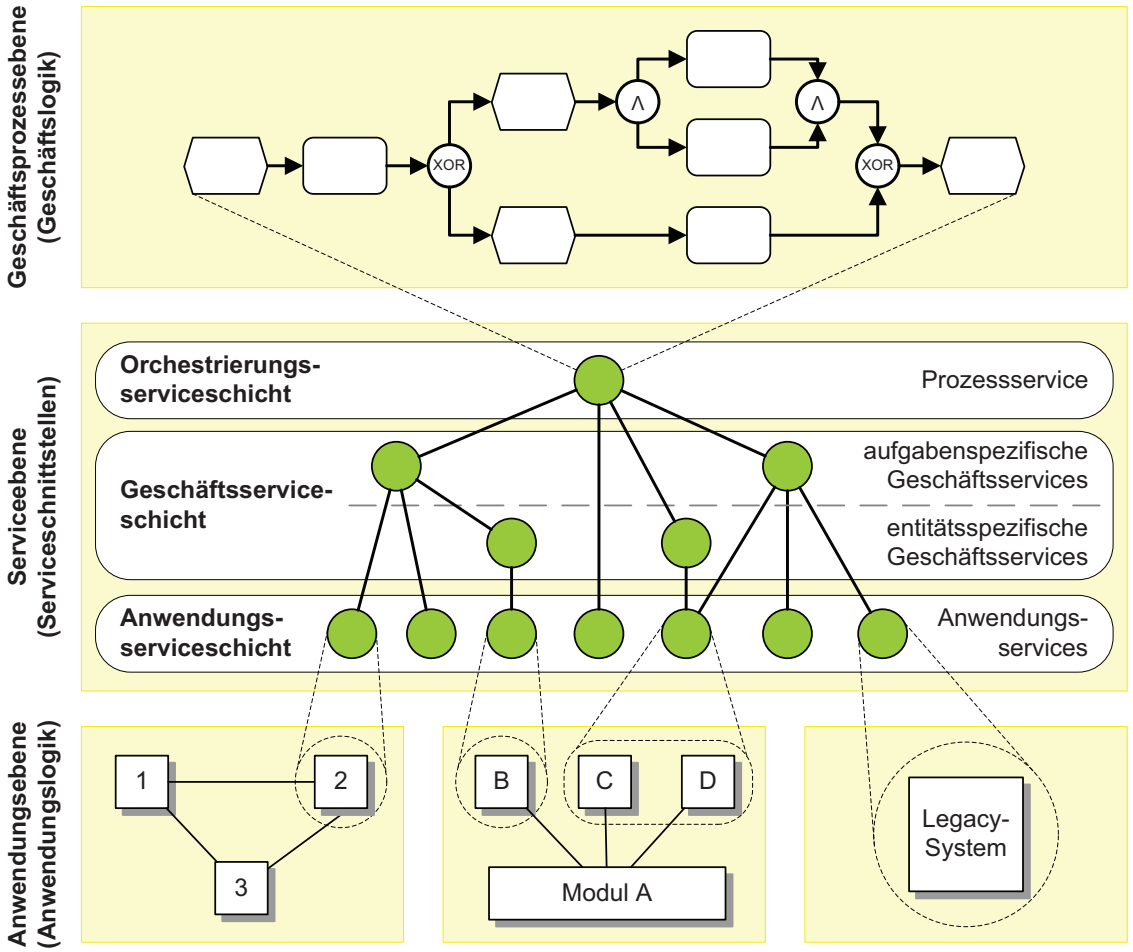


Bild A-1: Einordnung und Detaillierung der Serviceebene[340]

[340] Quelle: Erl /Service-Oriented Architecture/ 283 und 337

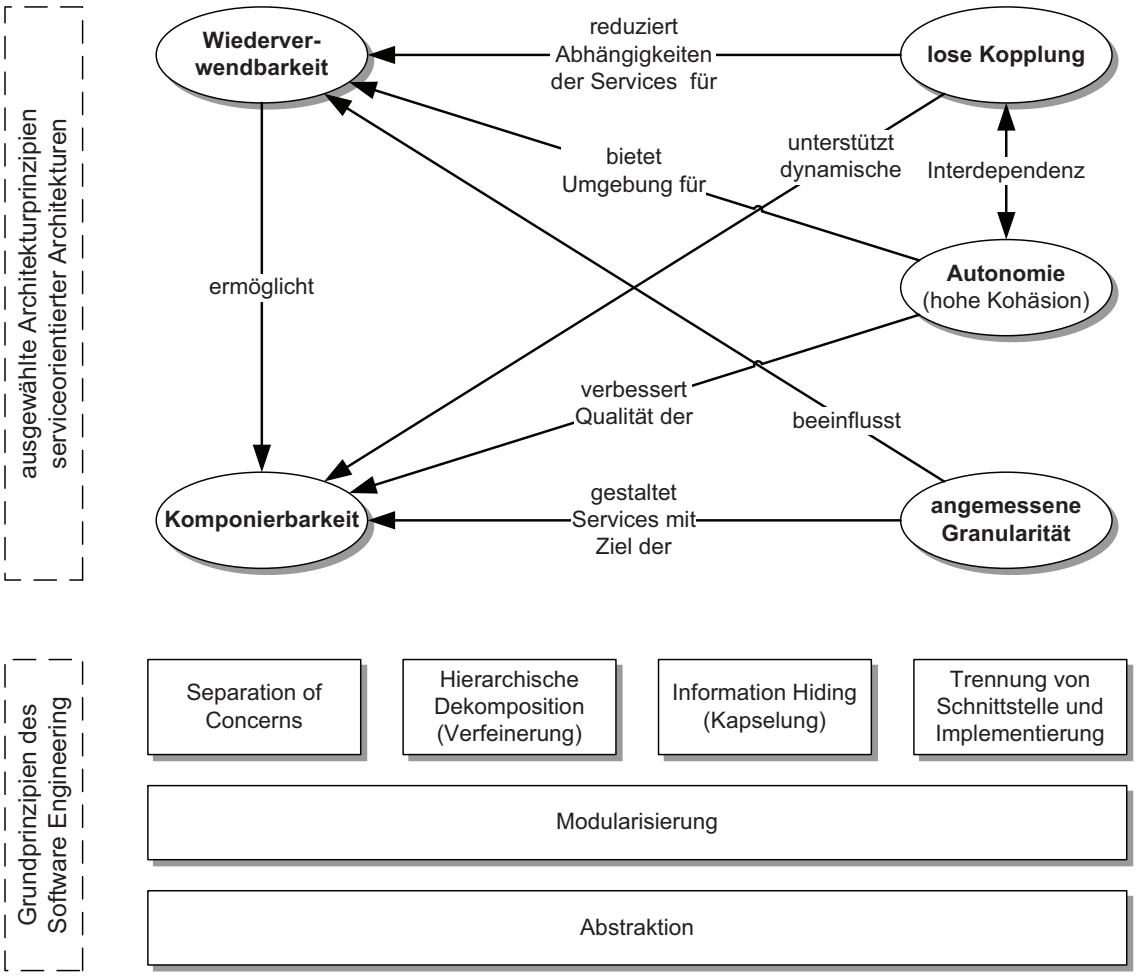


Bild A-2: Beziehungen der Architekturprinzipien der Serviceorientierung[341]

[341] Quelle: Erl /Service-Oriented Architecture/ 313-321

Anhang B Axiomatic Design im Entwurf serviceorientierter Architekturen

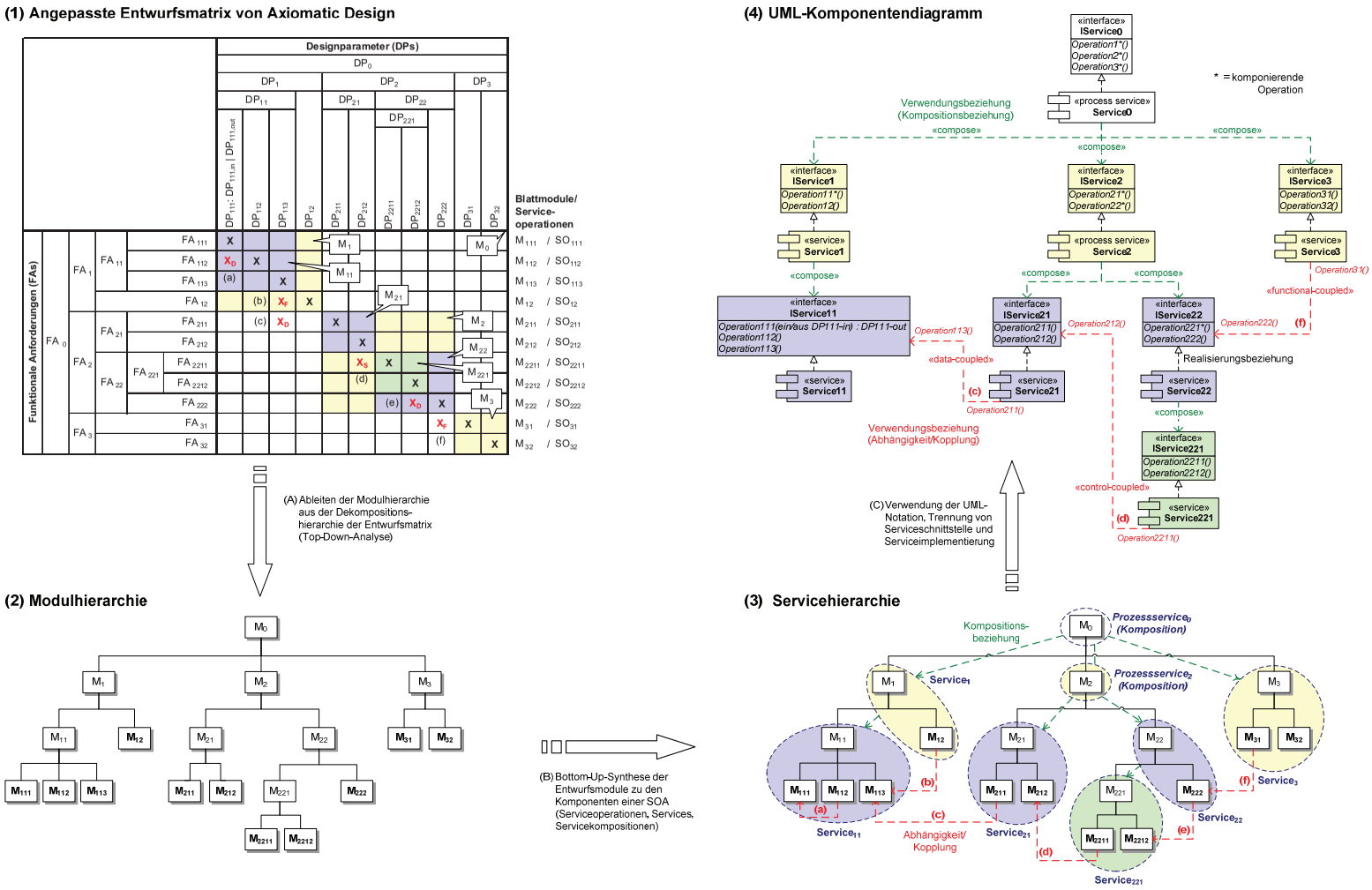


Bild B-1: Ableitung des SOA-Entwurfs aus der Entwurfsmatrix

Architekturbestandteile	Entwurfsunterstützung durch Axiomatic Design	Entsprechungen in einer SOA	Darstellung mittels UML
Gesamtarchitektur └ Systemkomponenten └ Systemmodule └ Modulschnittstelle	Gesamtentwurf (vollst. Entwurfsmatrix) └ Gruppierung von Modulen └ Entwurfsmodule └ Designparameter (DP_{in}, DP_{out})	Serviceorientierte Architektur └ Services, Kompositionen └ Serviceoperationen └ Operationsschnittstelle	Komponentendiagramm └ Komponenten («service») └ Interface-Methoden └ Methodenparameter
Beziehungen zwischen Systemkomponenten	Kompositionsbeziehungen (Dekompositionsstruktur der Modulhierarchie)	Servicekompositionen (Struktur der Servicehierarchie)	Verwendungsbeziehungen mit Stereotyp «compose»
	Kopplungsbeziehungen/Abhängigkeiten (Nichtdiagonalelemente der Entwurfsmatrix)	Abhängigkeiten zwischen Services	Verwendungsbeziehungen mit Stereotypen
	- Funktionskopplung - Datenkopplung - Steuerungskopplung	- funktionale Abhängigkeit - Datenabhängigkeit - Steuerungsabhängigkeit	- «functional-coupled» - «data-coupled» - «control-coupled»
Dynamische Interaktionen zwischen Systemkomponenten	Ableitung aus Prozessbeschreibung und Umsetzung der Kompositions- und Kopplungsbeziehungen	Serviceinteraktionen durch Nachrichtenaustausch	Kommunikationsdiagramm mit Nachrichtenreihenfolge und aufgelösten Serviceabhängigkeiten
Architekturprinzipien	Unabhängigkeits- und Informationsaxiom	Architekturprinzipien der Serviceorientierung (lose Kopplung, hohe Autonomie bzw. Kohäsion, angemessene Servicegranularität)	

Tabelle B-1: Zusammenhänge Architektur, Axiomatic Design, SOA und UML

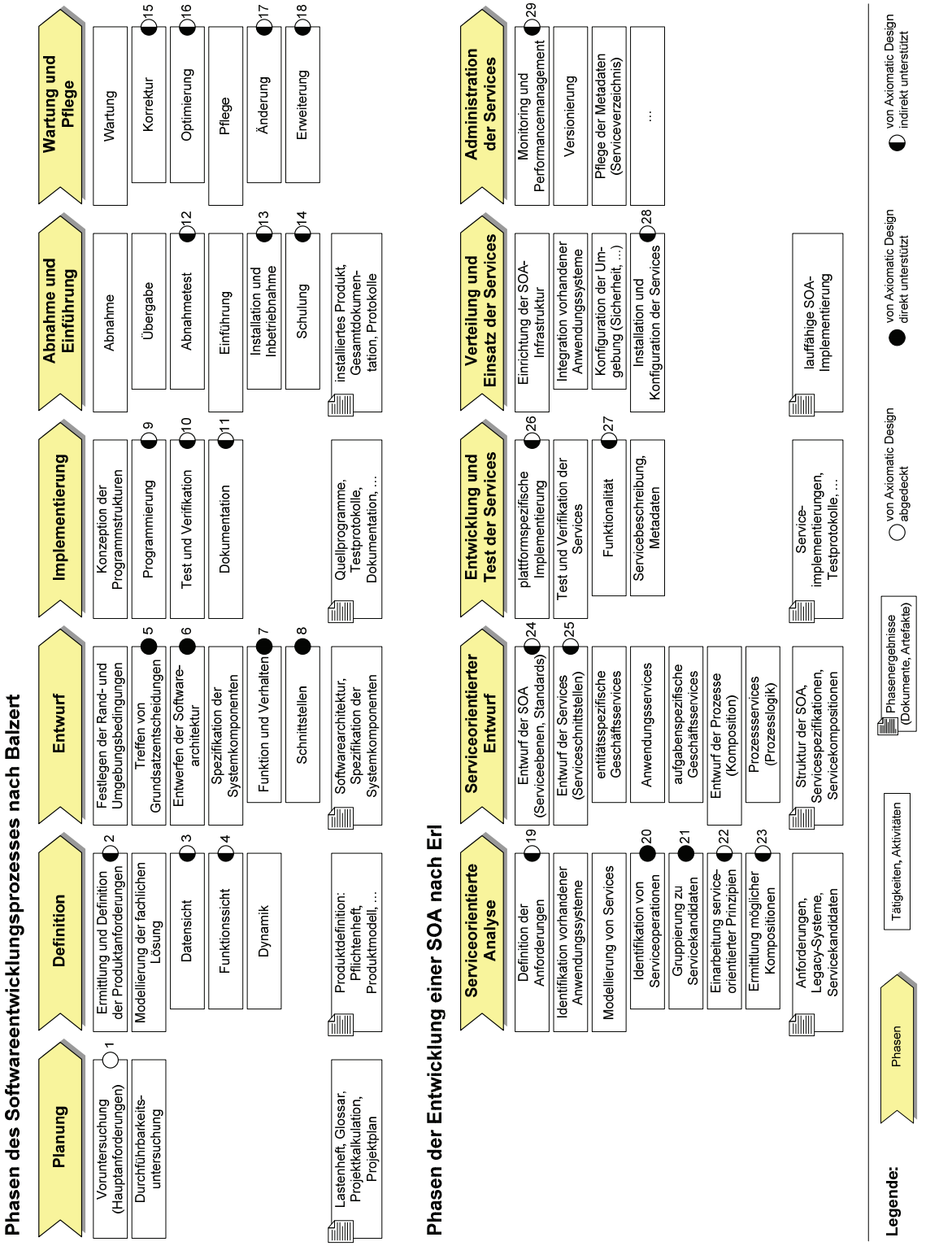


Bild B-2: Unterstützungspotenzial von Axiomatic Design

Quellen der Abbildung:

Für den Softwareentwicklungsprozess vgl. Balzert /Software-Entwicklung/ 58-60 (Planungsphase); 98 (Definitionsphase); 686 und 698 f. (Entwurfsphase); 1064 f. (Implementierungsphase); 1086-1088 (Abnahme- und Einführungsphase); 1090-1093 (Wartungs- und Pflegephase).
Für die Phasen der SOA-Entwicklung vgl. Erl /Service-Oriented Architecture/ 359 (Phasen); 379 und 399 (Serviceanalyse und -modellierung); 450, 477, 502, 523, 541, 587 (Serviceentwurf); 360-362 (Entwicklung, Test, Verteilung, Einsatz und Administration der Services).

Erläuterungen zur Abbildung:

Die Tätigkeiten in den einzelnen Phasen der Software- bzw. SOA-Entwicklung, die durch Axiomatic Design abgedeckt oder unterstützt werden, sind mit entsprechenden Symbolen versehen. Tätigkeiten die lediglich von AD abgedeckt werden, sind zwar im Entwurfsprozess berücksichtigt, werden aber nicht direkt durch Konstrukte oder Hilfsmittel von AD adressiert. Tätigkeiten, die direkt unterstützt werden, sind wesentliche Bestandteile des Entwurfs mit Axiomatic Design. Sie beruhen auf entsprechenden Entwurfsschritten, Hilfsmitteln oder Axiomen. Tätigkeiten, die nur indirekt unterstützt werden, basieren oft auf den Ergebnissen des Entwurfs mit Axiomatic Design. Jede Einschätzung des Unterstützungspotenzials ist mit einer Nummer versehen, die auf eine kurze Begründung in der folgenden Tabelle verweist.

Nr.	Unterstützungspotenzial von Axiomatic Design
1	Ermittlung der Kundenanforderungen erfolgt in der Kundendomäne.[342] Keine Unterstützung durch konkrete AD-Hilfsmittel.
2	Entspricht der Ableitung der funktionalen Anforderungen an den Entwurf aus den Kundenanforderungen.[343]
3, 4	Die Dekompositionshierarchien der FAs und DPs sowie die vollständige wurfsmatrix spezifizieren die angestrebte Lösung/den Entwurf.[344]
5	Jede Zuordnung eines Designparameters zu einer Anforderung ist eine

342 Vgl. Suh /Axiomatic Design/ 14
343 Vgl. Suh /Axiomatic Design/ 14-16
344 Vgl. Melvin /Axiomatic System Design/ 49

	wurfsentscheidung. Die Entscheidungen auf den oberen Hierarchieebenen schränken den Entwurfsspielraum auf den unteren Ebenen ein.[345]
6	Die Gesamtarchitektur (Systemkomponenten und ihre Beziehungen) lässt sich aus der vollständigen Entwurfsmatrix und der Modulhierarchie ableiten.[346]
7	Systemkomponenten ergeben sich aus den Modulen des Entwurfs und deren Synthese zu Subsystemen. Ihre Funktionen sind durch die jeweiligen funktionalen Anforderungen beschrieben. Die erforderlichen Eigenschaften sind durch die Designparameter und Restriktionen definiert.
8	Die Schnittstellen lassen sich aus den Designparametern und Beziehungen innerhalb der vollständigen Entwurfsmatrix ableiten.
9	Die Aufgabenteilung und -zuweisung während der Programmierung wird durch das Flussdiagramm (Flow Diagram) des Entwurfs vorgegeben.[347]
10, 12	Das Testen der Komponenten wird durch die explizit formulierten funktionalen Anforderungen, Restriktionen und Zielspannen unterstützt. Diese können als Testvorgaben genutzt werden. Die Ermittlung der Systemspanne kann im Rahmen der Testaktivitäten erfolgen.
11	Die während des Entwurfs mit Axiomatic Design erstellten Dokumente nen als Grundlage für die Systemdokumentation genutzt werden tion der Systemarchitektur durch Entwurfsgleichungen, Entwurfsmatrizen und Diagramme[348]; Beschreibung der Entwurfsentscheidungen). AD enthält jedoch keine konkreten Vorschriften oder Vorlagen für die Gestaltung dieser Dokumente.
13	Das Flussdiagramm bietet eine Grundlage für die Reihenfolge der Integration von Systemkomponenten zum Gesamtsystem.[349]
14	Die Entwurfsdokumente von AD können auch für Schulungszwecke eingesetzt werden.
15, 16	Zur Fehlerdiagnose können das Flussdiagramm und die vollständige wurfsmatrix herangezogen werden.[350]
17, 18	Die Analyse und Verfolgung der Auswirkungen von Änderungen am Entwurf

345 Vgl. Suh /Axiomatic Design/ 21 f.
346 Vgl. Melvin /Axiomatic System Design/ 43
347 Vgl. Suh /Axiomatic Design/ 249
348 Vgl. Suh /Axiomatic Design/ 209-213 (Module-Junction Diagram und Flow Diagram)
349 Vgl. Suh /Axiomatic Design/ 249
350 Vgl. Suh /Axiomatic Design/ 249

	können durch das Flussdiagramm und die vollständige Entwurfsmatrix tützt werden.[351] Hierdurch können die von Änderungen beeinflussten Module lokalisiert werden.
19	Vgl. Anmerkung 2
20	Kandidaten für Serviceoperationen lassen sich aus den Modulen der Entwurfsmatrix ableiten.
21	Die Gruppierung der Serviceoperationen zu Servicekandidaten kann anhand der Modulhierarchie oder aufgrund der Analyse der Kopplungsbeziehungen zwischen den Operationen erfolgen.
22	Die Einhaltung bestimmter Architekturprinzipien der Serviceorientierung wird durch Axiomatic Design unterstützt (z. B. lose Kopplung, hohe Autonomie bzw. Kohäsion und angemessene Servicegranularität).
23	Anhand der Beziehungen zwischen den Serviceoperationen und Services aus der vollständigen Entwurfsmatrix und der Modulhierarchie können Abhängigkeiten ermittelt werden. Diese bieten erste Ansätze für Servicekompositionen.
24	Für die Gestaltung der Servicehierarchie kann die Modulhierarchie des Entwurfs herangezogen werden.
25	Die Serviceschnittstellen sind bereits grob durch die korrespondierenden Designparameter vorgegeben. Allerdings sind erhebliche Verfeinerungen, insbesondere der genauen Datenstrukturen, notwendig.
26	Vgl. Anmerkung 9
27	Vgl. Anmerkungen 10, 12
28	Vgl. Anmerkung 13
29	Restriktionen und Zielspannen aus Axiomatic Design definieren Vorgaben für das Monitoring und Management der Services.

Tabelle B-2: Unterstützung der Software- und SOA-Entwicklung durch AD

[351] Vgl. Suh /Axiomatic Design/ 249

Anhang C Daten und Darstellungen des Fallbeispiels

Anhang C.1: Prozessdokumentation

Teilprozess	**(1) Empfange den Kundenauftrag**
Prozessziel	Die Daten des neuen Auftrags sind zu erfassen, zu verarbeiten, ggf. zu ergänzen und abschließend zu speichern.
Vorbedingungen	Es ist ein neuer Kundenauftrag eingetroffen.
Nachbedingungen	Die Daten des Kundenauftrags sind vollständig erfasst und wurden gespeichert.
Normalablauf	1. Für den neuen Auftrag wird eine eindeutige Auftragsnummer generiert. 2. Der Auftragsstatus wird gesetzt (z. B. „ungeprüft“). 3. Es wird geprüft, ob die mit dem Auftrag übermittelten Kundendaten bereits in der Kundenverwaltung vorhanden sind. a. Sind die Daten vorhanden und vollständig, ist keine Aktualisierung notwendig. b. Sind die Daten nicht vorhanden, wird ein neuer Kundendatensatz angelegt. c. Sind die Daten vorhanden, aber unvollständig, so werden sie ergänzt bzw. aktualisiert. 4. Die Positionen des Auftrags werden gelesen und verarbeitet. 5. Alle Auftragsdaten werden gespeichert.
Sonderfälle	keine

Tabelle C-1: Beschreibung des Teilprozesses „Empfange den Kundenauftrag“

Teilprozess	**(2) Prüfe den Kundenauftrag**
Prozessziel	Die Auftragsdaten sind auf ihre Plausibilität zu prüfen und die Kreditwürdigkeit des Kunden ist zu beurteilen.
Vorbedingungen	Die Auftragsdaten liegen vollständig und gespeichert vor.
Nachbedingungen	A. Die Prüfung wurde erfolgreich abgeschlossen und der Auftrag wurde genehmigt. B. Bei der Prüfung wurden Fehler festgestellt oder der Auftrag wurde nicht genehmigt.
Normalablauf	1. Die Auftragsdaten werden auf ihre Plausibilität geprüft. 2. Der Typ und die Nummer der Kreditkarte werden geprüft.

	3. Anhand des Kundenstatus und der Höhe der Auftragssumme wird die Form der Genehmigung für den Kundenauftrag ermittelt. a. Automatische Genehmigung des Auftrags b. Manuelle Genehmigung des Auftrags durch Anstoß eines Workflows
Sonderfälle	Wurde der Auftrag nicht genehmigt oder wurden bei seiner Prüfung wesentliche Fehler festgestellt (Nachbedingung B), kann die Bearbeitung nicht weiter fortgesetzt werden. Die Bearbeitung wird dann mit Teilprozess (4) beendet.

Tabelle C-2: Beschreibung des Teilprozesses „Prüfe den Kundenauftrag“

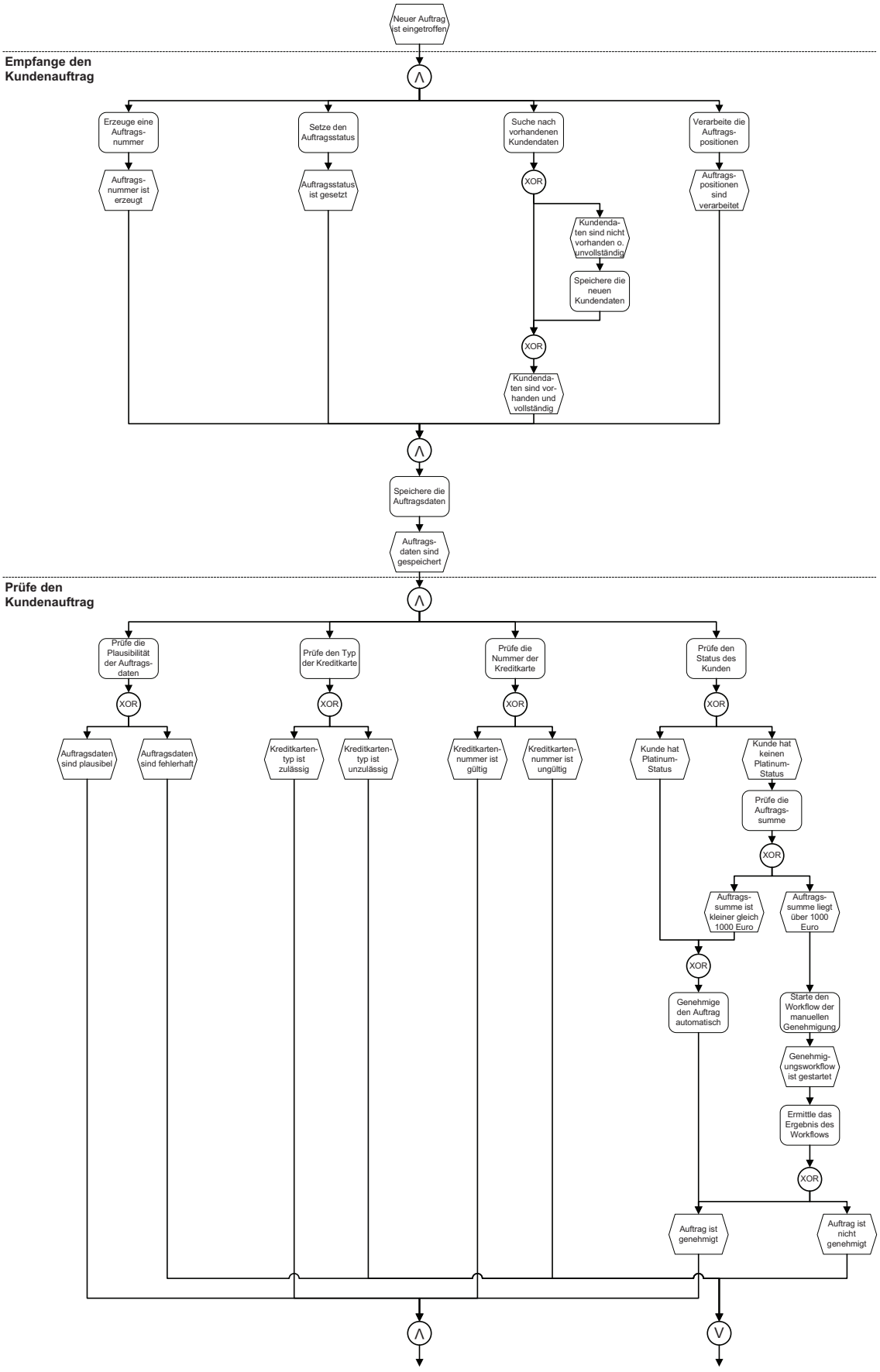
Teilprozess	**(3) Erfülle den Kundenauftrag**
Prozessziel	Für den Auftrag sind ein geeigneter Lieferant und ein Logistik-dienstleister auszuwählen und zu beauftragen.
Vorbedingungen	Die Auftragsdaten wurden erfolgreich geprüft und der Auftrag wurde genehmigt.
Nachbedingungen	Für die Lieferung der Auftragspositionen wurde ein Lieferant ausgewählt und beauftragt. Für die Zustellung der Lieferung wurde ein Logistikdienstleister ausgewählt und beauftragt.
Normalablauf	1. Für die Positionen des Kundenauftrags wird jeweils der eigene Lagerbestand ermittelt und die Bereitstellungskosten werden berechnet. 2. Für die Positionen des Kundenauftrags wird eine Angebotsanfrage an potenzielle Lieferanten erstellt und versandt. Anschließend werden die eintreffenden Angebote empfangen. 3. Anhand der Kostendaten wird zwischen der Bereitstellung aus dem eigenen Lager und der Fremdlieferung durch einen externen Lieferanten entschieden. 4. Der Lieferauftrag (intern oder extern) wird erteilt. 5. Bei Selbstbereitstellung ist der neue Lagerbestand auf die Unterschreitung des Bestellpunkts zu prüfen und ggf. eine Lagerauffüllung anzustoßen. 6. Anhand der Höhe der Auftragssumme sowie Ausgangs- und Zielort der Lieferung wird ein geeignetes

	nehmen ausgewählt. 7. Das Logistikunternehmen wird mit der Zustellung beauftragt.
Sonderfälle	keine

Tabelle C-3: Beschreibung des Teilprozesses „Erfülle den Kundenauftrag“

Teilprozess	**(4) Schließe den Kundenauftrag ab**
Prozessziel	Der Kunde ist über das Ergebnis der Auftragsbearbeitung zu informieren. Die Auftragsdaten sind bei erfolgreichem Abschluss zu verbuchen.
Vorbedingungen	A. Der Auftrag wurde vollständig bearbeitet (Lieferung und Zustellung wurden beauftragt). B. Bei der Auftragsprüfung wurden Fehler festgestellt oder der Auftrag wurde nicht genehmigt.
Nachbedingungen	Der Kunde wurde über die erfolgreiche Bearbeitung sowie den voraussichtlichen Zustellungstermin informiert. Die Auftragsdaten wurden verbucht. Der Kunde wurde über die Ablehnung des Auftrags und deren Gründe benachrichtigt.
Normalablauf	1. Der Kunde wird über das Ergebnis der Auftragsbearbeitung (erfolgreich oder abgelehnt) benachrichtigt. 2. Die Auftragsdaten werden bei erfolgreicher Bearbeitung (Vorbedingung A) im entsprechenden System verbucht.
Sonderfälle	keine

Tabelle C-4: Beschreibung des Teilprozesses „Schließe den Kundenauftrag ab“


Neuer Auftrag ist eingetroffen
Empfange den Kundenauftrag
Erzeuge eine Auftrags-nummer
Auftrags-nummer ist erzeugt
Setze den Auftragsstatus
Auftragsstatus ist gesetzt
Suche nach vorhandenen Kundendaten
XOR
Kundenda-ten sind nicht vorhanden o. unvollständig
Speichere die neuen Kundendaten
XOR
Kundenda-ten sind vor-handen und vollständig
Verarbeite die Auftrags-positionen
Auftrags-positionen sind verarbeitet
Speichere die Auftragsdaten
Auftrags-daten sind gespeichert
Prüfe den Kundenauftrag
Prüfe die Plausibilität der Auftrags-daten
XOR
Auftragsdaten sind plausibel
Auftragsdaten sind fehlerhaft
Prüfe den Typ der Kreditkarte
XOR
Kreditkarten-typ ist zulässig
Kreditkarten-typ ist unzulässig
Prüfe die Nummer der Kreditkarte
XOR
Kreditkarten-nummer ist gültig
Kreditkarten-nummer ist ungültig
Prüfe den Status des Kunden
XOR
Kunde hat Platinum-Status
Kunde hat keinen Platinum-Status
Prüfe die Auftrags-summe
XOR
Auftrags-summe ist kleiner gleich 1000 Euro
Auftrags-summe liegt über 1000 Euro
XOR
Genehmige den Auftrag automatisch
Starte den Workflow der manuellen Genehmigung
Genehmig-ungsworkflow ist gestartet
Ermittle das Ergebnis des Workflows
XOR
Auftrag ist genehmigt
Auftrag ist nicht genehmigt

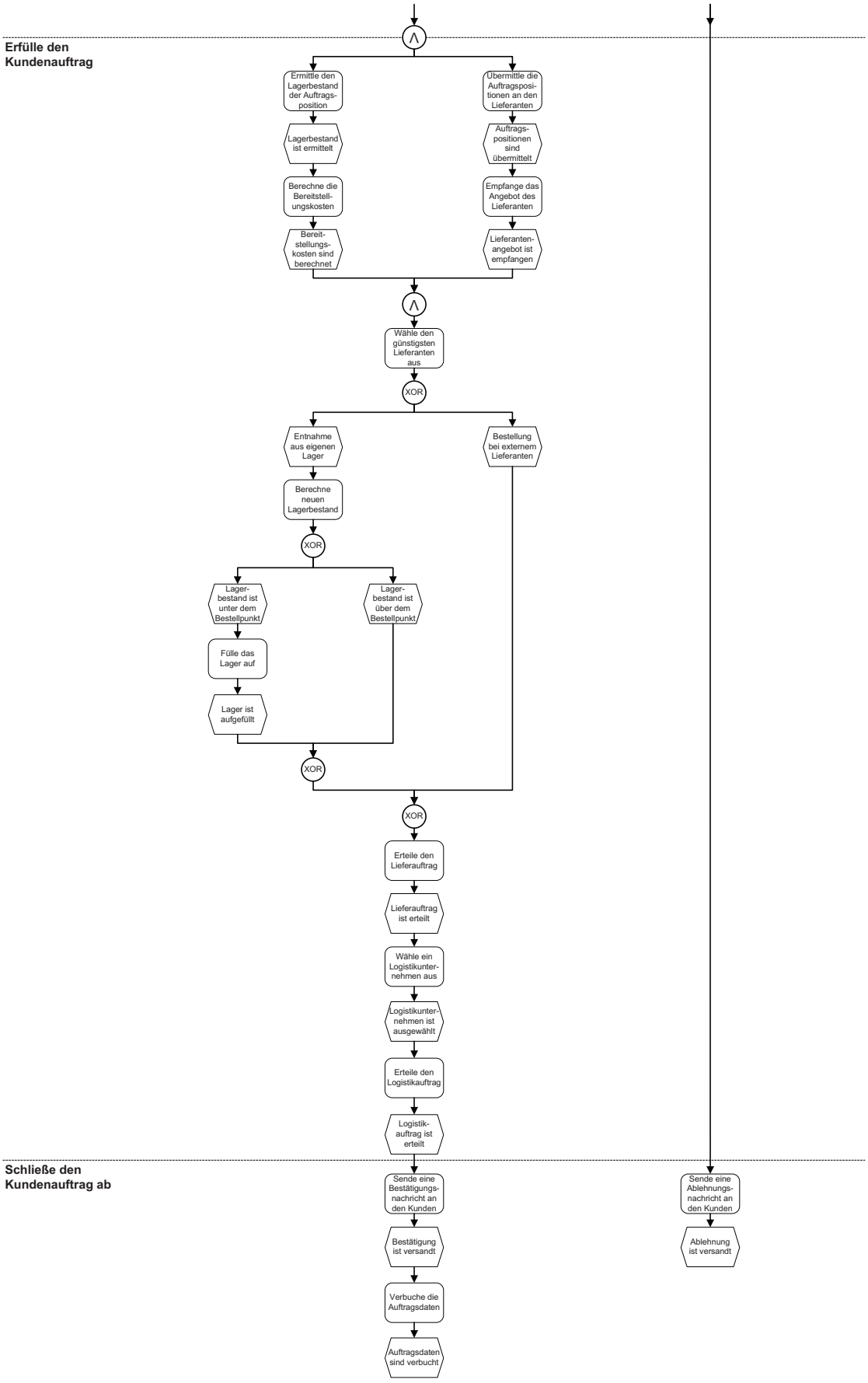


Bild C-1: EPK des Auftragsbearbeitungsprozesses

Anhang C.2: Entwurf der serviceorientierten Architektur

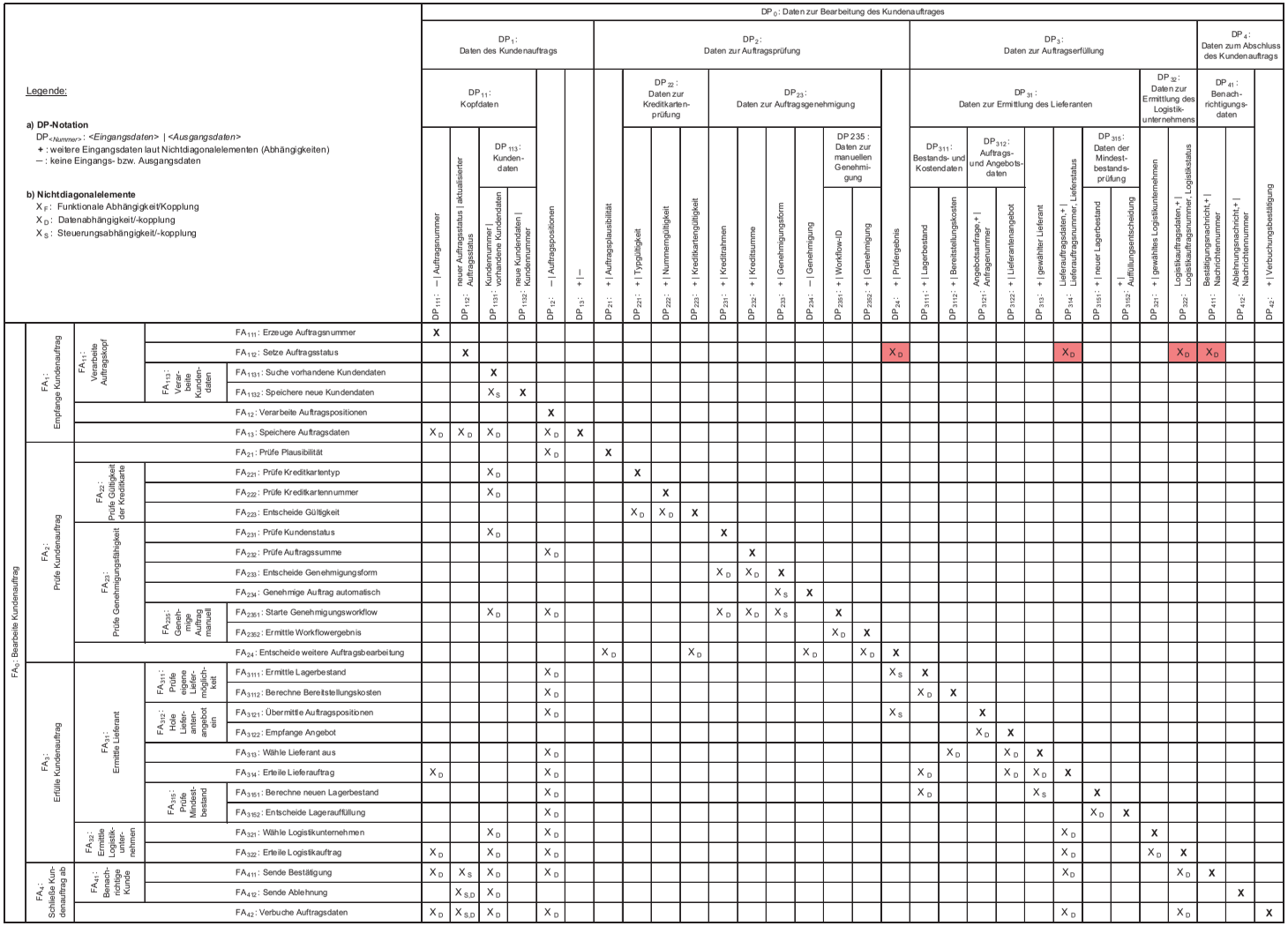


Bild C-2: Vollständige Entwurfsmatrix mit Hinkopplungen

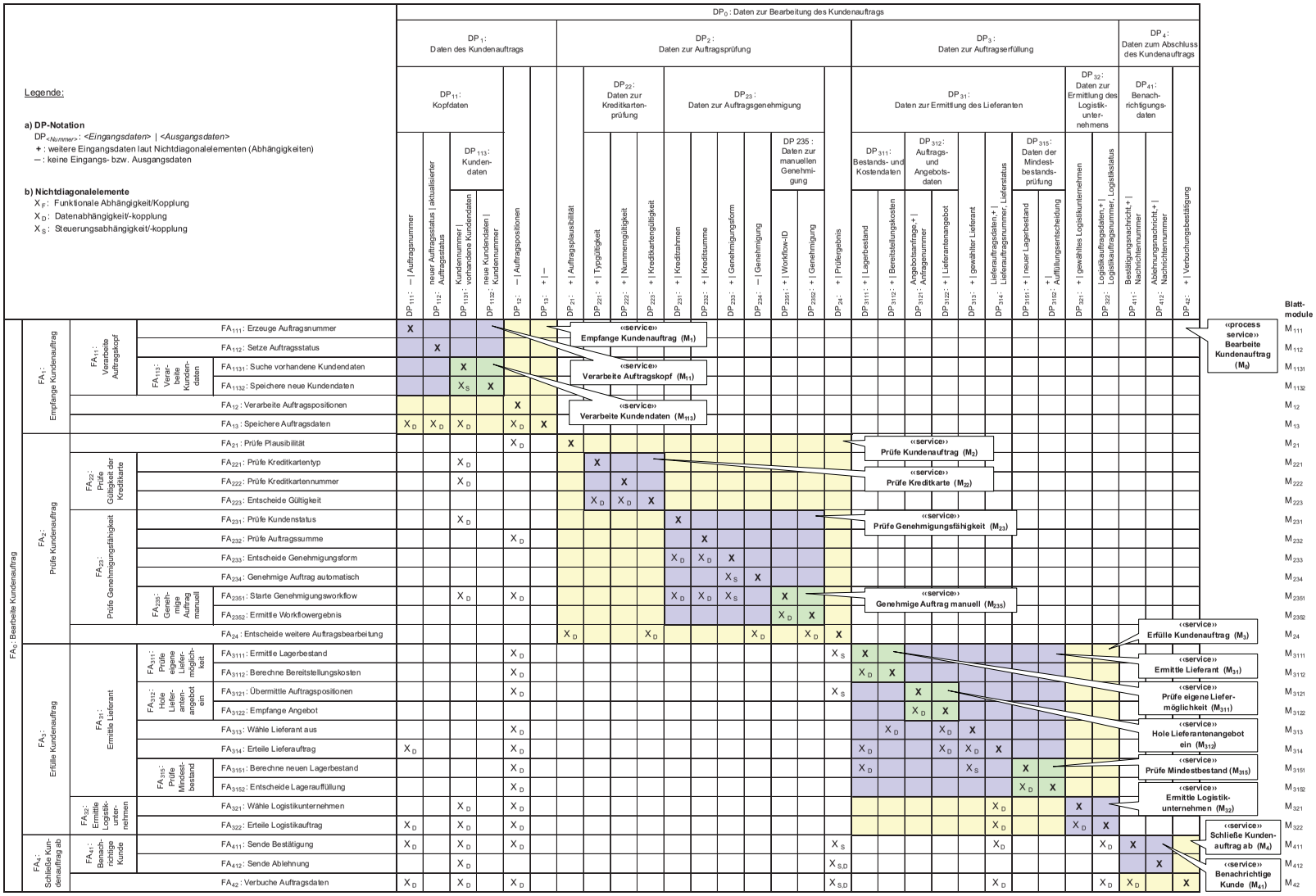


Bild C-3: Überarbeitete Entwurfsmatrix ohne Hinkopplungen

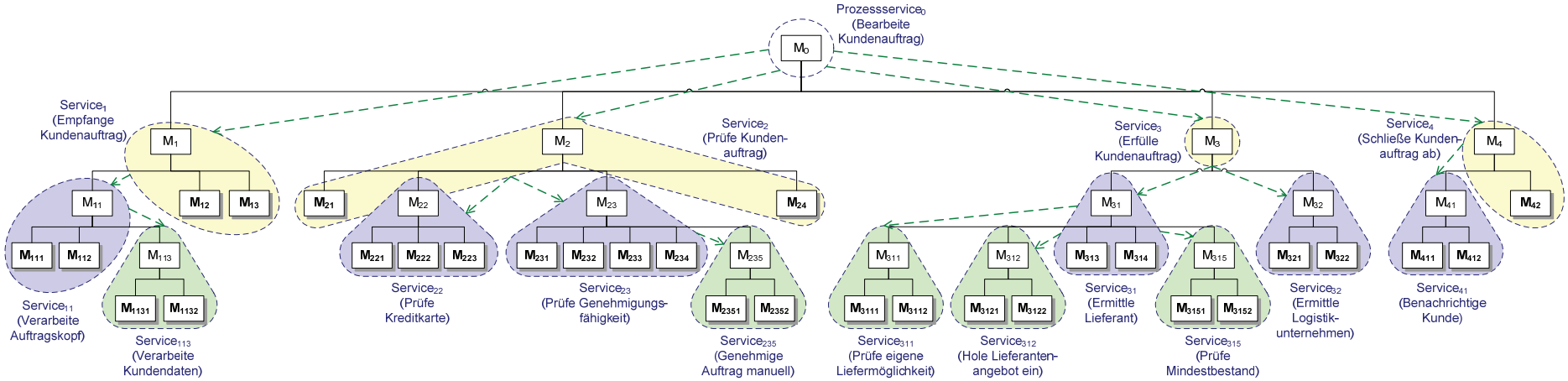


Bild C-4: Modulhierarchie und daraus abgeleitete Servicehierarchie

Anhang C.3: UML-Darstellung des SOA-Entwurfs

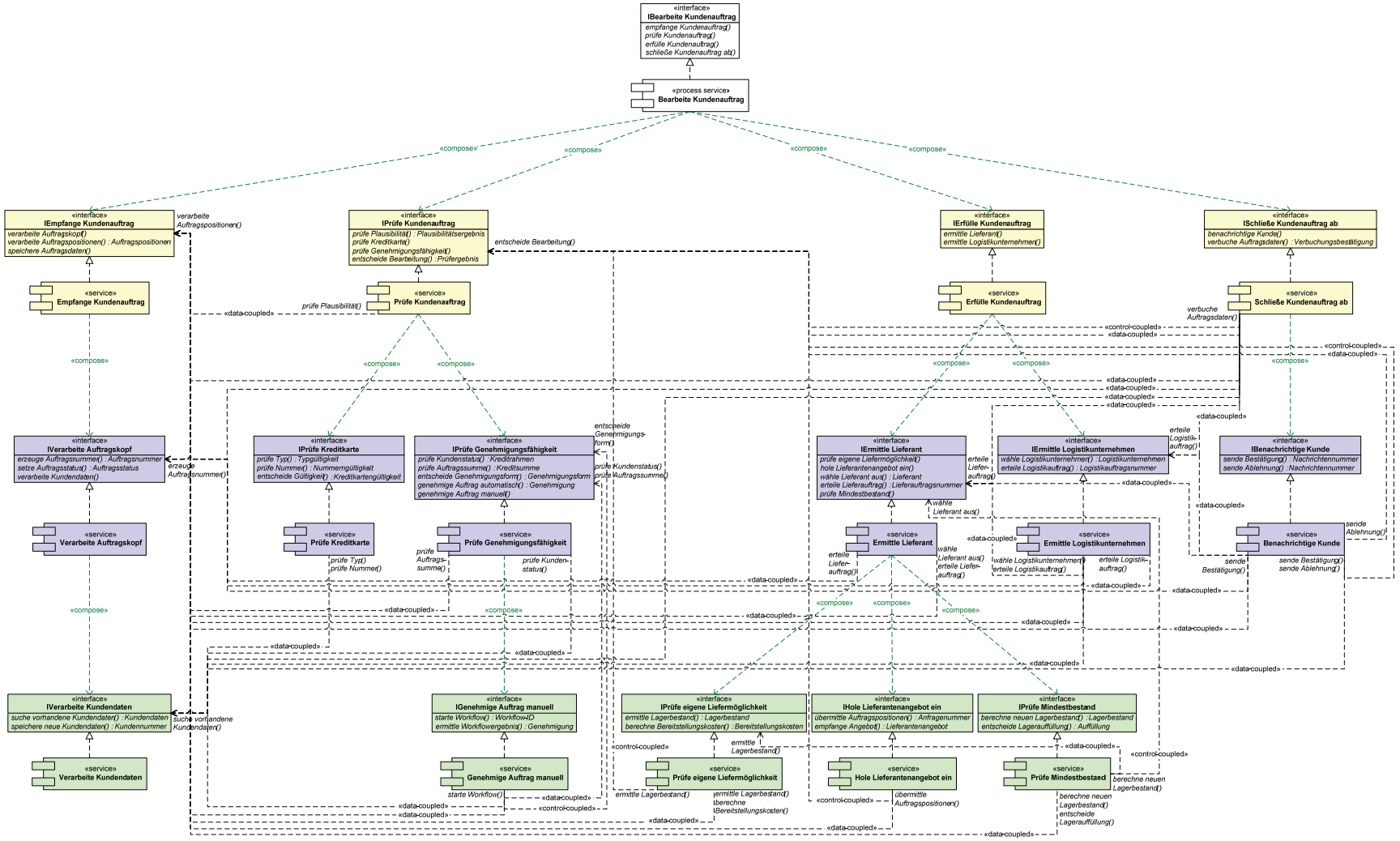


Bild C-5: UML-Komponentendiagramm des SOA-Entwurfs

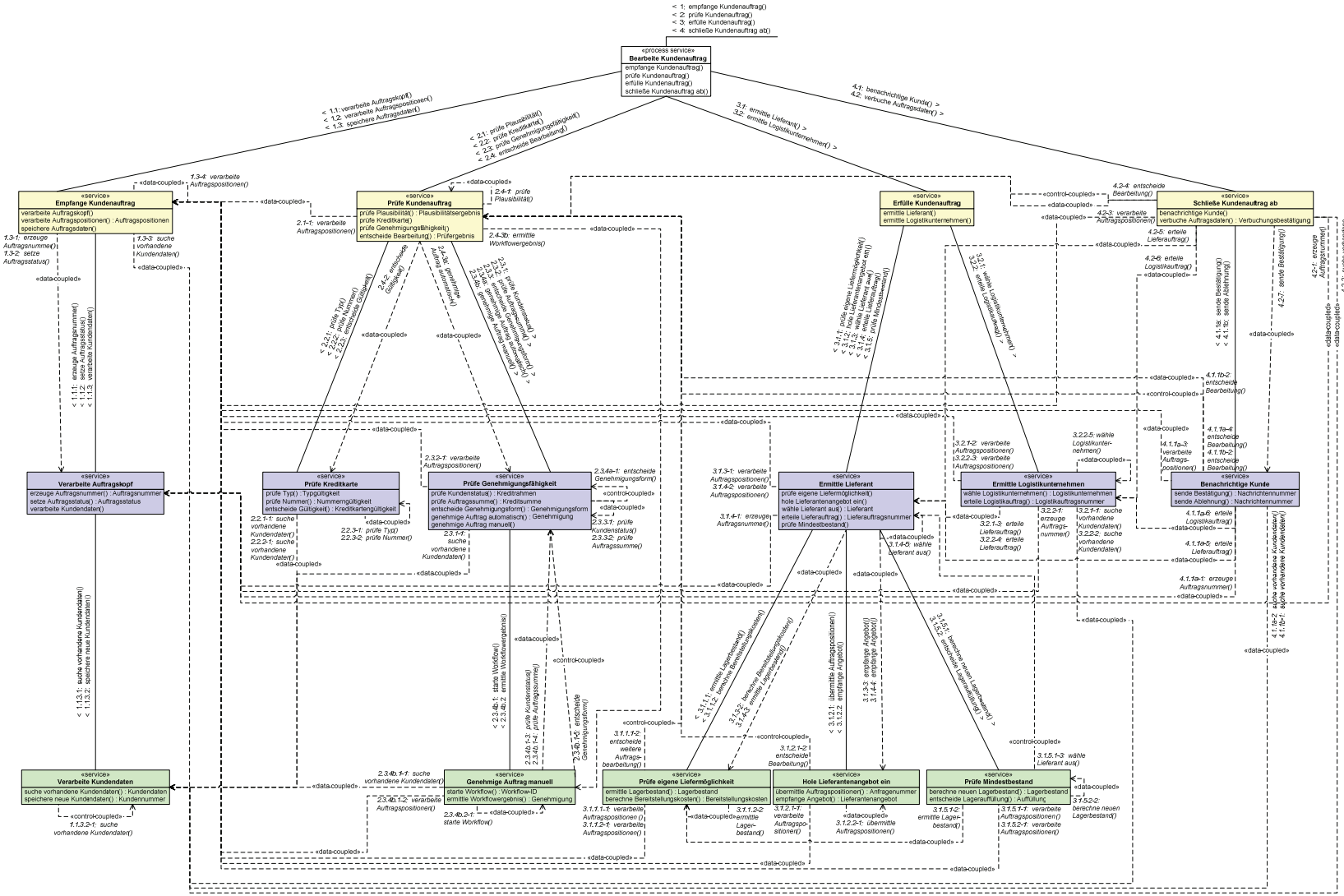


Bild C-6: UML-Kommunikationsdiagramm für die Auftragsbearbeitung

Anhang C.4: Metrikwerte des SOA-Entwurfs

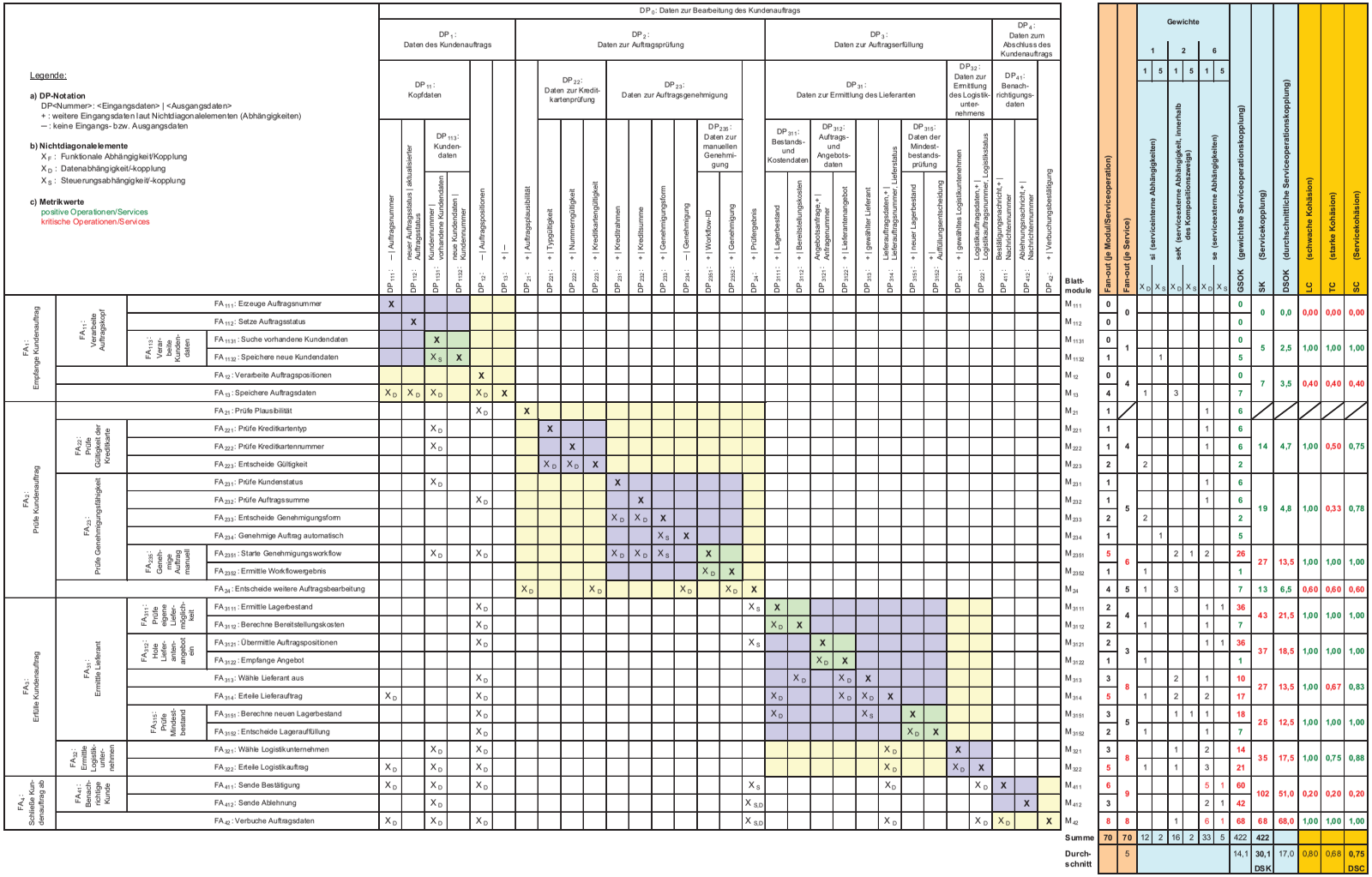


Bild C-7: Werte der Entwurfsmetriken für das Fallbeispiel