February 2007

Analyse des Beitrages von Axiomatic Design zum Entwurf Serviceorientierter Architekturen

René Fiege
Technische Universität Ilmenau, rene.fiege@tu-ilmenau.de

Dirk Stelzer
Technische Universität Ilmenau, dirk.stelzer@tu-ilmenau.de

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

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

Analyse des Beitrages von Axiomatic Design zum Entwurf Serviceorientierter Architekturen

René Fiege, Dirk Stelzer

Fachgebiet für Informations- und Wissensmanagement
Technische Universität Ilmenau
98693 Ilmenau
rene.fiege@tu-ilmenau.de, dirk.stelzer@tu-ilmenau.de

Abstract

Axiomatic Design (AD) ist eine Methode, die den Entwurf beliebiger Systeme unterstützen kann. AD hilft, Anforderungen klar voneinander abzugrenzen und es unterstützt die Entwicklung von Systemen, deren Komponenten eine überschaubare Komplexität aufweisen und weitgehend unabhängig voneinander sind. Diese Ziele des AD korrespondieren mit wesentlichen Architekturzielen für Serviceorientierte Architekturen (SOA), nämlich „ausgewogene Granularität", „lose Kopplung" und „hohe Autonomie" von Services. In diesem Papier analysieren wir, welchen Beitrag AD zum Entwurf von SOA leisten kann. Anhand eines Fallbeispiels untersuchen wir, inwiefern AD helfen kann, Services zu entwerfen, welche eine ausgewogene Granularität aufweisen und die in sich autonom und untereinander lose gekoppelt sind.

1 Einleitung

Serviceorientierte Architekturen (SOA) sollen es ermöglichen, wandlungsfähige bzw. „agile" Architekturen für Informationssysteme (IS) zu realisieren, so dass diese leicht an neue Anforderungen angepasst werden können [SiHu2005, 71 ff.; ZiTP2003]. In SOA werden so genannte Services einer Vielzahl von Teilnehmern zur Nutzung bereitgestellt. Services kapseln wiederverwendbare Funktionen. Sie sollen lose gekoppelt sein und je nach Bedarf zu beliebigen Anwendungen zusammengestellt werden können [DJMZ2005, 7 ff.].

Das SOA-Konzept ist relativ jung und befindet sich immer noch in einer Phase der Erprobung und Weiterentwicklung [Erl2005, 72]. Obwohl es bereits einige viel versprechende Modelle zur Unterstützung von Entwurf, Implementierung, Betrieb und Wartung serviceorientierter Systeme gibt [z. B. BuGa2005; EAAC2004, 83 ff.; Erl2005, 359 ff.; KoHB2005; LaMB2005; MaBe2006, 99-149], sind verschiedene Herausforderungen bisher nicht zufrieden stellend gelöst worden. Hierzu gehört unter anderem, wie Services mit einer ausgewogenen Granularität entworfen werden können und wie bereits im Entwurf darauf hingewirkt werden kann, dass Services entstehen, welche in sich möglichst autonom und untereinander lose gekoppelt sind. Die Architekturziele ausgewogene Granularität, lose Kopplung und hohe Autonomie der Services sind wichtig, um die Wiederverwendbarkeit und Komponierbarkeit der Services zu erhöhen [BuGa2005, 602; Erl2005, 290 ff.].

Axiomatic Design (AD) ist eine Methode zur strukturierten Gestaltung von Objekten[1] [Suh2001]. Urheber und Anwender von AD behaupten, dass diese Methode geeignet ist, Systeme zu entwerfen, deren Komponenten (a) eine überschaubare Komplexität aufweisen sowie (b) weitgehend unabhängig voneinander sind und dass (c) Anforderungen an das zu entwerfende System klar voneinander abgegrenzt werden können [Suh2001, 29 ff.]. Diese Ziele des AD korrespondieren mit den Architekturzielen für SOA.

Ziel dieses Beitrages ist es zu überprüfen, inwiefern AD dazu beitragen kann, die oben genannten Architekturziele beim Entwurf von SOA zu erreichen. Hierzu werden zunächst Architekturziele für SOA vorgestellt. Anschließend stellen wir Grundlagen des AD dar und demonstrieren mit Hilfe einer Fallstudie, wie AD im Rahmen des Entwurfs von SOA eingesetzt werden kann. Im Anschluss analysieren wir den Beitrag von AD für den Entwurf von SOA kritisch. Dabei legen wir insbesondere dar, in wie weit AD helfen kann, die oben genannten Architekturziele zu erreichen. Der Beitrag wird mit einer kurzen Zusammenfassung und einem Ausblick abgeschlossen.

2 Architekturziele Serviceorientierter Architekturen

Architekturziele repräsentieren Prinzipien, die eine SOA charakterisieren [Erl2005, 290]. Diese Ziele müssen bereits im Entwurf berücksichtigt werden, damit sie sich in der resultierenden

[1] Objekte können Materialien, beliebige Systeme, Software, Hardware, strategische Geschäftspläne, Organisationen und Prozesse sein.

Architektur widerspiegeln. Typische Architekturziele von SOA sind: Wiederverwendbarkeit und Komponierbarkeit sowie angemessene Granularität, lose Kopplung, hohe Autonomie, Zustandslosigkeit, Auffindbarkeit, Abstraktheit, Interoperabilität, Geschäftsorientiertheit, Nachhaltigkeit, Neutralität und wohldefinierter Servicekontrakt [BuGa2005, 602; DJMZ2005, 9; EKAP2005, 28; EMPR2005, 3-4; Erl2005, 290; MaBe2006, 39 ff.; SiHu2005, 76-77]. Wir beschränken uns in diesem Beitrag auf fünf dieser Architekturziele. Diese Ziele und ihre Zusammenhänge sind in Abb. 1 zusammengefasst.

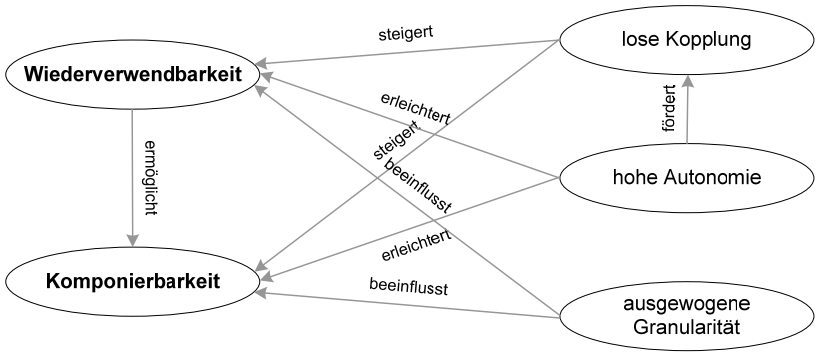

Abb. 1: Wesentliche Architekturziele Serviceorientierter Architekturen

Eines der wichtigsten Architekturziele ist die Wiederverwendbarkeit der Services. Sie sollte möglichst hoch sein, damit die SOA an Änderungen von Anforderungen ohne unangemessen hohen Entwicklungsaufwand angepasst werden kann [Erl2005, 292]. Damit unterstützt die Wiederverwendbarkeit auch die Agilität einer SOA, also die Leichtigkeit, mit der sich Änderungen oder Erweiterungen einer Architektur durchführen lassen [HaSc2006, 277].

Komponierbarkeit bedeutet, dass ein höherwertiger Service aus mehreren generischen Services zusammengesetzt werden kann. Die Logik, die in einem generischen Service gekapselt ist, kann so auf verschiedenen Granularitätsebenen wiederverwendet werden. Auch die Komponierbarkeit der Services einer SOA sollte möglichst groß sein, damit die SOA leicht an zukünftige Änderungen von Anforderungen angepasst werden kann [Erl2005, 301]. Die Wiederverwendbarkeit wirkt daher ebenfalls auf die Agilität der SOA [HaSc2006, 277].

Komponierbarkeit und Wiederverwendbarkeit hängen stark von der gewählten Granularität der Services ab [HaSc2006, 281]. Die Granularität wird bestimmt durch das Abstraktionsniveau der Aufgaben, die durch den Service unterstützt werden [Erl2005, 299]. Sie ist daher vergleichbar mit dem Prinzip der Abstraktion in der Softwareentwicklung [Balz1998, 559]. Granularität beschreibt den Umfang und die Art der Funktionen, welche durch den Service unterstützt werden [Balz1998, 559; Erl2005, 299, 302; MaBe2006, 40, 124]. Je weniger Funktionen und je

konkreter die Funktionen auf einen einzelnen Aufgabenbereich ausgerichtet sind, desto feiner ist die Granularität (und desto niedriger das Abstraktionsniveau). Die Servicegranularität sollte angemessen gewählt werden, um die Kompositions- und Wiederverwendungspotentiale zu erhöhen. Eine zu grobe Servicegranularität kann dazu führen, dass ein Service auf Grund von Performanceproblemen nicht wiederverwendet werden kann. Eine zu feine Granularität kann das Wiederverwendungspotential senken, da der Service auf einen bestimmten Aufgabenbereich zugeschnitten ist und nicht in anderen Aufgabenbereichen verwendet werden kann [MaBe2006, 40].

Lose Kopplung umfasst die Reduzierung von Abhängigkeiten zwischen Services [Balz1998, 474; CaGl1990, 31-41; Kaye2003; Raas1993, 364; VACI2005, 114]. Sie wird unter anderem dadurch erreicht, dass die Services untereinander über genau definierte Schnittstellen interagieren [Erl2005, 314]. Dadurch kann die Implementierung eines Service leicht ausgetauscht werden. Durch Reduzierung der Abhängigkeiten zwischen den Services wird außerdem das Potential zur Komposition und Wiederverwendung erhöht [MaBe2006, 41]. Daher sollte eine möglichst lose gekoppelte SOA angestrebt werden.

Serviceautonomie erfordert, dass die Logik, die innerhalb eines Service gekapselt ist, im Hinblick auf einen definierten Kontext (z. B. eine zu erfüllende Funktion) klar abgegrenzt werden kann [Erl2005, 303]. Autonomie entspricht dem Prinzip der Kohäsion in der Softwareentwicklung [Balz1998, 474; VACI2005, 117]. Eine hohe Autonomie liegt insbesondere dann vor, wenn sich die Servicelogik auf genau einen Kontext bezieht [Erl2005, 304]. Eine hohe Serviceautonomie minimiert Abhängigkeiten zwischen Services und fördert eine lose Kopplung [Balz1998; Erl2005, 303; VACI2005, 118]. Außerdem erleichtert die klare Abgrenzung und Kapselung der Servicelogik die Wiederverwendbarkeit und Komponierbarkeit von Services [Erl2005, 318]. Es sollte daher auch eine hohe Autonomie der Services angestrebt werden.

3 Grundlagen des Axiomatic Design

AD wurde Ende der 70er Jahre von Nam Pyo Suh am Massachusetts Institute of Technology (MIT) entwickelt [Suh1990, 18]. Es handelt sich dabei um eine Methode, mit der man strukturiert beliebige Objekte entwerfen kann. Das Grundprinzip von AD umfasst die strukturierte Suche und Zuordnung geeigneter Lösungen für zuvor festgelegte Anforderungen.

Ein Entwurf ist definiert als das Ergebnis dieses Zuordnungsprozesses [Suh2001, 2 ff.]. Er beschreibt, welche Anforderungen durch welche Lösung erfüllt werden können. AD basiert auf dem Konzept der Domänen sowie auf dem so genannten Unabhängigkeits- und dem Informationsaxiom. Beide Axiome formulieren Richtlinien für den Entwurfsprozess. Aus Platzgründen gehen wir in diesem Beitrag auf das Informationsaxiom nur kurz ein.

3.1 Konzept der Domänen

Das Konzept der Domänen umfasst die Kundendomäne, die funktionale und die physische Domäne sowie die Prozessdomäne (Abb. 2). Der Entwurfsprozess erstreckt sich über alle Domänen. Er beginnt in der Kundendomäne und endet in der Prozessdomäne. Jede Vorgängerdomäne beschreibt Anforderungen, jede Folgedomäne die korrespondierenden Lösungen. Zwischen allen Domänen erfolgt eine Zuordnung von Anforderungen zu korrespondierenden Lösungen [Suh2001, 10-14].

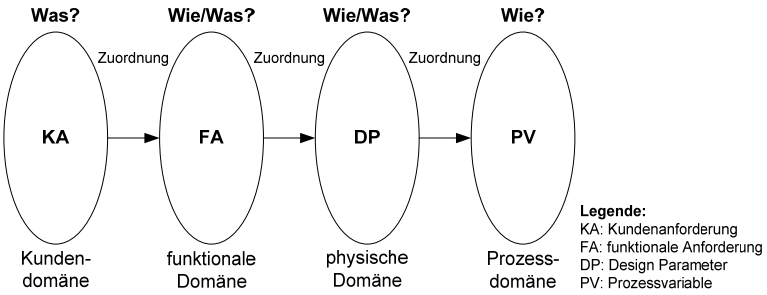

Abb. 2: Konzept der Domänen

Die Kundendomäne beinhaltet die Kundenanforderungen (KA) an das zu gestaltende Objekt. Sie beschreiben relativ grob, welche Eigenschaften das zu entwickelnde Objekt haben soll. Suh empfiehlt, die Kundenanforderungen nach ihrer Wichtigkeit zu ordnen [Suh2001, 14]. Die Kundenanforderungen werden in der funktionalen Domäne weiter zu funktionalen Anforderungen (FA) und Restriktionen (R) verfeinert. Funktionale Anforderungen beschreiben aus Sicht des Designers alle Anforderungen, die geeignet sind, die Bedürfnisse des Kunden abzudecken. Anzustreben ist die minimale Anzahl funktionaler Anforderungen, indem nur die wesentlichen Erfordernisse identifiziert werden. Alle weiteren Anforderungen werden auf tiefer liegenden Entwurfsebenen (vgl. Abschnitt 3.2) bearbeitet, da sie die Komplexität des Entwurfs erhöhen würden. Restriktionen ergänzen die funktionalen Anforderungen, indem sie den Lösungsraum einschränken. Sie repräsentieren zusätzliche Anforderungen und beschreiben Grenzen für korrespondierende Lösungen der funktionalen Anforderungen. Derartige Lösungen werden als Designparameter (DP) bezeichnet. Sie sind Inhalt der physischen Domäne. Im

Idealfall erhält jede funktionale Anforderung genau einen korrespondierenden Designparameter. Bei der Auswahl geeigneter Parameter müssen die Restriktionen berücksichtigt werden. Die Designparameter repräsentieren ihrerseits wieder Anforderungen für die Folgedomäne. Die Prozessdomäne umfasst die Prozessvariablen (PV). Hierbei handelt es sich um alle Hilfsmittel zur Erzeugung der Designparameter. Sie charakterisieren den Herstellungsprozess der Parameter. Idealerweise wird jedem Designparameter genau eine Prozessvariable zugeordnet.

3.2 Unabhängigkeitsaxiom

Während der Zuordnung zwischen den Domänen wird das Unabhängigkeitsaxiom angewendet. Im Folgenden wird dies anhand der Zuordnung zwischen der funktionalen und der physischen Domäne erläutert. Die Zuordnung und Anwendung des Axioms zwischen den anderen Domänen verläuft analog [SuDo2000, 37-38; Suh2001]. Das Unabhängigkeitsaxiom verlangt, dass die Unabhängigkeit der funktionalen Anforderungen nach Zuordnung geeigneter Designparameter gewahrt bleibt. Vollständige Unabhängigkeit liegt vor, wenn der gefundene Designparameter für eine spezifische funktionale Anforderung keine Auswirkungen auf andere funktionale Anforderungen hat. D. h., dass jede funktionale Anforderung durch genau einen Designparameter erfüllt wird. Der Zuordnungsprozess zwischen den Domänen wird hierarchisch im Top-down-Verfahren durchgeführt, um den Entwurf weiter zu verfeinern. Man spricht in diesem Zusammenhang vom Dekompositionsprozess (Abb. 3).

Der Dekompositionsprozess verlangt, dass zwischen den Domänen hin und her gesprungen wird. Wie in Abb. 3 dargestellt, springt man ausgehend von einer funktionalen Anforderung in die physische Domäne, um einen geeigneten Designparameter zuzuordnen. Anschließend erfolgt der Rücksprung in die funktionale Domäne.

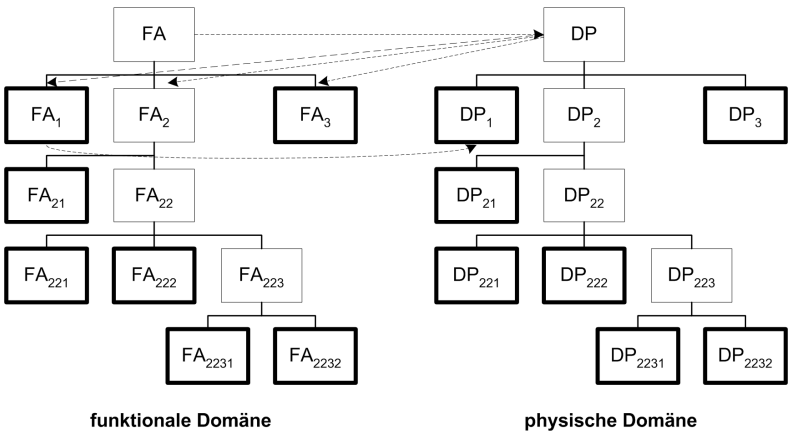

Abb. 3: Dekompositionsprozess

Die funktionale Anforderung wird auf der zweiten Ebene in ihre Teilanforderungen (FA_1, FA_2 und FA_3) zerlegt. Für jede Teilanforderung erfolgt wieder die Zuordnung geeigneter Designparameter (DP_1, DP_2 und DP_3). Dieser Prozess wird solange wiederholt, bis so genannte „elementare FA-DP-Kombinationen" gefunden wurden. Eine Elementarkombination (diese sind in Abb. 3 fett hervorgehoben) liegt vor, wenn für eine funktionale Anforderung ein Designparameter gefunden wird, der unmittelbar, d. h. ohne weitere Dekomposition, implementierbar ist.

Sobald eine Hierarchieebene im Dekompositionsprozess fertig gestellt wurde, wird das Unabhängigkeitsaxiom zur Prüfung der Unabhängigkeit der funktionalen Anforderungen herangezogen. Hierzu wird die Einflussmatrix [A] gebildet. Sie zeigt die Beziehung zwischen den funktionalen Anforderungen und Designparametern einer Hierarchieebene. Diese Beziehung wird durch folgende Gleichungen zum Ausdruck gebracht.

$$\{FA\} = [A]\{DP\} \text{ bzw. } \begin{Bmatrix} FA_1 \\ FA_2 \\ \dots \\ FA_n \end{Bmatrix} = \begin{bmatrix} A_{11} & A_{12} & \dots & A_{1n} \\ A_{21} & A_{22} & \dots & A_{2n} \\ \dots & \dots & \dots & \dots \\ A_{n1} & A_{n2} & \dots & A_{nn} \end{bmatrix} \begin{Bmatrix} DP_1 \\ DP_2 \\ \dots \\ DP_n \end{Bmatrix} \quad (1)$$

Die funktionalen Anforderungen und Designparameter einer Hierarchieebene konstituieren die Vektoren {FA} und {DP}. Die Einflussmatrix [A] ist eine $n \times n$-Matrix. Die Elemente A_{ij} beinhalten entweder den Wert „0" oder „X" (im Falle eines linearen Entwurfs, von dem hier ausgegangen wird [Suh2001, 19]). $A_{ij} = 0$ bedeutet, dass FA_i nicht durch DP_j beeinflusst wird. $A_{ij} = X$ sagt aus, dass DP_j Einfluss auf FA_i hat. Die drei folgenden Ausprägungen der Einflussmatrix [A] werden unterschieden: 1. Wenn für [A] gilt: alle $A_{ij} = 0$ für $i \neq j$, dann nehmen alle Elemente der Einflussmatrix, außer die auf der Hauptdiagonalen, den Wert „0" an. Eine solche Matrix wird als Diagonalmatrix bezeichnet. 2. Wenn für [A] gilt: entweder oberhalb oder unterhalb der Diagonalen sind alle $A_{ij} = 0$, dann wird [A] als Triangularmatrix bezeichnet. 3. Wenn für [A] gilt: sowohl oberhalb als auch unterhalb der Diagonalen existieren $A_{ij} = X$, dann wird [A] als Vollmatrix bezeichnet.

1. ungekoppelter Entwurf (Diagonalmatrix)

$$\begin{Bmatrix} FR_1 \\ FR_2 \\ FR_3 \end{Bmatrix} = \begin{bmatrix} X & 0 & 0 \\ 0 & X & 0 \\ 0 & 0 & X \end{bmatrix} \begin{Bmatrix} DP_1 \\ DP_2 \\ DP_3 \end{Bmatrix}$$

2. entkoppelter Entwurf (Triangularmatrix)

$$\begin{Bmatrix} FR_1 \\ FR_2 \\ FR_3 \end{Bmatrix} = \begin{bmatrix} X & 0 & 0 \\ X & X & 0 \\ X & X & X \end{bmatrix} \begin{Bmatrix} DP_1 \\ DP_2 \\ DP_3 \end{Bmatrix}$$

3. gekoppelter Entwurf (Vollmatrix)

$$\begin{Bmatrix} FR_1 \\ FR_2 \\ FR_3 \end{Bmatrix} = \begin{bmatrix} X & 0 & X \\ X & X & 0 \\ X & X & X \end{bmatrix} \begin{Bmatrix} DP_1 \\ DP_2 \\ DP_3 \end{Bmatrix}$$

Tab. 1: Ausprägungen der Einflussmatrix

Entsprechend der Ausprägungen der Einflussmatrix unterscheidet AD zwischen den drei Entwurfstypen der Tab. 1. Nur im ungekoppelten Entwurf sind alle funktionalen Anforderungen unabhängig voneinander, da jede funktionale Anforderung durch genau einen Designparameter erfüllt wird. Er repräsentiert daher die ideale Erfüllung des Unabhängigkeitsaxioms. Bei den anderen Typen gibt es zusätzliche Abhängigkeiten. Diese treten beim gekoppelten Entwurf am häufigsten auf. Er stellt daher eine Verletzung des Unabhängigkeitsaxioms dar. Ein Designer wendet das Unabhängigkeitsaxiom auf jeder Hierarchieebene des Dekompositionsprozesses an. Stellt er auf einer Ebene fest, dass ein gekoppelter Entwurf entstanden ist, versucht er die ermittelten FA-DP-Kombinationen so zu überarbeiten, dass entweder ein ungekoppelter oder ein entkoppelter Entwurf entsteht. Erst danach wird der Dekompositionsprozess auf tiefer liegenden Ebenen fortgesetzt.

3.3 Informationsaxiom

Liegen alternative FA-DP-Kombinationen vor, die das Unabhängigkeitsaxiom erfüllen, wird das Informationsaxiom angewendet, um den besten Entwurf zu ermitteln. Das Informationsaxiom ermöglicht eine quantitative Bewertung gegebener Entwürfe und ermöglicht eine Reduktion der Komplexität sowie eine Erhöhung der Zuverlässigkeit [ClHi2000, 274; Suh2001, 39 ff.]. In diesem Beitrag gehen wir aus Platzgründen nicht weiter auf das Informationsaxiom ein.

4 Axiomatic Design im Entwurf Serviceorientierter Architekturen

AD wurde für den Maschinenbau entwickelt und zunächst zum Entwurf von Produkten angewendet [Suh2001, 11]. Mittlerweile wurde diese Methode auch in vielen anderen Gebieten erfolgreich eingesetzt [DoSu2000; EnNo2000; Suh1990, 323-352; Suh1998; Suh2001, 341-375]. Tab. 2 gibt einen Überblick über publizierte Anwendungen von AD im Softwareentwurf.

Anwendung des Axiomatic Design zum Entwurf…	**Quelle**
… von Steuerungssoftware für elektromechanische Systeme	[HiNa1999]
… von Benutzerschnittstellen von Softwaresystemen	[Jams2004]
… einer Bibliotheksverwaltungssoftware	[KiSK1991]
… der objektorientierten Software Acclaro.	[SuDo2000]
… von Software für Programmable Logic Controller	[ScTs2000]

Tab. 2: Anwendung des Axiomatic Design im Softwareentwurf

Die oben aufgeführten Projektbeispiele (Tab. 2) belegen, dass die Vorteile des AD auch im Softwareentwurf erzielt werden können [DoSu1999, 121; HiNa1999, 1-2; Jams2004, 1;

ScTs2000, 270; SuDo2000, 95-96]. Zwischen dem Entwurf SOA und dem Entwurf von Software gibt es viele Parallelen [CeHa2005, 11; Erl2005, 321 ff.; KoHB2005]. Wir stellen die These auf, dass AD auch den Entwurf von SOA verbessern kann. Wir wollen überprüfen, ob AD helfen kann, die Architekturziele „ausgewogene Granularität“, „lose Kopplung“ und „hohe Autonomie“ zu erreichen.

Die Anwendung des AD auf den Entwurf SOA basiert auf Grundlagen, die in den vorhergehenden Abschnitten beschrieben wurden. Der Entwurfsprozess und die Inhalte der Domänen müssen allerdings an die Besonderheiten von SOA angepasst werden. Im Folgenden werden die relevanten Schritte im Entwurf SOA abgegrenzt. Anschließend illustrieren wir die Anwendung des AD beim Entwurf von SOA anhand einer Fallstudie.

4.1 Eingrenzung des relevanten Gegenstandsbereiches

Als Grundlage für die Anwendung des AD dient ein idealtypisches Vorgehensmodell zur Entwicklung SOA (Abb. 4).

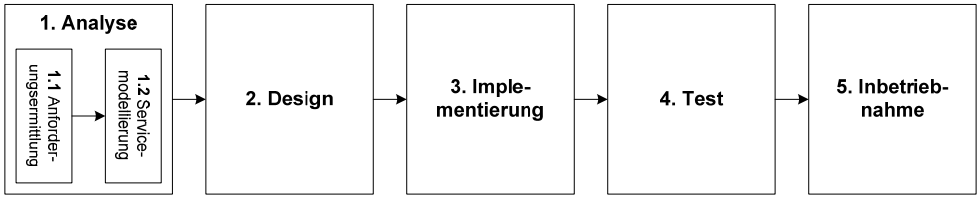

Abb. 4: Entwicklungsprozess Serviceorientierter Architekturen

Dieses Modell haben wir aus einer Synopse der Vorgehensmodelle von [BuGa2005; EAAC2004, 83 ff.; Erl2005, 359 ff.; KoHB2005; LaMB2005; MaBe2006, 99-149] abgeleitet. Es repräsentiert einen reinen Top-Down-Ansatz. Auf Grund der Problemkomplexität werden in diesem Beitrag Bottom-Up-Vorgehensweisen und Mischformen bewusst ausgeblendet. Dieses Modell bezeichnen wir im Folgenden als Entwicklungsprozess für SOA.

4.2 Anwendung des Axiomatic Design im Entwurf Serviceorientierter Architekturen

Die Anwendung des AD auf den Entwurf SOA haben wir auf der Grundlage des so genannten V-Modells des AD (Abb. 5) durchgeführt [DoSu2000, 279 ff.]. Wir haben uns dabei an der Anwendung des AD im Softwareentwurf orientiert. Anpassungen für den Entwurf SOA waren hinsichtlich der Auswahl der zu modellierenden Gegenstände in den einzelnen Domänen des AD notwendig. Außerdem mussten geeignete Vorschriften zur Ableitung von serviceorientierten Konstrukten aus der Gesamteinflussmatrix gefunden werden. Das V-Modell des AD darf nicht mit Vorgehensmodellen der Softwareentwicklung, zum Beispiel dem V-

Modell von Boehm [Boeh1981] oder dem V-Modell® XT der Bundesbehörden [KBSt2005], verwechselt werden. Die Intention des V-Modells ist, dass alle methodischen Schritte des linken Astes (Schritte eins bis vier) im Sinne des AD bearbeitet werden. Alle Schritte des rechten Astes (Schritte fünf bis sieben) können mit beliebigen Hilfsmitteln der Softwareentwicklung kombiniert werden [Suh2001, 266]. AD hilft, den Entwurf auf relevante Architekturziele zu fokussieren. Im Folgenden werden die einzelnen Schritte des V-Modells und deren Anwendung auf den Entwurf SOA beschrieben und an einem Fallsbeispiel demonstriert. Dieses Beispiel basiert auf einer Fallstudie von Erl [Erl2005, 430-444]. Grundlage ist ein Prozess zur Vorlage und Prüfung von Arbeitszeitnachweisen der Mitarbeiter, welcher durch eine SOA abgebildet werden soll. Dieser Prozess soll automatisiert werden.

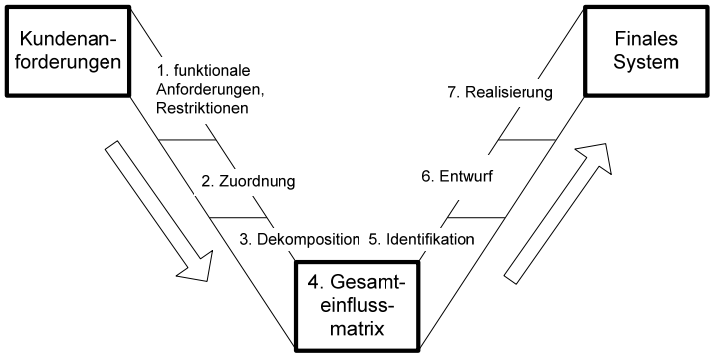

Abb. 5: V-Modell des Axiomatic Design (In Anlehnung an [SuDo2000, 96])

Der erste Schritt betrifft die Kundendomäne und die funktionale Domäne im AD. Er umfasst die Ermittlung von Kundenanforderungen an das zu entwerfende Softwaresystem. Die Kundenanforderungen umreißen relativ grob die Eigenschaften der zu entwickelnden SOA. Sie werden aus den Geschäftsprozessen abgeleitet, die in Phase „1.1 Anforderungsermittlung“ des Entwicklungsprozesses für SOA ermittelt werden [Erl2005, 363-365]. Diese Phase liefert den Informationsinput für den ersten Schritt des V-Modells des AD. Im Rahmen des Fallbeispiels wurde der folgende Prozess ermittelt (Abb. 6).

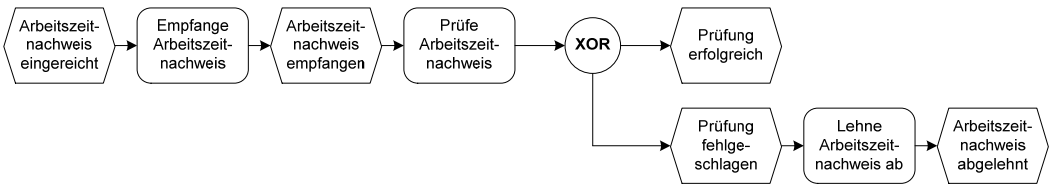

Abb. 6: Prozess zur Vorlage und Prüfung von Arbeitszeitnachweisen (In Anlehnung an [Erl2005, 430-444])

Den Kundenanforderungen werden funktionale Anforderungen und Restriktionen zugeordnet. Die funktionalen Anforderungen bilden dabei logische Kontexte ab (z. B. die betriebswirtschaftliche Funktion „Rechnung buchen“ oder die Entität „Buchhaltungssystem“),

nach denen später Operationen und Services gruppiert werden können. Restriktionen ergänzen die funktionalen Anforderungen, indem sie den Lösungsraum der physischen Domäne einschränken. Die wichtigste Kundenanforderung, die in unserem Beispiel ermittelt wurde, lautet: KA1: „Wir benötigen eine SOA, welche die vollautomatische Bearbeitung des Prozesses zur Vorlage und Prüfung von Arbeitszeitnachweisen ermöglicht“. Dieser Kundenanforderung wird folgende funktionale Anforderung zugeordnet: FA1: „Bilde den Prozess der Vorlage und Prüfung von Arbeitszeitnachweisen ab“. Eine typische Restriktion beim Entwurf SOA ist die Anforderung, die SOA auf Basis von Webservices zu entwickeln. Im zweiten und dritten Schritt werden den funktionalen Anforderungen geeignete Designparameter in der physischen Domäne zugeordnet. Die Designparameter repräsentieren Daten der Serviceverarbeitung [Erl2005, 35-37]. Es muss darauf geachtet werden, dass die Designparameter keine Restriktionen verletzen. Anschließend erfolgt die Dekomposition der funktionalen Anforderungen und Designparameter durch Sprung zwischen der funktionalen und der physischen Domäne. Während der Dekomposition muss auf jeder Hierarchieebene sichergestellt werden, dass das Unabhängigkeitsaxiom erfüllt ist. Dies geschieht durch Prüfung der Zuordnungsbeziehungen der Einflussmatrizen, die auf jeder Ebene gebildet werden. Abhängigkeiten, die auf der Diagonale einer Matrix liegen, werden durch Großbuchstaben, alle sonstigen Abhängigkeiten durch Kleinbuchstaben dargestellt. In diesem Fallbeispiel werden insgesamt fünf Dekompositionsebenen gebildet. Aus Platzgründen stellen wir nur die Matrix der zweiten Ebene dar (Tab. 3). Die Inhalte der Matrix verdeutlichen, dass z. B. zur Erfüllung der funktionalen Anforderung: FA11: „nehme Arbeitszeitnachweis entgegen“ und FA12: „verarbeite Arbeitszeitnachweis“ die Daten des Arbeitszeitnachweises, repräsentiert durch: DP11: „Arbeitszeitnachweis“, benötigt werden.

	DP11: Arbeitszeitnachweis	**DP12**: Daten zur Verarbeitung des Arbeitzeitnachweises
FA11: nehme Arbeitszeitnachweis entgegen	**B**	
FA12: verarbeite Arbeitszeitnachweis	**A**	**C**

Tab. 3: Einflussmatrix der zweiten Dekompositionsebene

Zur Erfüllung der funktionalen Anforderung FA12: „verarbeite Arbeitszeitnachweis“ werden zusätzlich Daten benötigt, die während der Verarbeitung eines Arbeitszeitnachweises wichtig sind. Diese Daten werden auf der zweiten Dekompositionsebene noch sehr abstrakt als: DP12: „Daten zur Verarbeitung des Arbeitzeitnachweises“ bezeichnet. Auf den tiefer liegenden

Dekompositionsebenen (Tab. 4) werden diese Daten jedoch weiter verfeinert und konkretisiert, z. B. zu: DP1141: „Profildaten“ oder DP11221: „abgerechnete Stunden“.

In Schritt vier wird die Gesamteinflussmatrix (Tab. 4) gebildet. Diese Matrix zeigt die Beziehungen zwischen den funktionalen Anforderungen und Designparametern aller Hierarchieebenen. Sie ergibt sich aus der Zusammenfassung der Einflussmatrizen aller Hierarchiestufen des Dekompositionsprozesses. Die Gesamteinflussmatrix ist eine wichtige Grundlage für die folgenden Schritte. Mit ihrer Hilfe wird überprüft, ob die Designentscheidungen konsistent sind, die während der Dekomposition getroffen wurden.

				DP1: Daten des Prozesses der Arbeitzeitnachweisvorlage									
				DP11: Arbeitszeit-nachweis	**DP12**: Daten zur Verarbeitung des Arbeitzeitnachweises								
				DP111: Arbeitszeit-nachweis	**DP112**: Prüfdaten				**DP113**: Prüfergebnisdaten		**DP114**: Mitarbeiter-daten	**DP115**: Mitteilungsdaten	
					DP1121: Arbeitszeitnachweisdaten			**DP1122**: Rechnungs-daten	**DP1131**: Kunden-abrechnungs-übereinstimmung	**DP1132**: Genehmigungsexistenz	**DP1141**: Profildaten	**DP1151**: Mitarbeiter-ablehnung	**DP1152**: Vorgesetzten-mitteilung
					DP11211: Stunden-nachweis	**DP11212**: Überstunden-nachweis	**DP11213**: Genehmigungsnachweis	**DP11221**: abgerechnete Stunden					
FA1: bilde den Prozess der Arbeitzeitnachweisvorlage ab	**FA11**: nehme AZN entgegen	**FA111**: speichere Arbeitszeitnachweis		**D**: speichere-Arbeitszeit-nachweis	B						C		
	FA12: verarbeite Arbeitszeitnachweis	**FA112**: ermittle Prüfdaten	**FA1121**: ermittle Arbeitszeit-nachweisdaten	**FA11211**: ermittle Stunden	*a : Nachricht D*	**P**: ermittle-Stunden			E				
				FA11212: ermittle Überstunden	*a : Nachricht D*		**Q**: ermittleÜber-stunden	I					
				FA11213: ermittle Genehmigungen	*a : Nachricht D*			**R**: ermittleGe-nehmigungen	J				
			FA1122: ermittle Rechnungs-daten	**FA11221**: ermittle abgerechnete Stunden					**S**: ermittleAbge-rechnete-Stunden		F		
		FA113: prüfe Arbeitszeitnachweis	**FA1131**: prüfe Übereinstimmung mit Kundenabrechnung		*b : Nachricht P*			*b : Nachricht S*	**K**: prüfeÜber-einstimmung				
			FA1132: prüfe Genehmigungen für Überstunden			*b : Nachricht Q*	*b : Nachricht R*			**L**: prüfeGe-nehmigungen		G	
		FA114: aktu-alisiere Mitarbei-terdaten	**FA1141**: aktualisiere Profildaten						*c : Nachricht K*	*c : Nachricht L*	**M**: aktualisiere-Profildaten		H
		FA115: sende Mitteilungen	**FA1151**: sende Ablehnung an Mitarbeiter						*d : Nachricht K*	*d : Nachricht L*	*e : Nachricht M*	**N**: sendeAb-lehnung	
			FA1152: sende Mitteilung an Vorgesetzten						*d : Nachricht K*	*d : Nachricht L*	*e : Nachricht M*		**O**: sende-Mitteilung

Tab. 4: Gesamteinflussmatrix[2]

Der fünfte Schritt dient der Identifikation serviceorientierter Konstrukte. Die Inhalte der Gesamteinflussmatrix werden dabei auf Services und Operationen abgebildet. Tab. 5 zeigt einen Ausschnitt des Ergebnisses. Aus den funktionalen Anforderungen werden vorläufige Services, so genannte Servicekandidaten, abgeleitet. Die Daten, die der Service verarbeitet, ergeben sich aus den Designparametern. Operationen der Services werden durch die Elemente innerhalb der Gesamteinflussmatrix repräsentiert.

[2] Eine größere Darstellung dieser Matrix kann über die Website http://www.wirtschaft.tu-ilmenau.de/im/ abgerufen werden.

Service	FA1	Arbeitzeitnach-weisvorlage-prozess	FA11	Arbeitszeitnach-weisentgegen-nahme	FA12	Arbeitszeitnach-weisverarbeitung	FA112	Prüfdatener-mittlung	FA113	Arbeitszeitnach-weisprüfung
Daten	**DP11**	Arbeitzeitnachweis	**DP111**	Arbeitszeitnachweis	**DP112**	Prüfdaten	**DP1121**	Arbeitszeitnachweis-daten	**DP1131**	Kundenabrechnungs-übereinstimmung
	DP12	Daten zur Verarbeitung des Arbeitzeitnachweises			**DP113**	Prüfergebnisdaten	**DP1122**	Rechnungsdaten	**DP1132**	Genehmigungs-existenz
					DP114	Mitarbeiterdaten				
					DP115	Mitteilungsdaten				
Opera-tionen	**A**	*Arbeitzeitnachweis-vorlageprozess*	**B**	*Arbeitszeitnachweis-entgegennahme*	**C**	*Arbeitszeitnachweis-verarbeitung*	**E**	*Prüfdatenermittlung*	**F**	*Arbeitszeitnachweis-prüfung*
									K	prüfeÜberein-stimmung
			D	speichereArbeitszeit-nachweis					**L**	prüfeGenehmigungen

Tab. 5: Identifikation serviceorientierter Konstrukte

Im sechsten Schritt werden die identifizierten Konstrukte in eine vorläufige SOA überführt. Die Ergebnisse werden anschließend der Phase „2. Design“ des Entwicklungsprozesses für SOA zugeführt. Die Spezifikation der SOA kann mit Hilfsmitteln der UML abgebildet werden [LaPi2005]. Sie beinhaltet die zuvor identifizierten Services, deren Operationen und Daten. Außerdem werden Abhängigkeiten zwischen Services auf Grund einer Servicekomposition oder auf Grund des Nachrichtenaustausches zwischen Services spezifiziert. Diese Abhängigkeiten werden ebenfalls aus der Gesamteinflussmatrix abgeleitet. Servicekompositionen ergeben sich aus der hierarchischen Strukturierung der funktionalen Anforderungen und Designparameter. Der Nachrichtenaustausch zwischen Services wird aus den nichtdiagonalen Inhaltselementen der Gesamteinflussmatrix abgeleitet.

Abb. 7: Servicekomposition

Abb. 7 wurde in Anlehnung an ein UML-Klassendiagramm erstellt. Es beinhaltet alle im letzten Schritt identifizierten Services sowie deren Operationen und Abhängigkeiten.

Kompositionsbeziehungen zwischen Services werden durch Aggregationsbeziehungen im UML-Diagramm ausgedrückt.
Schritt sieben betrifft die Prozessdomäne im AD. Er bezieht sich auf die Realisierung der SOA auf einer technischen Plattform. Da es sich hierbei um Aufgaben außerhalb des Entwurfs handelt, verzichten wir auf eine Beschreibung dieses Schritts.

4.3 Kritische Analyse des Beitrags von Axiomatic Design

Im Folgenden wollen wir darlegen, welchen Beitrag AD zur Erreichung der Architekturziele „hohe Autonomie“, „lose Kopplung“ und „ausgewogene Granularität“ beim Entwurf von SOA leisten kann und welche Nachteile dem gegenüber stehen.
Die Anwendung von AD fördert eine hohe Autonomie der Services, da Entwurfsanforderungen klar voneinander abgegrenzt werden. Im Rahmen der Zuordnung und Dekomposition (Schritt zwei und drei im V-Modell des AD) wird sichergestellt, dass für jede funktionale Anforderung einer Dekompositionsebene auf der nächst tieferen Ebene ausschließlich FA-DP-Kombinationen erarbeitet werden, die einen Beitrag zur Erfüllung dieser funktionalen Anforderung leisten. Im Fallbeispiel (Tab. 4) zielen z. B. die funktionalen Anforderungen FA1131, FA1132 und korrespondierende DP ausschließlich auf die Erfüllung der funktionalen Anforderung FA113: „prüfe Arbeitszeitnachweis“. Dies zeigt, dass auf jeder Dekompositionsebene kohärente Teillösungen gruppiert werden, die auf die Erfüllung einer funktionalen Anforderung abzielen. Auf diese Weise entstehen Services, deren Operationen ebenfalls auf genau einen definierten Kontext (z. B. eine Aufgabe) ausgerichtet sind. Aus der funktionalen Anforderung FA113 wird der Service „Arbeitszeitnachweisprüfung“ abgeleitet (Abb. 7). Er beinhaltet die zwei Operationen „prüfeÜbereinstimmung“ und „prüfeGenehmigung“. Diese Operationen sind kohärent, da sie beide auf genau einen Kontext (die Aufgabe, den Arbeitszeitnachweis zu prüfen) ausgerichtet sind.
Durch AD wird auch das Ziel der losen Kopplung gefördert, da die Unabhängigkeit von Entwurfsbestandteilen angestrebt wird. Dies wird durch das Unabhängigkeitsaxiom erzielt. Es stellt sicher, dass jede funktionale Anforderung durch möglichst wenige Designparameter beeinflusst wird. Abhängigkeiten sind nur auf oder unterhalb der Diagonalen der Gesamteinflussmatrix erlaubt. Im Fallbeispiel (Tab. 4) ist das Unabhängigkeitsaxiom erfüllt, da oberhalb der Diagonalen der Gesamteinflussmatrix alle Felder leer sind. Durch Verminderung der Abhängigkeiten in der Gesamteinflussmatrix wird in der SOA die Anzahl der Beziehungen zwischen den Services reduziert. Im Fallbeispiel erhält z. B. der Service

„Arbeitszeitnachweisprüfung“ einen Dateninput vom Service „Prüfdatenermittlung“. In der Gesamteinflussmatrix wird dies durch die nichtdiagonalen Elemente „b“ verdeutlicht. Dieser Service sendet selbst Daten an die Services „Mitarbeiterdatenaktualisierung“ und „Mitteilungsversendung“. Dies wird durch die nichtdiagonalen Elemente „c“ und „d“ verdeutlicht.

AD trägt auch zu einer ausgewogenen Servicegranularität bei, da Entwurfsbestandteile in überschaubarer Komplexität entstehen. Das Top-Down-Vorgehen bei der Zuordnung und Dekomposition bewirkt, dass ein Gesamtsystem rekursiv in immer feiner granulierte Subsysteme zerlegt wird. Jede Dekompositionsebene enthält nur die FA-DP-Kombinationen, die dem Abstraktionsniveau dieser Ebene entsprechen. In der Gesamteinflussmatrix (Tab. 4) wurde z. B. auf der dritten Dekompositionsebene die noch relativ abstrakte FA112: „ermittle Prüfdaten“ festgelegt. Erst auf der vierten und fünften Ebene erfolgt eine Konkretisierung hinsichtlich der zu ermittelnden Daten – z. B. konkretisiert FA1121, dass Arbeitszeitnachweisdaten ermittelt werden müssen, FA11211 konkretisiert noch stärker, dass es sich dabei u. a. um Stunden handelt. Auf diese Weise wird die Komplexität der gesamten SOA über mehrere Ebenen auf Einheiten überschaubarer Größe verteilt. So wird sichergestellt, dass keine zu grob granulierten Services entstehen. Im geschilderten Beispiel wurde aus FA112 der Service „Prüfdatenermittlung“ abgeleitet (Abb. 7). Er sorgt für die Komposition der feiner granulierten Services „Arbeitszeitnachweisdatenermittlung“ und „Rechnungsdatenermittlung“.

Diesen Vorteilen stehen einige negative Aspekte von AD gegenüber. Ein Nachteil der Anwendung von AD ist der hohe Dokumentationsaufwand. Auf jeder Ebene des Zuordnungs- und Dekompositionsprozesses müssen Designgleichungen und -matrizen erstellt werden. Zwar kann dieser Aufwand durch Verwendung der Software Acclaro® verringert werden.[3] Die FA-DP-Kombinationen müssen aber in jedem Fall manuell eingegeben werden. Die Verfechter des AD führen die starke Strukturierung und Formalisierung von Entwurfsprozessen als Vorteil auf, da positive Effekte, wie die Reduzierung von Entwurfsschritten und eine Erhöhung der Kreativität der Designer, entstehen sollen [Suh2001, 239 ff.]. Allerdings ist die Erstellung der Gesamteinflussmatrix aufwändig. Die benötigte Zeit fehlt evtl. für andere Aufgaben. Wie die Softwareentwicklung zeigt, kann eine zu starke Strukturierung und Formalisierung auch zu nachteiligen Effekten führen, z. B. zur Einschränkung von Kreativität.

[3] Informationen zur Software Acclaro® sind unter: http://www.axiomaticdesign.com abrufbar.

5 Zusammenfassung und Ausblick

Wir haben an einer Fallstudie demonstriert, dass AD dazu beiträgt, die Architekturziele „ausgewogene Granularität“, „lose Kopplung“ und „hohe Autonomie“ für SOA zu fördern. AD hilft auch, den Entwurfsprozess für SOA aus einer fachlichen Perspektive zu strukturieren.

Wir konnten zwar zeigen, dass AD einen positiven Beitrag zum Entwurf SOA leistet. Offen ist allerdings, wie groß dieser Beitrag in realen Entwicklungsprojekten ist. In weiteren Experimenten und Fallstudien bleibt daher zu prüfen, inwieweit die beschriebenen Vorteile in realen Projekten zur Entwicklung umfangreicher SOA erreicht werden können. Außerdem muss untersucht werden, ob die Vorteile der Anwendung von AD den Aufwand rechtfertigen, der mit der Anwendung der Methode verbunden ist.

In Rahmen dieses Beitrages konnten wir aus Platzgründen nur auf einige Aspekte des AD eingehen. Es wäre z. B. interessant, auch den Beitrag des Informationsaxioms zum Entwurf von SOA zu analysieren, da es die quantitative Bewertung der Komplexität eines Entwurfes ermöglicht und somit zur Bewertung und Auswahl alternativer SOA-Spezifikationen herangezogen werden kann. Außerdem bietet AD weitere Hilfsmittel zur Unterstützung der Implementierungsphase an [Suh1998; Suh2001, 196-198]. Wir planen, auch den Beitrag dieser Hilfsmittel für die Entwicklung von SOA zu untersuchen.

Wir haben uns darauf beschränkt, die Auswirkungen von AD auf ausgewählte Architekturziele zu untersuchen. Wir vermuten, dass AD auch einen positiven Einfluss auf weitere Ziele – wie z. B. Geschäftsorientiertheit – hat. Allerdings gilt dies nicht für alle Ziele. Bestimmte Ziele, wie die Zustandslosigkeit oder Abstraktheit repräsentieren grundlegende Prinzipien für SOA, die zwar beim Entwurf berücksichtigt werden müssen, aber nicht direkt durch eine spezifische Entwurfsmethode beeinflusst werden können.

Fraglich ist auch, inwieweit die am Fallbeispiel demonstrierten Erkenntnisse allgemeine Gültigkeit besitzen. Die Konzepte von Axiomatic Design sind in jedem Anwendungsgebiet – beim Entwurf von Produkten, Software, SOA, etc. – dieselben. Daraus schlussfolgern wir, dass auch die Vorteile des Axiomatic Design in jedem Anwendungsgebiet erzielt werden können. Wir vermuten, dass die Erreichung der Architekturziele für SOA, unabhängig von den Besonderheiten eines spezifischen Entwicklungsprojektes, durch den Einsatz von Axiomatic Design gefördert werden kann. Zur Überprüfung dieser Vermutung, haben wir die Durchführung und Evaluierung weiterer Fallstudien und Praxisprojekte geplant.

Literaturverzeichnis

[Balz1998] *Balzert, H.*: Lehrbuch der Software-Technik: Software-Management, Softwarequalitätssicherung, Unternehmensmodellierung. Spektrum, Heidelberg et al. 1998.

[Boeh1981] *Boehm, B. W.*: Software Engineering Economics. Prentice-Hall, Englewood Cliffs et al. 1981.

[BuGa2005] *Buchmann, I.; Gamber, M.*: Methoden zur Unterstützung der Entwicklung einer SOA. In: *Cremers, A. B. et al. (Hrsg.)*: Informatik 2005: Informatik live!; Beiträge der 35. Jahrestagung der Gesellschaft für Informatik e.V. GI, Bonn 2005, S. 601-605.

[CaGl1990] *Card, D. N.; Glass, R. L.*: Measuring Software Design Quality. Prentice Hall, Englewood Cliffs 1990.

[CeHa2005] *Cervantes, H.; Hall, R. S.*: Technical Concepts of Service Orientation. In: *Stojanovic, Z. et al. (Hrsg.)*: Service-Oriented Software System Engineering: Challenges and Practices. IGP, Hershey et al. 2005, S. 1-26.

[ClHi2000] *Clapsis, P. J.; Hintersteiner, J. D.*: Enhancing Object-Oriented Software Development through Axiomatic Design. In: First International Conference on Axiomatic Design (ICAD2000). Cambridge 2000, S. 272-277.

[DJMZ2005] *Dostal, W.; Jeckel, M.; Melzer, I.; Zengler, B.*: Service-Orientierte Architekturen mit Web Services: Konzepte - Standards - Praxis. Spektrum, München 2005.

[DoSu2000] *Do, S.-H.; Suh, N. P.*: Object-Oriented Software Design with Axiomatic Design. In: Proceedings of ICAD2000. Cambridge 2000, S. 278-284.

[DoSu1999] *Do, S.-H.; Suh, N. P.*: Systematic OO Programming with Axiomatic Design. In: IEEE Computer 32 (1999) 10, S. 121-124.

[EAAC2004] *Endrei, M.; Ang, J.; Arsanjani, A.; Chua, S. et al.*: Patterns: Service-Oriented Architecture and Web Services. http://www.redbooks.ibm.com, 2004, Abruf am 2006-06-27.

[EKAP2005] *Erradi, A.; Kulkarni, N.; Anand, S.; Padmanabhuni, S.*: Designing Reusable Services: An Experimential Perspective for the Securities Trading Domain. In: *Chung, J.-Y. et al. (Hrsg.)*: Proceedings of the First International Workshop on Design of Service-Oriented Applications (WDSOA'05). IBM Research Division, Amsterdam 2005, S. 25-32.

[EMPR2005] *Eidson, B.; Maron, J.; Pavlik, G.; Raheja, R.*: SOA and the Future of Application Development. In: *Chung, J.-Y. et al. (Hrsg.)*: Proceedings of the First International Workshop on Design of Service-Oriented Applications (WDSOA'05). IBM Research Division, Amsterdam 2005, S. 1-8.

[EnNo2000] *Engelhardt, F.; Nordlund, M.*: Strategic Planning based on Axiomatic Design. In: Proceedings of ICAD2000. Cambridge 2000, S. 26-34.

[Erl2005] *Erl, T.*: Service-Oriented Architecture: Concepts, Technology, and Design. Prentice Hall PTR, Upper Saddle River et al. 2005.

[HaSc2006] *Hagen, C.; Schwinn, A.*: Measured Integration - Metriken für die Integrationsarchitektur. In: *Schelp, J. et al. (Hrsg.)*: Integrationsmanagement. Planung, Bewertung und Steuerung von Applikationslandschaften. Springer, Berlin et al. 2006, S. 267-292.

[HiNa1999] *Hintersteiner, J. D.; Nain, A. S.*: Integrating Software into Systems: An Axiomatic Design Approach. In: Proceedings of the 3rd International Conference on Engineering Design and Automation. Vancouver 1999, S. 1-7.

[Jams2004] *Jamshidnezhad, B.*: Towards a Rational Basis for User Interface Design Methods. In: Proceedings of ICAD2004. Seoul 2004, S. 1-4.

[Kaye2003] *Kaye, D.*: Loosely Coupled: The Missing Pieces of Web Services. RDS Press, Marin County 2003.

[KBSt2005] *Koord.- und Beratungsstelle der Bundesreg. für IT in der Bundesver*: V-Modell XT, Version 1.2.0. http://www.v-modell-xt.de, 2005, Abruf am 2006-04-25.

[KiSK1991] *Kim, S. J.; Suh, N. P.; Kim, S. G.*: Design of Software Systems based on Axiomatic Design. In: Robotics & Computer-Integrated Manufacturing 8 (1991) 4, S. 243-255.

[KoHB2005] *Kotonya, G.; Hutchinson, J.; Bloin, B.*: A Method for Formulating and Architecting Component- and Service-Oriented Systems. In: *Stojanovic, Z. et al. (Hrsg.)*: Service-Oriented Software System Engineering: Challenges and Practices. IGP, Hershey et al. 2005, S. 155-181.

[LaMB2005] *Laures, G.; Meyer, H.; Breest, M.*: An Engineering Method for Semantic Service Applications. In: *Chung, J.-Y. et al. (Hrsg.)*: Proceedings of the First International Workshop on Design of Service-Oriented Applications (WDSOA'05). IBM Research Division, Amsterdam 2005, S. 79-86.

[LaPi2005] *Latchem, S.; Piper, D.*: Service-Oriented Design Process Using UML. In: *Stojanovic, Z. et al. (Hrsg.)*: Service-Oriented Software System Engineering: Challenges and Practices. IGP, Hershey et al. 2005, S. 88-108.

[MaBe2006] *Marks, E. A.; Bell, M.*: Service-Oriented Architecture: A Planning and Implementation Guide for Business and Technology. Wiley, Hoboken et al. 2006.

[Raas1993] *Raasch, J.*: Systementwicklung mit strukturierten Methoden: ein Leitfaden für Praxis und Studium. 3. Aufl., Hanser, München et al. 1993.

[ScTs2000] *Schreyer, M.; Tseng, M. M.*: Hierarchical State Decomposition for Design of PLC Software by applying Axiomatic Design. In: Proceedings of ICAD2000. Cambridge 2000, S. 264-271.

[SiHu2005] *Singh, M. P.; Huhns, M. N.*: Service-Oriented Computing: Semantics, Processes, Agents. Wiley, Chichester et al. 2005.

[SuDo2000] *Suh, N. P.; Do, S.-H.*: Axiomatic Design of Software Systems. In: Annals of the CIRP 49 (2000) 1, S. 95.

[Suh1990] *Suh, N. P.*: The Principles of Design. Oxford, New York et al. 1990.

[Suh1998] *Suh, N. P.*: Axiomatic Design Theory for Systems. In: Research in Engineering Design 10 (1998) 4, S. 189-209.

[Suh2001] *Suh, N. P.*: Axiomatic Design: Advances and Applications. Oxford, New York 2001.

[VACI2005] *Vogel, O.; Arnold, I.; Chughtai, A.; Ihler, E. et al.*: Software-Architektur: Grundlagen - Konzepte - Praxis. Spektrum, München 2005.

[ZiTP2003] *Zimmermann, O.; Tomlinson, M.; Peuser, S.*: Perspectives on Web Services: Applying SOAP, WSDL and UDDI to Real-World Projects. Springer, Berlin et al. 2003.