1 Motivation

Wir leben in harten Zeiten. Die soziale wird durch eine globale Marktwirtschaft abgelöst, bei der nur das Recht des Stärkeren zählt. Die Marketing-Abteilungen regieren die Welt. Konsequenterweise muss man zum Überleben schnell und flexibel sein. Im Grunde haben wir eine Renaissance des Darwinismus, bei dem es unter anderen heißt:

> *Weder die stärksten noch die intelligentesten Arten überleben, sondern nur die, die am schnellsten auf Veränderungen reagieren.*

Der entscheidende Punkt ist Flexibilität. In allen großen Firmen und verteilten Systemen ist IT (Informationstechnik) dabei von größter Wichtigkeit. Man kann sogar sagen, dass IT dabei ist, zum entscheidenden Faktor (englisch: key enabler) von geschäftlichem Erfolg zu werden.

Die Prozesse und Systeme werden dabei immer komplexer. Wir haben bereits die Stufe verlassen, bei der Automatisierung nur einzelne Systeme betraf, und bewegen uns in großen Schritten in eine Welt, in der alle Systeme zu einem einzigen verteilten System werden.

Die daraus resultierende Herausforderung ist Wartbarkeit. Es zeigt sich, dass die herkömmliche Art und Weise, mit den Anforderungen von Skalierbarkeit und Verteilung umzugehen, nicht mehr funktioniert. Wir können nämlich nicht mehr länger einfach harmonisieren und alles zentral kontrollieren. Zentralismus, eine der Vorbedingungen für Harmonisierung und Kontrolle, skaliert nicht unbegrenzt, und wir sind dabei, die Grenze dieser Skalierbarkeit zu erreichen. Aus diesem Grund müssen wir neue Mechanismen finden, wie wir mit den aktuellen Anforderungen umgehen. Das bedeutet zunächst, dass ein solcher Ansatz Heterogenität und Dezentralismus akzeptieren muss.

Zusätzlich haben wir nach wie vor das Problem einer immer wieder auftretenden Kluft zwischen Fachlichkeit und Technik, das sogenannte »Business/IT-Gap«. Diese Kluft äußert sich im Wesentlichen dadurch, dass die beteiligten Personen ein unterschiedliches Verständnis besitzen und aneinander vorbeireden. Fachexperten und Techniker scheinen verschiedene Sprachen zu sprechen. Wir haben also auch das Ziel, dafür zu sorgen, dass sich diese Welten einander annähern.

Hier kommt Service-orientierte Architektur (SOA) ins Spiel. SOA ist ein Ansatz, der dafür sorgt, dass Systeme skalierbar und flexibel bleiben, während sie wachsen und sich verändern. Gleichzeitig wird versucht, die Kluft zwischen Fachseite und IT zu reduzieren.

Der SOA-Ansatz basiert auf drei Hauptelementen:

- **Services**, die auf der einen Seite in sich abgeschlossene fachliche Funktionalitäten (als Teil ein oder mehrerer Geschäftsprozesse) repräsentieren und auf der anderen Seite von beliebigen Plattformen und Technologien bereitgestellt werden können.
- **Eine spezifische Infrastruktur**, Enterprise-Service-Bus (ESB) genannt, die es ermöglicht, diese Services einfach und flexibel zu verwenden und zu kombinieren.
- **Richtlinien (englisch: policies) und Prozesse**, die die Tatsache berücksichtigen, dass große verteilte Systeme heterogen sind, sich in permanenter Wartung und Pflege befinden und unterschiedlichen Eigentümern gehören.

SOA ist damit ein Konzept, das Heterogenität, Dezentralismus und Fehlertoleranz als einzige Möglichkeit betrachtet, große Systeme wartbar und flexibel zu halten.

Klingt wie ein Traum, oder?

Das Problem dabei ist, dass man SOA nicht kaufen kann; man muss es verstehen und in der Praxis leben. SOA ist ein Paradigma, ein Denkmuster, ein Wertesystem für die Architektur und das Design großer verteilter Systeme, die intern wiederum aus einzelnen Systemen bestehen. SOA ist also ein Denkmuster für die Wartung und Pflege von Systemlandschaften (Systemen, die aus Systemen bestehen).

Dieses Buch wird dieses Paradigma erklären und auf Basis konkreter Erfahrungen diskutieren. Bei der Diskussion geht es zwar immer noch um die Konzepte, aber sie werden in einem Detaillierungsgrad betrachtet, der die Probleme der berüchtigten Hockeyschläger[1]-Funktion vermeidet (siehe Abbildung 1-1): Bis zu einem gewissen Maß an Komplexität ist der dazu notwendige Aufwand relativ gering, und alles sieht bestens aus. Doch dann kommt der Punkt, bei dem der Aufwand plötzlich stärker steigt als die Komplexität, sodass die Kosten den Nutzen übersteigen, bis es schließlich zum Kollaps kommt.

Im Alltag wird SOA oft nur unzureichend erklärt und realisiert. Es ist nicht damit getan, eine Infrastruktur wie Web-Services zu etablieren. Dies reicht, um Interoperabilität zwischen Systemen herzustellen, führt aber nicht unbedingt dazu, dass verteilte Geschäftsprozesse skalieren. Die gesamte Architektur muss zusammenpassen. Dazu gehören der Umgang mit Services, die Infrastruktur, Richtlinien und Prozesse. Wenn man erst einmal verstanden hat, wie man SOA realisiert, ist die Umsetzung nicht wirklich schwierig; aber man benötigt Zeit und

[1] Beim Hockeyschläger meint der Amerikaner natürlich einen Eishockeyschläger.

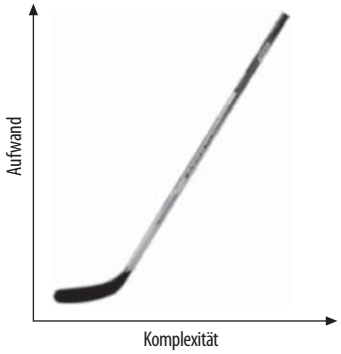

Abbildung 1-1: Die Hockeyschläger-Funktion

Courage (OK, es ist doch schwierig). Dazu gehört auch eine Menge Aufwand, damit die beteiligten Personen verstehen, was passiert (das fängt bei einem selbst an). Wenn man diesen Aufwand scheut, wird SOA ein Fehlschlag.

Bevor wir in den nachfolgenden Kapiteln in die Details gehen, würde ich gern noch etwas zum Kontext und zur Geschichte von SOA sagen. Die nachfolgenden Abschnitte werden daher noch einige »Hintergrund-Stories« rund um SOA präsentieren, um mit dem Thema schon etwas vertrauter zu werden.

1.1 Kennzeichen großer Systeme

SOA ist ein Konzept für große verteilte Systeme. Um SOA zu verstehen, muss man deshalb die Eigenschaften großer Systeme verstehen.

Große Systeme bestehen zunächst einmal aus **Altlasten** (englisch: legacy). Das ist nicht despektierlich gemeint, denn im Grunde ist jedes in Betrieb befindliche System eine Altlast. Es bedeutet aber, dass man SOA nicht als Konzept für die Entwicklung eines neuen Systems (oder Systemverbunds) betrachten kann. Man muss sich der Tatsache stellen, dass die Systeme im Normalfall bereits verwendet werden und dies auch weiterhin der Fall sein wird. Das bedeutet, dass SOA nicht mit einem Projekt vergleichbar ist, das ein neues System entwirft. Es beschäftigt sich vielmehr mit Änderungen an der Struktur und den Eigenschaften existierender Systeme. Dadurch müssen alte Plattformen unterstützt werden, und Rückwärtskompatibilität spielt eine große Rolle. Genau genommen ist SOA ein Konzept für die Wartung und Pflege großer Systemlandschaften.

Naturgemäß sind große Systeme **heterogen**. Die einzelnen Komponenten dienen verschiedenen Zwecken und wurden zu unterschiedlichen Zeiten von unterschiedlichen Personen und Teams implementiert. Systemlandschaften sind eine wüste Zusammenstellung verschiedener Plattformen, Programmiersprachen und Programmierparadigmen. Selbst die Middleware kann heterogen sein. In der Vergangenheit gab es oft den Ansatz, das Problem der Heterogenität durch

eine Harmonisierung zu lösen. Es ist keine Frage, dass Harmonisierung (da wo sie funktioniert) hilft. Wenn man veraltete Plattformen los wird, macht dies Dinge im Normalfall einfacher. Das Problem ist, dass man den Zustand der Harmonisierung trotzdem nie erreicht. Kurz bevor das letzte Fünkchen Heterogenität eliminiert wird, gibt es eine Firmenfusion oder eine neue Technologie, und alles fängt von vorne an.

Ein Grund für Heterogenität ist die **lange Lebensdauer** der Systeme und der darin verwalteten Daten. Es kommen zwar immer neue Funktionen und Systeme hinzu, aber es fällt schwer, existierende Systeme abzuschalten. Ein Problem dabei ist, dass das Abschalten keinen direkten Geschäftsnutzen hat. Es handelt sich vielmehr um eine Investition in die Wartbarkeit der Systeme. Derartige Investitionen haben in der Praxis gegenüber neuen konkreten Features mit fachlichem Nutzen einen schweren Stand. Sie werden daher oft erst dann getätigt, wenn die Systeme außer Kontrolle geraten sind und die Probleme offensichtlich werden. Doch dann ist es mitunter zu spät oder es wird zumindest sehr teuer.

Naturgemäß sind große Systeme **komplex**. Aus diesem Grund kann es schwierig werden, die richtige Stelle für eine Änderung zu finden und die Auswirkungen einer Änderung vorherzusagen. In [*Spanyi03*] ist das wie folgt formuliert:

> *So etwas wie den »schnellen Fix« gibt es nicht. Organisationen sind komplexe fachliche Gebilde, bei denen eine Änderung in einer Komponente mit hoher Wahrscheinlichkeit Auswirkungen auf andere Komponenten hat.*

Große Systeme haben eine weitere wichtige Eigenschaft, die große Auswirkungen hat: **unterschiedliche Eigentümer**. Die einzelnen Systeme werden von unterschiedlichen Teams, Abteilungen, Geschäftseinheiten, Konzerntöchtern oder Firmen verwaltet, und dies bedeutet unterschiedliche Budgets, Zeitpläne, Ansichten und Prioritäten, die alle in Betracht gezogen werden müssen. Selbst wenn alles formal unter zentraler Kontrolle liegt, ist es de facto oft so, dass es den Verantwortlichen nicht möglich ist, operative Änderungen, die alle Systeme betreffen, von oben herab zu befehlen und dabei sicher zu sein, dass diese Befehle auch umgesetzt werden. Sie sind also auf Zusammenarbeit und Verhandlungen angewiesen, die aufgrund politischer Intrigen mitunter kompliziert werden können.

Eine weitere Eigenschaft großer Systeme ist ihr **Mangel an Perfektionismus**. Perfektionismus ist schlicht und einfach nicht bezahlbar. Oder wie es Winston Churchill einst sagte:

> *Perfektionismus wird P-A-R-A-L-Y-S-E buchstabiert.*

In der Praxis verhalten sich die Systeme immer etwas »schlampig«. 99 % ihrer Aufgaben werden fehlerfrei erledigt, aber 1 % der Aufgaben führen zu Fehlern, die dann manuell nachgearbeitet werden müssen, auf einem Fehlerarbeitsplatz

landen oder zu unzufriedenen Kunden führen. Perfekt ist keines der Systeme. Lediglich das Maß an Perfektionismus ist verschieden (lebenswichtige Systeme haben sicherlich ein höheres Maß an Perfektionismus, aber auch hier gibt es eine Grenze, ab der man die Kosten zur Beseitigung eines geringen Risikos als zu hoch einschätzt).

Genauso haben große Systeme immer ein gewisses Maß an **Redundanz**. Während manche Redundanz dabei nicht beabsichtigt ist, gibt es auch einen signifikanten Anteil an »verwalteter« oder »kontrollierter« Redundanz. Schon aus Performance-Gründen ist es nicht möglich, alle Daten normalisiert zu halten, sodass sie nur genau einmal gespeichert werden. Im einfachen Fall der Redundanz ist zumindest klar, wer der Eigentümer (englisch: master) ist und wer eine Kopie der Daten hält (jede Änderung muss beim Master erfolgen und wird dann an die Kopien durchgereicht). In komplexen Szenarien gibt es mehrere Eigentümer oder der Haupteigentümer ist nicht klar definiert. Wenn Kunden zum Beispiel in mehreren Firmen gehalten werden, die gemeinsame Geschäftsprozesse abwickeln, wird jede Firma Eigentümer ihrer eigenen Kundendaten sein. In der Praxis kann die Sicherstellung von Konsistenz deshalb zu einem großen Problem werden.

Schließlich gilt für große Systeme, dass jede Form von **Flaschenhals »tödlich« sein kann**. Das bedeutet nicht, dass keine Flaschenhälse existieren, aber es ist ein wichtiges allgemeines Ziel, Flaschenhälse so weit wie möglich zu vermeiden. Dabei meine ich nicht nur technische Flaschenhälse. Auch organisatorische Flaschenhälse verhindern, dass Systeme skalieren, weil die Prozesse irgendwann nicht mehr funktionieren.

1.2 Das Märchen vom »Magischen Bus«

Es war einmal eine Firma, die über die Jahre kräftig gewachsen war. Doch dann fing sich diese Firma eine schwere Krankheit ein, die »Durcheinander« genannt wurde. Dies führte dazu, dass die Menschen die Kontrolle über ihre Systemlandschaft verloren, und wann immer sie versuchten, die Kontrolle zurückzuerlangen, war der Preis entweder zu hoch oder alles wurde nur noch schlimmer.

Die Firma bat viele Experten, wie Wahrsager, Weise und Zauberer, um Hilfe, und diese brachten viele verschiedene Ideen ins Spiel, wie neue Systementwürfe, Protokolle oder Architektur-Patterns. Aber es wurde alles nur noch schlimmer, sodass die Lage hoffnungslos wurde.

Eines Tages erschien ein Prinz, der Enterprise Integration genannt wurde, und behauptete, er könne die Firma heilen. Er ging zum CEO der Firma und sagte: »Dein Problem ist ein Mangel an Interoperabilität. Bei einem derartigen Durcheinander von Systemen und Protokollen hast Du das Problem, dass Du für jede Verbindung zwischen zwei Systemen eine individuelle Lösung benötigst. Selbst wenn Du nur 10 unterschiedliche Plattformen und 5 verschiedene Protokolle hättest, bräuchte man über 100 verschiedene Lösungen, um jede Plattform

mit jeder anderen Plattform kommunizieren zu lassen. Und Du hast weit mehr als nur 10 Plattformen.« Die genaue Methode, mit der die Zahl 100 berechnet wurde, ist leider nicht überliefert, aber einige erhalten gebliebene Schmierzettel lassen den Schluss zu, dass jede mögliche Verbindung zweier Plattformen mit der durchschnittlichen Anzahl verwendeter Protokolle kombiniert worden war. »Sieh nur meine Skizze hier«, fuhr der Prinz fort und zeigte die Skizze, die in Abbildung 1-2 rekonstruiert wurde, »wir lösen Dein Problem ganz einfach, indem wir einen *Magischen Bus* erzeugen.«

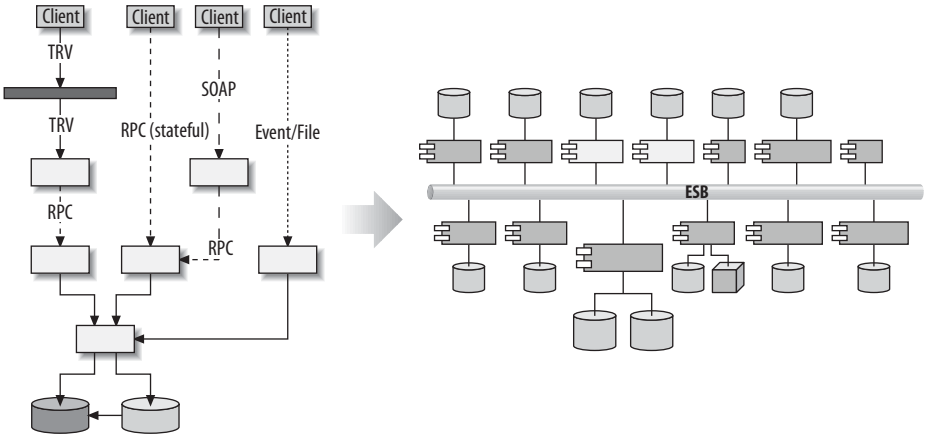

Abbildung 1-2: Reproduktion der ersten Skizze eines Magischen Busses

»Was ist ein Magischer Bus?«, fragte der CEO.

Der Prinz antwortete: »Ein Magischer Bus ist ein Stück Software, das die Anzahl der Verbindungen und Schnittstellen in Deinem System reduziert. Während Dein bisheriger Ansatz für `n` Systeme bis zu `n*(n-1)/2` Verbindungen benötigt (und doppelt so viele Schnittstellen), braucht ein Magischer Bus nur eine Verbindung und Schnittstelle pro System. Das bedeutet, dass dadurch die Anzahl der Schnittstellen für `n` Systeme um den Faktor `n-1` reduziert wird (also ein Faktor von 9 für 10 Systeme und ein Faktor von 49 für 50 Systeme).«

Beeindruckt von diesen Zahlen stimmte der CEO dieser neuen Technik zu. Und die Technik wurde in aller Munde gepriesen, sodass im ganzen Land die Liedermacher und Barden Lieder über den Magischen Bus schrieben. Das bekannteste wurde von einer Band mit dem Namen »The Who« geschrieben und hieß schlicht »Magic Bus«. Darin gab es zum Beispiel Zeilen wie die folgende (siehe [*MagicBus*] für den kompletten Text des Songs):

Magic Bus – It's a bus-age wonder

Nun mag man zum Abschluss ein »Und wenn sie nicht gestorben sind, dann leben sie noch heute« erwarten, aber die Geschichte ist noch nicht zu Ende. Nachdem die Firma den Magischen Bus in Betrieb genommen hatte, wurde es in der Tat

sehr leicht, Systeme zu verbinden, und somit gab es keine Hindernisse mehr, kräftig zu wachsen. Und das System wuchs und wuchs und wuchs ..., bis eines Tages nichts mehr ging.

Was war geschehen?

Es stellte sich heraus, dass der Magische Bus nicht in der Lage war, die Abhängigkeiten zwischen den einzelnen Systemen zu verwalten. Während man in der Vergangenheit derartige Abhängigkeiten implizit an der individuellen Verbindung erkennen konnte, war dies beim Bus nicht mehr möglich. Fing ein System an, Dinge zu ändern, verursachte dies sofort Probleme in anderen Systemen. Dabei stellte man mitunter zur allgemeinen Überraschung überhaupt erst fest, dass eine Abhängigkeit bestand. Am Ende hing jedes System von jedem anderen System ab. Und als das Risiko, irgendetwas zu ändern, zu groß wurde, entschied sich die Firma dafür, das System so zu lassen, wie es war. Doch Stillstand bedeutet Tod, und ein Jahr später war die Firma nicht mehr im Geschäft.

Natürlich gab es genug Warnsignale. Der aufmerksame Zuhörer des Songs von The Who wird zum Beispiel am Ende des Songs eine verklausulierte Anspielung hören, die auf das Problem hinwies:

Every day you'll see the dust
as I drive my baby in my Magic Bus

Zu deutsch:

Jeden Tag wird man den Nebel sehen,
wenn ich mein Geschäft im Magischen Bus betreibe

Und wenn die Konkurrenten nicht gestorben sind, dann leben sie noch heute.

1.3 Was man vom Magischen Bus lernen kann

In diesen Zeiten preisen wir oft das Wunder vom Bus-Zeitalter (»bus-age wonder«). Auch wenn die Idee eines IT-Busses relativ alt ist, hat es in den letzten Jahren eine Renaissance dieser Idee gegeben. Es fing an mit der Einführung eines »Enterprise-Application-Bus« (EAI-Bus) und wurde später durch einen »Enterprise-Service-Bus« (ESB) abgelöst. Letzterer ist so wichtig geworden, dass es sogar eine öffentliche Auseinandersetzung darüber gab, wer den ESB erfunden hat (siehe [*ESB Inventor*]).

Ein Bus steht für hohe Interoperabilität. Die Idee dahinter ist, dass man jedes System nur an den Bus andocken muss, um mit allen anderen Systemen, die am Bus hängen, zu kommunizieren. Dies vereinfacht natürlich die Möglichkeit, miteinander zu kommunizieren, erheblich, aber wie die vorherige Geschichte zeigt, kann dieser Ansatz auch Nachteile haben. Konnektivität führt zu Chaos, wenn keine entsprechenden Strukturierungen eingeführt werden. Aus diesem Grund wurden irgendwann globale Variablen durch Prozeduren und Geschäftsobjekte

durch Module und Komponenten ersetzt. Und dieses Chaos wird sich wieder einstellen, wenn wir einen Bus einfach unstrukturiert einführen.

Deshalb ist die erste Botschaft dieses Buchs, dass für große Systeme mehr als nur Interoperabilität notwendig ist. Man braucht Strukturen, die durch technische und organisatorische Regelungen und Patterns eingeführt werden. Hohe Interoperabilität muss von wohldefinierten Schnittstellen, Strukturen und Prozessen begleitet werden. Wer dies zu spät erkennt, ist schnell nicht mehr im Markt.

1.4 Die Geschichte von SOA

Es ist überraschend schwierig herauszufinden, wer eigentlich den Begriff »SOA« erfunden hat. Roy Schulte von Gartner gab mir die exakten Details 2007 in einem privaten Gespräch:

> *Alexander Pasik, ein früherer Gartner-Analytiker, erfand den Begriff »SOA« für eine Middleware-Schulung, die er 1994 gab. Das war bevor XML oder Web-Services erfunden worden waren, aber die Grundprinzipien von SOA haben sich seitdem nicht geändert.*
>
> *Pasik suchte nach einem neuen Begriff, weil »Client/Server« dabei waren, ihre ursprüngliche Bedeutung zu verlieren. In der Industrie wurden »Client/Server« zunehmend mit verteiltem Computing zwischen einem PC und einem anderen Computer gleichgesetzt. Auf dem »Client«-Desktop lief typischerweise das User-Interface und das meiste an Business-Logik. Auf den »Server«-Backends liefen die Datenbank-Management-Systeme, wurden Daten gespeichert und lief manchmal auch ein gewisses Maß an Business-Logik. In diesem Kontext wurden »Client« und »Server« also mehr oder weniger mit Hardware gleichgesetzt. Es wurde irrelevant, dass die Software auf dem Frontend-PC manchmal der Server-Software im ursprünglichen Sinne von Client/Server entsprach. Um die Verwirrung zwischen der alten und neuen Bedeutung von Client/Server zu vermeiden, bemühte Pasik den Begriff »Server-Orientierung«, als er Entwickler dazu ermutigte, SOA-basierte Fachanwendungen zu entwerfen.*

Die Gartner-Analytiker Roy W. Schulte und Yefim V. Natis veröffentlichten die ersten Reports über SOA 1996 (siehe [*Gartner96*] und [*Gartner03*] für Details).

Das eigentliche Momentum für SOA wurde durch Web-Services ausgelöst, die, ursprünglich von Microsoft getrieben, 2000 öffentlich relevant wurden (siehe [*WS-History*] für Details zur Geschichte von Web-Services). In [*Gartner03*] heißt es dazu:

Auch wenn Web-Services nicht unbedingt mit SOA gleichzusetzen sind und nicht jede Form von SOA auf Web-Services basiert, gibt es doch eine wichtige Beziehung zwischen beiden Technologien, und sie haben sich gegenseitig beeinflusst: Das Momentum von Web-Services macht SOA zum Mainstream für Anwender, und die »Best-Practice«-Architektur von SOA wird dazu beitragen, dass Web-Services-Initiativen erfolgreich sind.

Es dauerte nicht lange und andere Firmen und Hersteller machten mit (einschließlich der führenden IT-Hersteller wie IBM, Oracle, HP, SAP und Sun). Es gab Geld zu verdienen, indem man die Idee erläuterte oder neue Konzepte und Tools verkaufte (oder alte Konzepte und Tools mit neuen Bezeichnungen versah). Die Zeit war reif für diese Ansätze, denn Firmen waren zunehmend auf der Suche nach Möglichkeiten, ihr Business mit anderen Systemen, Abteilungen und Firmen zu verknüpfen (im Zuge der aufkommenden »B2B«-Bewegung[2]).

Später wurde SOA als Konzept für zukünftige Software gepriesen. Gartner schrieb zum Beispiel 2005 in [*Gartner05*]:

2008 wird SOA die Basis von 80 % aller Entwicklungsprojekte sein.

Die Zeit wird zeigen, ob dies zutrifft (was auch eine selbsterfüllende Prophezeiung sein kann, wenn alle Firmen wegen dieser Vorhersage auf SOA setzen).

Jeder Hype erzeugt aber natürlich auch Kritik. Diese wird natürlich auch dadurch forciert, weil zu viele auf den Zug aufspringen und mitunter das Blaue vom Himmel herunter versprechen. Hier zum Beispiel ein Zitat von Grady Booch, einem der Väter von UML und derzeitiger IBM »Fellow«, das in seinem Blog von März 2006 nachgelesen werden kann (siehe [*Booch06*]):

Meine Einschätzung der gesamten SOA-Bewegung fällt etwas nervöser aus als bei den meisten anderen Dingen, die ich gesehen habe. Zu sehr wird mir der Eindruck erweckt, es handle sich um das Beste seit Erfindung der Lochkarten. SOA scheint wohl nicht nur Deine Firma agiler und innovativer zu machen, sondern auch dazu zu führen, dass Teenager wieder mit Dir reden und Du ein besserer Liebhaber wirst (oder besser »kommst«). Zusätzlich wird der Eindruck erweckt, dass all das ohne großen Aufwand möglich ist: Kratze zusammen, was Du an Werten hast, pflanze hier und da ein paar Services und verdrahte sie, und plötzlich ist alles virtualisiert, automatisiert und dienstleistungsorientiert.

Welch ein Unsinn.

Grady Booch hat Recht. Der entscheidende Punkt ist, dass SOA eine *Strategie* ist. Mit dieser Strategie kann man Zeit und Aufwand sparen, man braucht aber auch Verständnis und Erfahrung, um SOA richtig einzuordnen und angemessen

[2] B2B bedeutet »business to business«, steht also für Kommunikation zwischen zwei fachlichen Systemen.

einzusetzen. Glücklicherweise gibt es SOA inzwischen so lange, dass es zumindest einige signifikante Erfahrungen gibt, die über einfache Prototypen und vage Vorstellungen der Integration von zig Systemen mit Hunderten von Services hinausgehen. Diese Erfahrungen werde ich in diesem Buch präsentieren.

1.5 SOA in fünf Folien

Dieses Buch wird etliche Aspekte von SOA aus der Praxis diskutieren. Das bedeutet, dass wir deutlich tiefer in die Materie einsteigen als ein typischer »fünf Folien«-Überblick, der häufig zu der Frage führt, warum das Ganze eigentlich so kompliziert und/oder bemerkenswert ist.

Aber auch ich möchte mit meinen fünf wichtigsten Folien zu SOA beginnen. Diese Folien bergen natürlich ebenso die Gefahr einer zu großen Vereinfachung. Der Teufel steckt bekanntlich im Detail. Aber schon diese fünf Folien mögen ein wenig anders aussehen als Folien, die einfach nur für SOA werben.

1.5.1 Folie 1: SOA

Service-orientierte Architektur (SOA) ist ein Paradigma (Denkmuster) für die Realisierung und Pflege von Geschäftsprozessen, die sich über große verteilte Systeme erstrecken. Dabei spielen drei technische Konzepte eine große Rolle: Services, Interoperabilität und lose Kopplung:

- Ein **Service** ist ein Stück in sich abgeschlossene fachliche Funktionalität. Diese Funktionalität kann einfach (Kundendaten lesen oder schreiben) oder komplex (ein Geschäftsprozess für eine Bestellung) sein. Da sich Services auf den geschäftlichen Nutzen einer Schnittstelle konzentrieren, helfen sie, die übliche Kluft zwischen IT und Fachlichkeit zu überbrücken.
- Ein **Enterprise-Service-Bus (ESB)** ist die Infrastruktur, die hohe Interoperabilität zwischen verteilten Systemen mit Services ermöglicht. Er macht es aus technischer Sicht einfach, Geschäftsprozesse über verschiedene Systeme abzuwickeln, die dabei unterschiedliche Plattformen und Technologien verwenden.
- **Lose Kopplung** ist ein Konzept zur Reduzierung von Abhängigkeiten zwischen Systemen. Da die Geschäftsprozesse über verschiedene Systeme durchgeführt werden, ist es wichtig, die Auswirkungen von Änderungen und Fehlerfällen zu minimieren. Ansonsten wird das Risiko zu groß, dass der Ausfall eines Systems oder eine Änderung an einem System alle anderen Systeme betrifft. Man beachte, dass man für lose Kopplung aber einen Preis zahlt: Komplexität. Lose gekoppelte Systeme sind schwerer zu entwerfen, zu realisieren und zu pflegen.

1.5.2 Folie 2: Richtlinien und Prozesse

Verteilte Verarbeitung (englisch: distributed processing) ist nicht nur ein technisches Detail. Verteilte Verarbeitung ändert alles. Neue Funktionalitäten sind nicht mehr eine Sache für nur eine Abteilung, sondern sie bestehen aus einer ganzen Reihe von Aufgaben und Änderungen für verschiedene Systeme. Diese Systeme und die dabei involvierten Teams müssen kooperieren.

Dafür braucht man klar definierte Rollen, Richtlinien und Prozesse. Zu den Prozessen gehören unter anderem Lebenszyklen von Services und modellgetriebene Entwicklung. Zusätzlich muss verteilte Entwicklung organisiert werden (von verteiltem Design über verteilte Tests bis hin zu verteilter Fehlerbehandlung).

Das Aufsetzen der dazugehörigen Richtlinien und Prozesse kostet mehr Zeit und Aufwand als die initialen technischen Details. Nach wie vor gilt das, was Fred Brooks schon 1974 in [*Brooks74*] geschrieben hat:

> *Ein Produkt eines Programmsystems kostet 9 Mal so viel wie ein einfaches Programm.*

Ein Faktor von drei kommt hinzu, weil es sich um ein Produkt handelt (also um ein Stück Software, das von verschiedenen Personen gestartet, getestet, repariert und erweitert werden kann). Nochmal ein Faktor von drei kommt hinzu, weil es zu einer Komponente eines Systems wird (also weil Integration und die dazugehörigen Tests hinzukommen). Der zweite Faktor erhöht sich noch, wenn viele Komponenten ins Spiel kommen, wie das bei SOA der Fall ist.

1.5.3 Folie 3: Web-Services

Web-Services sind *eine* Möglichkeit, die technischen Aspekte von SOA zu implementieren (aber vergessen Sie dabei nicht, dass zu SOA sehr viel mehr als nur technische Aspekte gehören).

Web-Services bringen selbst aber auch einige Probleme mit sich. Zunächst sind die Standards bisher nicht ausgereift genug, um wirklich Interoperabilität zu garantieren. Außerdem führt die direkte Anwendung von Web-Services nicht zu einem ausreichendem Maß an loser Kopplung.

Man sollte daher nicht erwarten, dass mit Web-Services alle Probleme gelöst sind. Es müssen vielmehr ausreichende Ressourcen (Zeit und Geld) vorgesehen werden, um die zu erwartenden Probleme zu lösen.

Schließlich sollte man nicht in die Falle tappen, alles nur auf Web-Services auszurichten. Auch Web-Services werden kein endgültiger Standard zur Systemintegration sein. Aus diesem Grund sollten Web-Services bei SOA erst dann eine Rolle spielen, wenn es zur Umsetzung in einer konkreten Infrastruktur kommt.

1.5.4 Folie 4: SOA in der Praxis

L. Berra hat richtig gesagt:

> *In der Theorie sind Theorie und Praxis das Gleiche.*
> *In der Praxis sind sie es nicht.*

Dies gilt natürlich auch für SOA. Theoretische Einsparpotenziale stellen sich in der Praxis als nicht ganz so umfassend heraus, wenn plötzlich Aspekte wie Performance und Sicherheit ins Spiel kommen.

Hinzu kommt, dass SOA eine Strategie zur Wartung und Weiterentwicklung existierender Systeme darstellt. Das bedeutet, dass Stabilität und Rückwärtskompatibilität eine Rolle spielen.

Schließlich gilt auch hier wie überall in der IT, dass keine zwei Systeme gleich sind. Eine Konsequenz daraus ist, dass man SOA nicht kaufen kann. Um die passende Form von SOA zum Leben zu erwecken, braucht man also einen inkrementellen und iterativen Ansatz.

Dabei spielt es im Übrigen auch gar keine Rolle, ob das, was dann herauskommt, wirklich SOA ist. Entscheidend ist, dass die eingeführten IT-Konzepte und -Lösungen im spezifischen Kontext und unter den gegebenen Anforderungen angemessen sind.

1.5.5 Folie 5: SOA-Governance und Management-Unterstützung

Der vermutlich wichtigste Aspekt zur erfolgreichen Umsetzung von SOA besteht darin, den richtigen Ansatz und das richtige Maß an Steuerung (»Governance«) zu finden:

- Man braucht zwar ein zentrales Team, das die Grundlagen schafft, das eigentliche Ziel besteht aber darin, Zentralismus abzuschaffen (nur so skalieren große Systeme). Man muss also das richtige Maß zwischen Zentralismus und Dezentralismus finden.
- Man braucht die richtigen Leute. Große Systeme sind anders als kleine Systeme. Daher sind Personen erforderlich, die bereits Erfahrung mit großen Systemen haben. Wenn Konzepte aus pragmatischen Gründen nicht skalieren, werden unerfahrene Personen sonst versuchen, diese pragmatischen Gründe zu bekämpfen, anstatt sie als potenzielle Eigenschaft großer Systeme anzusehen und zu akzeptieren. Zusätzlich neigen zentrale und querschnittliche Teams dazu, »Elfenbeintürme« zu werden (realitätsfern zu denken). Sie müssen daher durch die Anforderungen der fachlichen Entwicklungsteams gesteuert werden. Im Grunde müssen sie sich selbst auch als Dienstleister (Service-Erbringer) betrachten.

- Eins nach dem anderen. Es macht keinen Sinn, mit der Verwaltung aller Services anzufangen. Service-Management ist erst dann notwendig, wenn man ausreichend viele Services hat. Genauso wenig macht es Sinn, erst die komplette Infrastruktur zu entwickeln oder erst alle Services zu entwerfen. Dies alles muss sich nach und nach gemeinsam entwickeln und herauskristallisieren. Und während dies passiert, hat man genug mit den Problemen zu tun, die jeweils anliegen.
- Nicht zuletzt benötigt man die Unterstützung des Top-Managements. SOA ist eine Strategie, die Auswirkungen auf eine Firma als Ganzes hat. Sorgen Sie also dafür, dass das Top-Management hinter dem Konzept steht, die dazu notwendigen Entscheidungen trifft und genug Zeit und Geld bereitstellt. Der entscheidende Punkt ist nicht, kurzfristig möglichst viel Ressourcen zur Verfügung zu haben, um möglichst viel leisten zu können, der entscheidende Punkt ist Geduld (was natürlich nicht zu blindem Vertrauen führen sollte). Wenn SOA-Budgets auf halbem Wege gekappt werden, kann das zum Desaster führen, weil alles nur noch komplizierter als vorher geworden ist.