SOA und Softwarequalität

Andreas Göb

Technische Universität München

Institut für Informatik
der Technischen Universität München

SOA und Softwarequalität

Andreas Göb

Vollständiger Abdruck der von der Fakultät für Informatik der Technischen Universität München zur Erlangung des akademischen Grades eines

Doktors der Naturwissenschaften (Dr. rer. nat.)

genehmigten Dissertation.

Vorsitzender:	Univ.-Prof. Dr. Hans Michael Gerndt
Prüfer der Dissertation:	
1.	Univ.-Prof. Dr. Dr. h.c. Manfred Broy
2.	Univ.-Prof. Dr. Mathias Weske, Universität Potsdam

Die Dissertation wurde am 20.02.2013 bei der Technischen Universität München eingereicht und durch die Fakultät für Informatik am 21.06.2013 angenommen.

Abstract

Service-oriented Architecture (SOA) has been established as an architectural style for distributed applications, in particular in the enterprise. Rapidly increasing complexity of software systems imposes new requirements not only on the architecture, but also on the quality of software products. Despite the great importance of both SOA and quality for the future of software development, there is very little documented knowledge on relationships between them. To overcome this research gap, this dissertation proposes a quality model for service-oriented architectures based on both scientific literature and requirements from the software industry.

The dissertation provides the following contributions to theory and practice: First, impacts of SOA on software quality are identified based on the state of research and practice. Second, an extensive literature review reveals general architectural properties and their impact on software quality. Third, a quality model is defined, which addresses both of these dimensions and allows for measurement-based quality analysis. Fourth, using this quality model, the promises with regard to quality that are often associated with SOA are critically analyzed and supported with evidence or counter-statements. Fifth, for practical applicability, the model can be extended domain- or project-specific and integrated with quality analysis tools.

The quality model has been evaluated in interviews with SOA experts from academia and industry. As a result, the model was considered well-structured and understandable. The experts also found that the model adequately addresses the most important architectural properties. The SOA quality model can be used to analyze quality goals with regard to influencing factors originating from the SOA principles as well as others caused by further architectural properties. With this information one can argue to which degree SOA imposes positive effects on software quality and whether they might be reduced by not adhering to other design principles.

Moreover, the quality model provides a foundation for standardized and comparable quality assessment. The gain in reproducibility caused by this fact is tightly related to a reduction of efforts for quality analysis, because the processing of measurement results can be performed automatically by existing tools. This also leads to a reduction of discontinuities in the quality management of software projects. During architecture definition, tools can be leveraged that are also used for the quality analysis of source code in later project stages. In the end, this consolidation of quality analysis tools can lead to cost reductions in software quality management.

Kurzfassung

Als Architekturstil für verteilte Anwendungen haben sich vor allem im Geschäftsumfeld serviceorientierte Architekturen (SOAs) etabliert. Die rapide steigende Komplexität von Softwaresystemen stellt nicht nur neue Anforderungen an die Architektur, sondern auch an die Qualität von Software. Trotz der großen Bedeutung der Kernthemen SOA und Qualität für die Zukunft der Softwareentwicklung existiert wenig dokumentiertes Wissen über die Zusammenhänge zwischen diesen. Zur Schließung dieser Forschungslücke wird ein Qualitätsmodell für serviceorientierte Architekturen entwickelt, das wissenschaftlich begründet ist und sowohl auf Literaturarbeit als auch auf Anforderungen aus der Software-Industrie basiert.

Die Arbeit leistet die folgenden Beiträge zu Theorie und Praxis: Aus dem Stand von Forschung und Technik werden Einflüsse von SOA auf die Qualität einer Software identifiziert. Als zweiten Beitrag erarbeitet ein umfangreiches Literaturreview allgemeine Architektureigenschaften und ihre Auswirkung auf die Softwarequalität. Auf Basis dessen wird drittens ein Qualitätsmodell definiert, das diese Dimensionen abbildet und eine Qualitätsanalyse mit Hilfe von Messungen erlaubt. Viertens werden mit dem Qualitätsmodell Versprechungen hinsichtlich Qualität, die oft mit SOA einhergehen, analysiert und bestätigt oder widerlegt. Als Beitrag zur Praxis lässt sich das Modell in bestehende Qualitätsanalysewerkzeuge integrieren und domänen- oder projektspezifisch erweitern.

Das Qualitätsmodell wurde durch Befragungen von SOA-Experten aus Industrie und Forschung evaluiert. Dabei wurde das Modell als gut strukturiert und verständlich bewertet. Außerdem wurde deutlich, dass das Modell die wichtigsten Eigenschaften der Architektur angemessen abbildet. Das Qualitätsmodell kann verwendet werden, um ausgehend von einem Qualitätsziel zu analysieren, welche Einflüsse sich aufgrund der SOA-Prinzipien ergeben, und welche weiteren Einflüsse auf dieses Qualitätsziel von darüber hinaus gehenden Eigenschaften der Architektur bestehen. Damit wird deutlich, inwieweit SOA an sich bereits positive Auswirkungen auf die Softwarequalität besitzt und ob diese durch Nichtbeachtung weiterer Design-Eigenschaften möglicherweise relativiert werden.

Das Qualitätsmodell bildet außerdem die Grundlage für standardisierte und vergleichbare Qualitätsbewertungen. Damit trägt das Qualitätsmodell zur Reproduzierbarkeit von Qualitätsanalysen bei. Diese geht mit einer Aufwandsreduktion einher, da die Aufbereitung der Messwerte durch Anbindung bestehender Werkzeuge automatisiert erfolgen kann. Außerdem werden so Medienbrüche im Qualitätsmanagement von Softwareprojekten vermieden. Bereits auf Ebene der Architektur können Werkzeuge zum Einsatz kommen, welche auch für die Analyse des Quellcodes verwendet werden. Durch eine Konsolidierung von Qualitätsanalysewerkzeugen kann letztlich eine Kosteneinsparung im Qualitätsmanagement erreicht werden.

Abkürzungsverzeichnis

ABQM	Activity-Based Quality Model
ACM	Association for Computing Machinery
ADL	Architecture Description Language
ARID	Active Reviews for Intermediate Designs
ATAM	Architecture Tradeoff Analysis Method
AWT	Abstract Window Toolkit
BISA	Business Information Systems' Analysis
BPEL	Business Process Execution Language
BPM	Business Process Management
COCOMO	Constructive Cost Model
CPU	Central Processing Unit
DSL	Domain-Specific Language
EA	Enterprise Architecture
EU	Europäische Union
FMC	Fundamental Modeling Concepts
GQM	Goal-Question-Metric
HTML	Hypertext Markup Language
IEC	International Electrotechnical Commission
IEE	Institution of Electrical Engineers
IEEE	Institute of Electrical and Electronics Engineers
ISO	International Organization for Standardization
IT	Information Technology
MDSD	Model-Driven Software Development
MOF	Meta Object Facility
OASIS	Organization for the Advancement of Structured Information Standards
OMG	Object Management Group
OO	Object-Orientation
QoS	Quality of Service
SAAM	Software Architecture Analysis Method
SE	Software Engineering
SLA	Service Level Agreement
SLOC	Source Lines of Code
SOA	Service-oriented Architecture
SoaML	Service oriented architecture Modeling Language
SOC	Service-oriented Computing
TAM	Technical Architecture Modeling
UML	Unified Modeling Language

W3C	World Wide Web Consortium
WSDL	Web Services Description Language
XSD	XML Schema Definition
XML	Extensible Markup Language

Inhaltsverzeichnis

1 Einleitung

1.1 Motivation

Die Größe von Softwaresystemen (gemessen in Zeilen Programmcode) wächst Studien zufolge exponentiell über die Zeit [61, 72]. Dieser Umstand bringt eine steigende Notwendigkeit neuartiger Integrationslösungen mit sich. Innerhalb der letzten Jahre haben sich serviceorientierte Architekturen (SOAs) als ein ernstzunehmender Ansatz für die Implementierung von kollaborativen Infrastrukturen etabliert [37]. Oft werden diese Infrastrukturen über Firmengrenzen hinweg im Internet realisiert, was zum Entstehen der Bezeichnung *Internet der Dienste* geführt hat, die oft im Zusammenhang mit der zukünftigen Nutzung von Software gebraucht wird[1].

Die zentrale Rolle von SOA im Umfeld von Geschäftsanwendungen wird in der Literatur oft hervorgehoben. So ist z. B. bei Masak [96, S. 132] zu lesen: „Service Oriented Architectures werden auf Dauer das Rückgrat des B2B-Geschäfts darstellen, da sie die entsprechende Flexibilität besitzen, sich auf rasche Veränderungen anzupassen.“ Insbesondere ist zu beobachten, dass die Anzahl der Services, welche über das Internet angeboten werden, in der jüngsten Vergangenheit rapide angestiegen ist. So sind aus dem Jahr 2004 Angaben über etwa 1200 Dienste zu finden [20], während für das darauf folgende Jahr bereits von 20 000 Diensten berichtet wird [87]. Dass auch nach diesem ersten starken Anstieg weiterhin eine rapide Zunahme der Dienste im Internet zu verzeichnen ist, wird unter anderem in einer Studie von Al-Masri und Mahmoud [97] deutlich, die für das Jahr 2007 eine Zunahme öffentlich verfügbarer Web Services um 131 % im Vergleich zum Vorjahr verzeichnen.

Eine solche Steigerung des Angebots an Services führt langfristig dazu, dass eine Vielzahl von Diensten existiert, welche sich in ihrer Funktionalität nicht oder nur geringfügig unterscheiden und Funktionalität als differenzierender Faktor an Bedeutung verliert. Hier bietet die Produktqualität eine entscheidende Möglichkeit der Differenzierung. Auch die steigende Komplexität von Softwareprodukten stellt neue Anforderungen an die Qualitätssicherung: Wie Garbani [61] illustriert, können die derzeitigen Qualitätssicherungsmaßnahmen in Zukunft nicht mehr die nötige Produktqualität sicherstellen. Eine gleich bleibende relative Anzahl von Qualitätsproblemen innerhalb der Software führt bei exponentiell steigender Komplexität zu einer deutlich steigenden absoluten Anzahl wahrgenommener Qualitätsprobleme durch den Benutzer.

Sowohl SOA als auch der Softwarequalität werden also in Zukunft besondere Bedeutung beigemessen. Auf dieser Grundlage ist es besonders relevant, Zusammenhänge zwischen den beiden zu identifizieren und zu untersuchen.

[1] siehe auch `http://future-internet.eu`

1.2 Problemstellung

Die Einführung serviceorientierter Architekturen wird oft mit dem Argument besserer Integration (siehe z. B. Nissen, Petsch und Schorcht [108]) und Interoperabilität begründet. In welchem Maße eine SOA-Anwendung diesem Anspruch gerecht wird, wird in der Literatur kontrovers diskutiert. So zitieren Lawler und Howell-Barber [91] eine Umfrage von *Information Week Research*, in der 55 % der Befragten angaben, einer der Gründe für die SOA-Einführung in ihrem Unternehmen sei die bessere Integration mit Geschäftspartnern gewesen. Gleichzeitig gaben allerdings auch 35 % der Befragten an, eines der Probleme bei der SOA-Einführung sei gewesen, dass der erwartete Integrationsgrad nicht erreicht worden sei.

Welche Auswirkung die Entscheidung für SOA als zugrundeliegender Architekturstil auf die übrigen Aspekte der Softwarequalität hat, ist ebenfalls oft nicht eindeutig nachvollziehbar. Svahnberg und Wohlin [132] argumentieren sogar, es existiere generell sehr wenig belastbares Wissen darüber, welche Qualitätsattribute durch verschiedene Architekturstile unterstützt oder negativ beeinflusst werden. Vorgespräche mit erfahrenen Architekten und Entwicklern serviceorientierter Softwaresysteme haben durchaus kontroverse Auffassungen zutage gefördert: Der Standpunkt, dass SOA sich positiv auf die Gesamtqualität auswirkt ist genauso vertreten wie der, dass die negativen Auswirkungen von SOA auf die Produktqualität in Summe überwiegen. Das allein zeigt, dass eine detaillierte Betrachtung und fundierte Untersuchung notwendig ist.

Die Bewertung von Softwarearchitekturen ist im Allgemeinen mit enormem Aufwand verbunden, da viele anerkannte Verfahren auf der manuellen Analyse von Szenarien basieren. Insbesondere im Hinblick darauf, dass Services im Internet stark inkrementell entwickelt werden und häufigen (wenn auch vergleichsweise kleinen) Änderungen unterworfen sind, erscheint ein solches Vorgehen auf lange Sicht nicht praktikabel. Zudem erschweren solche in hohem Maße individuellen Ansätze zur Architekturbewertung die Vergleichbarkeit von Bewertungsergebnissen, etwa um die am besten bewerteten Architekturen einer Menge von Projekten zu identifizieren und diese miteinander in Bezug zu setzen.

Für den Software-Entwicklungsprozess bedeutet dies außerdem, dass zwischen der Qualitätsbewertung in der Architekturphase und jener in der Implementierungsphase fundamental unterschiedliche Vorgehensweisen zum Einsatz kommen und auch die Ergebnisse in unterschiedlicher Form vorliegen und kommuniziert werden. Es existieren also Medienbrüche, welche eine kontinuierliche Überwachung der Softwarequalität erschweren und damit die Kosten des Qualitätsmanagements negativ beeinflussen.

1.3 Forschungsfragen

Aus der oben beschriebenen Forschungslücke leitet sich die für diese Dissertation zentrale Frage ab, welche spezifischen Qualitätseigenschaften SOA besitzt und wie diese im Rahmen eines geeigneten Qualitätsmodells dargestellt werden können. Konkret lässt

sich diese Fragestellung im komplexen Spannungsfeld von SOA und Softwarequalität in fünf Forschungsfragen unterteilen:

1. Welchen Einfluss besitzt SOA auf die Qualitätsattribute einer Software?

2. Lässt sich durch SOA eine – häufig versprochene – Erhöhung des Qualitätsniveaus, insbesondere hinsichtlich Flexibilität, Interoperabilität und Uniformität, tatsächlich nachweisen?

3. Welche nicht SOA-spezifischen Eigenschaften einer Architektur besitzen einen Einfluss auf ihre Qualität, und wie lassen sich diese adäquat abbilden?

4. Wie sieht ein Qualitätsmodell aus, das diese beiden Sichtweisen vereint, die identifizierten Eigenschaften messbar macht und damit die Grundlage für reproduzierbare Qualitätsanalysen bildet?

5. Wie kann das Qualitätsmodell in bestehende Infrastrukturen zur Qualitätsanalyse eingebunden werden, um die Konsistenz im Qualitätsmanagement zu erhöhen?

1.4 Beitrag

Zur Beantwortung dieser Fragen wird ein Qualitätsmodell für serviceorientierte Architekturen entwickelt, das wissenschaftlich begründet ist und sowohl auf Literaturarbeit als auch auf Anforderungen aus der Software-Industrie basiert. Die Arbeit leistet im Wesentlichen fünf Beiträge zu Theorie und Praxis:

- Eine Analyse sowohl von wissenschaftlicher Literatur als auch von Befragungen in der Industrie fasst den Stand von Forschung und Technik zusammen und liefert eine Analyse der Einflüsse des Architekturstils SOA auf die Qualität einer Software.

- Ein zweites umfangreiches Literaturreview analysiert empirische Studien bezüglich allgemeiner Architektureigenschaften, Verfahren zu ihrer Messung und ihre Auswirkung auf die Softwarequalität.

- Auf Basis dieser Analysen wird ein Qualitätsmodell definiert, das diese beiden Dimensionen abbildet und eine Qualitätsanalyse auf Basis von Messungen erlaubt.

- Mit Hilfe des Qualitätsmodells werden Versprechungen hinsichtlich Qualität, die oft mit SOA einhergehen, analysiert und bestätigt oder widerlegt.

- Als Beitrag zur praktischen Anwendbarkeit lässt sich das Qualitätsmodell in bestehende Qualitätsanalysewerkzeuge integrieren und domänen- oder projektspezifisch erweitern.

Um sicherzustellen, dass die erarbeiteten Konzepte sowohl die gestellten Forschungsfragen adressieren als auch für die Praxis relevant und anwendbar sind, erfolgt eine Evaluation durch Befragungen von SOA-Experten aus Industrie und Forschung. Dabei wird das Qualitätsmodell hinsichtlich seiner Vollständigkeit, Angemessenheit und Verständlichkeit beurteilt. Diese Vorgehensweise ist im explorativen Charakter der Arbeit und der typischerweise langen Laufzeit von Qualitätsmodell-Fallstudien begründet, insbesondere bei Vorher-Nachher-Vergleichen von Projekten in der Softwareentwicklung.

Die Arbeit liefert somit einen wertvollen Beitrag, die für sich jeweils wissenschaftlich gut untersuchten Themengebiete SOA und Softwarequalität zueinander in Bezug zu setzen. Gleichzeitig wird die industrielle Relevanz der Ergebnisse in den Expertenbefragungen deutlich.

1.5 Inhaltsüberblick

Der übrige Teil dieser Dissertation ist wie folgt gegliedert: Kapitel 2 gibt eine Einführung in grundlegende Konzepte aus den Bereichen der Softwarequalität, Softwarearchitektur und SOA. Darüber hinaus wird der Stand der Wissenschaft in Hinblick auf Qualitätsmodelle für SOA vorgestellt und kritisch betrachtet. In Kapitel 3 wird eine Literaturanalyse dokumentiert, welche für Software allgemein nachgewiesene Zusammenhänge zwischen bestimmten Eigenschaften der Architektur und der Softwarequalität identifiziert, gruppiert und in einer Art und Weise aufarbeitet, dass diese für das SOA-Qualitätsmodell angepasst werden können. Das erarbeitete Qualitätsmodell wird in Kapitel 4 beschrieben. Dazu wird zunächst die grundlegende Struktur des Modells entwickelt. Zur eindeutigen Beschreibung der Artefakte eines SOA-Systems und ihrer Beziehungen wird ein Referenzmodell von SOA erarbeitet. Anschließend werden Faktoren und Maße definiert, mittels derer die Qualität einer SOA beschrieben und bewertet werden kann. Kapitel 5 beschreibt eine Studie zur Vollständigkeit und Angemessenheit des SOA-Qualitätsmodells. Außerdem werden mögliche Einsatzgebiete für das Qualitätsmodell präsentiert, indem es zur Beantwortung der Forschungsfragen verwendet wird. Darüber hinaus wird das Qualitätsmodell exemplarisch zur Bewertung einer SOA eingesetzt und die Bewertungsergebnisse mit der von Experten wahrgenommenen Qualität des Systems verglichen. Kapitel 6 fasst die wichtigsten Inhalte der Dissertation zusammen und zeigt Möglichkeiten für zukünftige Forschung auf dem Gebiet von SOA und Softwarequalität auf.

2 Grundlagen und verwandte Arbeiten

2.1 Einleitung

Dieses Kapitel stellt Grundlagen aus den Bereichen SOA und Softwarequalität zusammen, welche für die nachfolgenden Kapitel von Bedeutung sind. Im Zuge dessen werden auch Arbeiten vorgestellt, in denen Qualitätsmodelle für SOA vorgeschlagen werden.

Abschnitt 2.2 führt zunächst die grundlegenden Konzepte und Bezeichnungen aus den Bereichen Qualität, Softwarequalität und Qualitätsmodellierung ein und gibt einen Überblick über Qualitätsomdelle für Software. In Abschnitt 2.3 werden die Begriffe Architektur, Softwarearchitektur und SOA eingeführt sowie in Bezug auf diese Dissertation abgegrenzt. Abschnitt 2.4 stellt den Zusammenhang zwischen den grundlegenden Charakteristiken von SOA und der Softwarequalität her. Dabei werden zunächst ausgehend von den SOA-Charakteristiken Schlussfolgerungen bezüglich verschiedener Qualitätsattribute gezogen. Darüber hinaus werden die Argumente reflektiert, die üblicherweise für die Einführung von SOA herangezogen werden. In diesem Zusammenhang werden auch bestehende Qualitätsmodelle für SOA vorgestellt, abgegrenzt und kritisch gewürdigt. Abschnitt 2.5 fasst das Kapitel zusammen.

2.2 Softwarequalität

Um einen umfassenden Überblick über verwandte Arbeiten und den Stand der Wissenschaft und Praxis im Umfeld der vorliegenden Arbeit zu erlangen, müssen zunächst einige Begriffe definiert und gegeneinander abgegrenzt werden. Die folgenden Abschnitte geben daher einen Abriss über die relevanten Konzepte in den Bereichen Softwarequalität und serviceorientierte Architekturen.

2.2.1 Qualität

Der Qualitätsbegriff wurde lange vor dem Zeitalter der Software geprägt und in den achtziger Jahren des letzten Jahrhunderts durch Garvin [62] für den Bereich der Produktqualität umfassend definiert und charakterisiert. Garvin beschreibt fünf Sichten auf Produktqualität.

Die *transzendente Sicht* beschreibt Qualität als eine Eigenschaft, die unabhängig von anderen Eigenschaften des Produkts vorhanden ist oder nicht. Sie lässt sich nicht konkret definieren, sondern nur als Ganzes erkennen. In der *Produktsicht* lässt sich Qualität aus Eigenschaften des Produktes ableiten. Dabei unterscheidet die Menge an bestimmten Inhaltsstoffen oder anderen Eigenschaften ein Produkt hoher Qualität

von einem Produkt geringerer Qualität. Aus *Nutzersicht* ist Qualität charakterisiert durch den Grad der Erfüllung von Kundenwünschen oder -anforderungen. Je besser ein Produkt an den konkreten Bedarf eines Benutzers angepasst ist, desto höher wird seine Qualität bewertet. Die *Herstellersicht* bildet Qualität auf die Erfüllung spezifizierter Anforderungen ab. Je genauer das fertige Produkt die Entwürfe und Spezifikationen (oder auch gesetzliche Anforderungen) erfüllt, desto höher ist seine Qualität aus Sicht des Herstellers. Die *wertebasierte* Sicht schließlich beschreibt den Grad, zu dem das Produkt unter Einbeziehung seiner Kosten für einen bestimmten Zweck geeignet ist. Dabei handelt es sich gewissermaßen um das Preis-Leistungsverhältnis des Produktes.

Aus dieser Übersicht lässt sich unschwer erkennen, dass sich zwischen den einzelnen Betrachtungsweisen Inkonsistenzen ergeben können. Ein Produkt, das sämtliche gesetzlichen und planerischen Vorgaben erfüllt, also aus Herstellersicht von erstklassiger Qualität ist, kann möglicherweise zu einem überhöhten Preis angeboten werden, was ein ungünstigeres Preis-Leistungs-Verhältnis (also nach obiger Definition eine geringe wertebasierte Qualität) zur Folge hätte, oder sogar aus Nutzersicht völlig ungeeignet sein. Sich auf eine dieser Perspektiven zu beschränken reicht also gemeinhin nicht aus, um ein qualitativ hochwertiges Produkt zu erzeugen. Andererseits hat jede Sicht ihre Spezifika, wodurch es nicht möglich ist, sie auf eine einzige Darstellung zu projizieren. Produktqualität muss also als multidimensionales Konzept verstanden werden.

2.2.2 Softwarequalität

Kitchenham und Pfleeger [88] übertragen den Qualitätsbegriff auf die Softwareentwicklung und definieren diesen spezifisch für Software. Dabei orientieren sie sich an den Sichten auf Produktqualität nach Garvin und ergänzen, dass auch die Erhebung von Messwerten stark von der eingenommenen Sichtweise abhänge. So sei beispielsweise für den Benutzer einer Software die Interaktion mit dem fertigen Softwareprodukt maßgeblich, bei der es nicht nur auf Zuverlässigkeit ankomme, sondern auch auf Aspekte wie einfache Benutzbarkeit und Erlernbarkeit.

Grundsätzlich erfordern den Autoren zufolge unterschiedliche Softwaresysteme unterschiedliche Qualitätsaspekte. In Abhängigkeit von den als wichtig empfundenen Qualitäten und der eingenommenen Sichtweise seien wiederum unterschiedliche Messverfahren nötig. In diesem Zusammenhang verweisen die Autoren auf das *Goal-Question-Metric (GQM)*-Framework von Basili und Weiss [24]. Die starke Bedeutung der Messung und Bewertung von Produktqualität wird durch die Aussage bekräftigt, dass die Einhaltung von Prozessen allein noch nicht die Qualität des entwickelten Produktes garantiert.

Ein häufig verwendetes Hilfsmittel zur Analyse von Softwarequalität sind Metriken sowie Regeln, welche von unterschiedlichen Werkzeugen automatisch bestimmt bzw. überprüft werden können. Üblicherweise sind solche Werkzeuge spezifisch für eine Programmiersprache. Beispiele sind FindBugs[1] und PMD[2] für Java, PC-Lint[3] für

[1] `http://findbugs.sourceforge.net/`
[2] `http://pmd.sourceforge.net/`
[3] `http://www.gimpel.com/html/pcl.htm`

C und C++ oder Gendarme[4] für C#. Die Interpretation der von diesen Werkzeugen gelieferten Daten ist allerdings ein kontrovers diskutiertes Thema. Hierzu existieren unterschiedliche Ansätze vom Festlegen absoluter Grenzwerte für Metriken und Regelverletzungen bis hin zu rein vergleichenden Methoden mittels *Benchmarking* [68, 129]. Die Definition absoluter Grenzwerte für Metriken ist wie zuvor erwähnt schwierig, da sich deren Bedeutung je nach Blickwinkel und Zielsetzung ändern kann. Vergleichende Ansätze wirken dem entgegen, indem für jede Qualitätsanalyse eine geeignete Referenz, die sogenannte *Benchmarking-Basis*, ausgewählt werden kann. Eine ausführliche Beschreibung des Ansatzes sowie ein umfassender Überblick über verwandte Ansätze sind in der Arbeit von Gruber [67] zu finden.

2.2.3 Softwarequalitätsmodelle

Ein hilfreiches Mittel zur Unterstützung der Qualitätssicherung in der Softwareentwicklung sind Softwarequalitätsmodelle.

Im Rahmen der vorliegenden Arbeit ist der Begriff *Softwarequalitätsmodell* als Sammelbegriff für alle Arten von Modellen zu verstehen, die dazu verwendet werden, die Qualität eines Softwareproduktes zu *beschreiben*, zu *bewerten* oder *vorherzusagen* [47]. Entlang dieser Dimensionen lassen sich bestehende Arbeiten klassifizieren. Einige Qualitätsmodelle versuchen dabei, Qualität als Ganzes zu beschreiben [28, 52, 100], andere fokussieren bestimmte Qualitätsaspekte [101, 128] oder Qualitätssicherungsmaßnahmen [83].

Waren in der Vergangenheit oft Qualitätsmodelle zu finden, die vor allem eine beschreibende und definitorische Funktion hatten, so haben sich in letzter Zeit auch solche entwickelt, die direkte Anwendbarkeit und Werkzeugunterstützung bieten [46, 105].

Ein üblicher Ansatz bei der Modellierung von Softwarequalität ist die Beschreibung von Aspekten, in denen sich Qualität äußert. Unabhängig davon, welche Repräsentation dieser Aspekte gewählt wird, haben diese üblicherweise den Charakter von Zielen. Um zu beurteilen, inwieweit diese Ziele durch ein Softwaresystem erreicht werden, können Fakten beschrieben werden, welche für das System gelten und dort beobachtbar sind. Dies entspricht der Produktsicht auf Qualität in der Garvin'schen Terminologie. Dieser Ansatz wird auch im Rahmen dieser Dissertation verfolgt. Gemeinsam mit den Maßen zur Quantifizierung der Faktoren wird also eine Dreiteilung deutlich, wie Abbildung 2.1 veranschaulicht.

ISO/IEC 9126 und 25000

Die am weitesten verbreiteten Normen für Softwarequalität sind ISO/IEC 9126 [78] und ISO/IEC 14598 [75]. Erstere beschreibt ein Qualitätsmodell für Software, letztere liefert eine Methodenbeschreibung zur Bewertung der Qualität von Softwareprodukten. Das Qualitätsmodell der ISO/IEC 9126 baut dabei auf den wissenschaftlichen Arbeiten von McCall, Richards und Walters [100] sowie Boehm u. a. [28] vom Ende

[4] `http://www.mono-project.com/Gendarme`

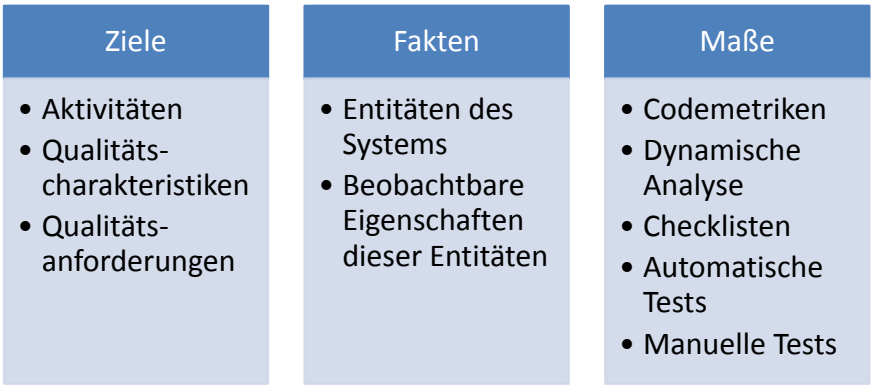

Abbildung 2.1: Abstraktionsebenen in Qualitätsmodellen

der 1970er Jahre auf und definiert eine hierarchisch aufgebaute Menge von Software-Qualitätsattributen. Da der Standard jedoch hinsichtlich der Auswahl und Struktur der Qualitätsattribute als überarbeitungswürdig charakterisiert wurde [82], ist mit ISO/IEC 25000 eine neue Serie von Normen entstanden um den bestehenden Standard abzulösen [76]. Diese enthält neben Referenzmodellen und Definitionen auch eine Beschreibung zum Vorgehen für diese Ablösung. Serviceorientierung ist allerdings auch in diesem Standard nicht explizit vorgesehen.

Prinzipiell unterscheidet der Standard zwischen Produktqualität, Nutzqualität[5] und Datenqualität. Im Rahmen der vorliegenden Arbeit ist vor allem das Produktqualitätsmodell relevant. Dieses besteht aus Charakteristiken und Subcharakteristiken, wobei zwischen diesen eine 1:n-Beziehung besteht. Die Charakteristiken und Subcharakteristiken für Produktqualität sind in Abbildung 2.2 dargestellt.

Aktivitätenbasierte Qualitätsmodelle

Um die hierarchische Gliederung von Qualitäts-Charakteristiken nach semantisch eindeutigen Regeln vorzunehmen, wurde an der Technischen Universität München das Konzept der aktivitätenbasierten Qualitätsmodelle (*Activity-Based Quality Models*, ABQMs) entwickelt [32, 49].

Dabei werden relevante Eigenschaften des Systems oder seiner Umgebung (z. B. auch Fähigkeiten von Personen) durch *Fakten* beschrieben, die sich über *Einflüsse* auf *Aktivitäten* auswirken, die am oder mit dem System durchgeführt werden. Die Kernelemente eines ABQM sind:

Activity Eine Aktivität, die am oder mit dem System durchgeführt werden soll. Aktivitäten können sich aus feingranulareren Aktivitäten zusammensetzen, z. B. [Wartung] aus [Analyse], [Modifikation] und [Testen]. Die Qualität eines Systems hinsichtlich der betrachteten Aktivitäten wird bestimmt durch die Kosten, die für

[5] *Quality in Use*

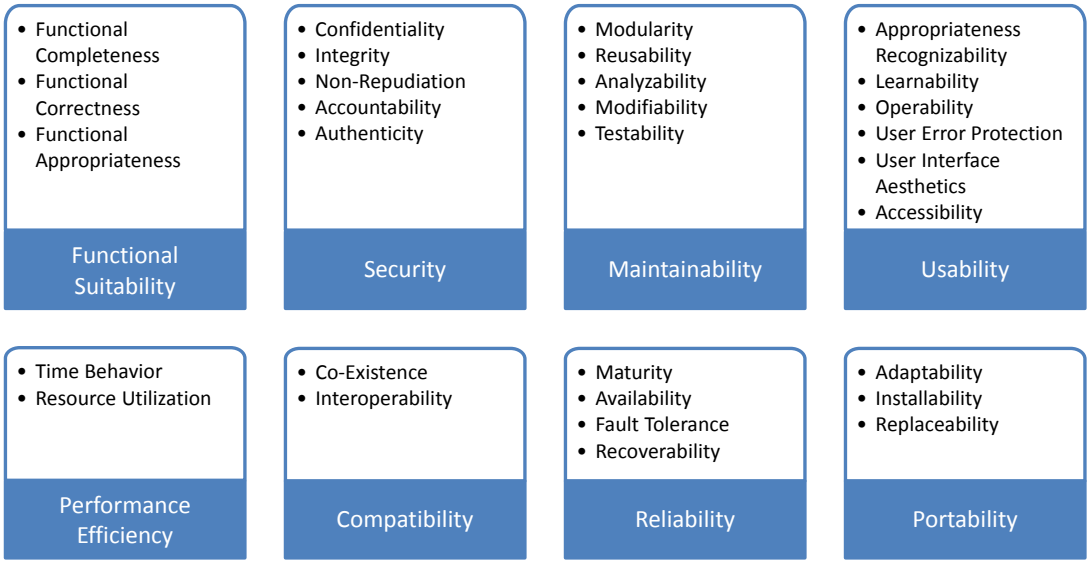

Abbildung 2.2: Qualitäts-Charakteristiken nach ISO/IEC 25010 (Quelle: [77])

die Durchführung dieser Aktivitäten anfallen. Allerdings sind diese Kosten nicht explizit im Modell abgebildet.

Entity Entitäten beschreiben Konzepte innerhalb eines Systems. Dies können sowohl technische Konzepte sein wie ein Ausdruck im Quelltext eines Programms, als auch abstraktere Konzepte wie ein Programmierer.

Attribute Attribute werden verwendet, um Eigenschaften von Entitäten zu beschreiben. Typische Attribute sind z. B. ANGEMESSENHEIT oder EINDEUTIGKEIT.

Fact Ein Tupel [Entity | ATTRIBUTE] wird als Fakt bezeichnet. Gültige Fakten sind diejenigen Tupel, die einen Einfluss auf eine oder mehrere Aktivitäten innerhalb des Modells besitzen. Typische Fakten sind z. B. [Programmierer | ERFAHRUNG] oder [Ausdruck | EINDEUTIGKEIT].

Impact Einflüsse beschreiben die Zusammenhänge zwischen Fakten und Aktivitäten. Dabei wird zwischen positivem und negativem Effekt eines Einflusses unterschieden: [Ausdruck | EINDEUTIGKEIT] $\xrightarrow{+}$ [Analyse]

Abbildung 2.3 zeigt beispielhaft die grafische Darstellung eines ABQM für die Wartung eines Softwaresystems. Der Abbildung liegen einige Einschränkungen zugrunde: Aufgrund der hierarchischen Struktur sowohl der Entitäten als auch der Aktivitäten werden Einflüsse nur auf der jeweils unteren Ebene definiert, was eine Matrixdarstellung erlaubt. Außerdem ist nur das Vorhandensein von Einflüssen, nicht aber deren Effekt dargestellt.

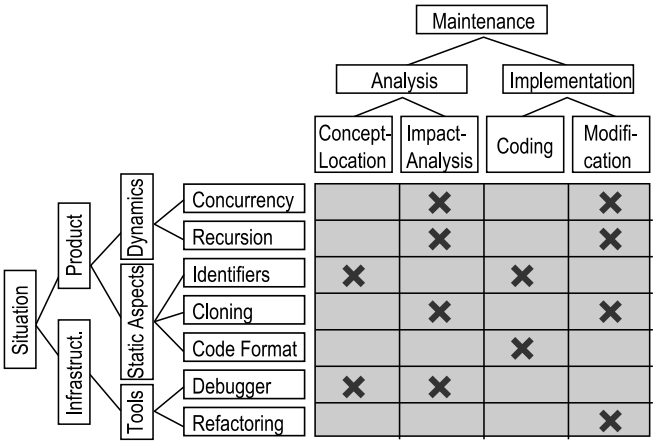

Abbildung 2.3: Aktivitätenbasiertes Maintainability-Modell (Quelle: [49])

Qualitätskostenmodelle

Während in einem ABQM Kosten lediglich ein Hilfskonstrukt sind, um von Aktivitäten auf Qualität zu schließen, existiert eine Reihe von Qualitätsmodellen, in denen der finanzielle Aspekt das zentrale Element darstellt.

Granja-Alvarez und Barranco-García [66] beschreiben eine Methode, um Wartungskosten für Software anhand verschiedener Quelltext-Metriken vorherzusagen. Dabei verwenden sie das *Constructive Cost Model (COCOMO)* [27, 29] als Ausgangspunkt. Die Autoren unterteilen Wartungsaufwand in Aufwand zum Verstehen, zum Ändern und zum Testen von Programmcode. Dabei wird für jede Wartungsaktivität die Abbildung der Metrik auf den Aufwand durch eine Funktion dargestellt, die anhand historischer Daten des Projektes approximiert wird. Sind diese Funktionen bekannt, kann daraus zukünftiger Wartungsaufwand basierend auf den Metriken vorhergesagt werden.

Einen umfassenden Überblick über Qualitätskostenmodelle liefert z. B. das Literatur-Review von Karg, Grottke und Beckhaus [85].

Squale

Die *OpenSource Software*-Gruppe des *System@tic Paris-Region competitive cluster* erarbeitete einen Projektantrag für den fünften *FUI call for projects* unter der zentralen Prämisse, ein praktisch anwendbares Qualitätsmodell zu entwickeln, das sowohl technische als auch ökonomische Aspekte von Softwarequalität betrachtet. Das Projekt startete im Juni 2008 unter dem Namen *Squale* (**S**oftware **Qual**ity **E**nhancement) und veröffentlichte im Januar 2009 eine erste Version seiner Anwendung zur Qualitätsbewertung unter einer freien Lizenz. Die Software basiert auf bestehenden Werkzeugen zur Datenerhebung (z. B. Metriksammlungen) und aggregiert die so gewonnenen Informationen mit Hilfe des Qualitätsmodells zu höherwertigen qualitätsrelevanten Fak-

toren. Die Anwendung bietet Unterstützung für verschiedene Programmiersprachen, darunter Java, C/C++, .NET, PHP, Cobol und andere.

Das entwickelte Qualitätsmodell basiert dabei im Wesentlichen auf ISO/IEC 9126, die es um das Konzept der *Practice* erweitert [105]. Practices stellen die Verknüpfung zwischen messbaren Eigenschaften der Software und den Qualitäts-Charakteristiken der ISO her. Eine Practice besteht dabei aus einem Namen, einer Definition, einer Menge von Maßen, einem Betrachtungsobjekt sowie zwei Bewertungsformeln. Dabei spezifiziert eine Formel die Transformation von Messwerten auf die Bewertung einer einzelnen Instanz des Betrachtungsobjekts (*Individual Mark*); die zweite beschreibt die Aggregation dieser Bewertungszahlen auf das Gesamtsystem (*Practice Mark*). Dieser Ansatz, innerhalb einer Practice immer das Gesamtsystem zu bewerten, vermeidet die Erfordernis eines formal spezifizierten Modells der System-Artefakte und deren Beziehungen. Die Aggregation verschiedener Practices auf Instanzebene und Aggregation auf komplexere Betrachtungsobjekte (z. B. Komponente, Subsystem) ist damit jedoch nicht möglich.

2.3 Von Architektur zu SOA

Der Begriff *Architektur* wird in der *Encyclopedia Britannica* wie folgt definiert [15]:

> The art and technique of designing and building as distinguished from the skills associated with construction. [...] The characteristics that distinguish a work of architecture from other man-made structures are
>
> - the suitability of the work to use by human beings in general and the adaptability of it to particular human activities,
> - the stability and performance of the works construction, and
> - the communication of experience and ideas through its form.

Architektur allgemein bezeichnet also insbesondere die Struktur von Dingen, die meist in Form von Zeichnungen kommuniziert wird, welche als Grundlage für die Erstellung oder spätere Umbauten verwendet werden können. In den folgenden Abschnitten wird diese Definition auf die Softwareentwicklung übertragen. Abschnitt 2.3.1 gibt einen allgemeinen Überblick über Softwarearchitektur, bevor Abschnitt 2.3.2 bestehende Ansätze zur Architekturmodellierung beschreibt. Verfahren zur Architekturbewertung von Software werden in Abschnitt 2.3.3 vorgestellt. In Abschnitt 2.3.4 werden Konzepte und Definitionen aus dem Umfeld serviceorientierter Architekturen vorgestellt und gegeneinander abgegrenzt. Basierend darauf wird in Abschnitt 2.3.5 eine Definition serviceorientierter Softwarearchitektur als Grundlage für das SOA-Qualitätsmodell erarbeitet.

2.3.1 Softwarearchitekturen

In der Softwareentwicklung bezeichnet der Begriff *Architektur* analog zur vorangegangenen, allgemeinen Definition den strukturellen Aufbau einer Software, ihre Aufteilung

in einzelne Subsysteme oder Komponenten und die Verbindungen zwischen diesen. Architektur ist in der fertigen Software also ebenso inhärent wie in einem Bauwerk. Dies wird beispielsweise in der Definition von Bass, Clements und Kazman [25] deutlich:

> The software architecture of a program or computing system is the structure or structures of the system, which comprise software elements, the externally visible properties of those elements, and the relationships among them.

In Anlehnung daran definieren Vogel u. a. [138] den Begriff der Softwarearchitektur aus struktureller Sicht wie folgt:

> Die Software-Architektur eines Systems beschreibt dessen Software-Struktur respektive dessen -Strukturen, dessen Software-Bausteine sowie deren sichtbare Eigenschaften und Beziehungen zueinander und zu ihrer Umwelt.

Insbesondere argumentieren die Autoren, dass der Begriff Softwarearchitektur zwei komplementäre Konzepte vereint: die Softwarestruktur eines IT-Systems auf der einen, und die Disziplin zum Entwurf und zur Umsetzung dieser Struktur auf der anderen Seite. Im weiteren Verlauf dieser Arbeit bezeichnet der Begriff grundsätzlich den strukturellen Aspekt.

Als *Unternehmensarchitektur* (*Enterprise Architecture*, EA) bezeichnet man die Architektur eines gesamten Unternehmens aus informationstechnischer Sicht. Es wird also nicht der strukturelle Aufbau einer einzelnen Softwareanwendung beschrieben, sondern das Zusammenspiel aller IT-Systeme, die in einem Unternehmen existieren.

Perry und Wolf [119] beschreiben Softwarearchitektur durch das Tripel aus Elementen, Form und Begründung. Elemente können dabei ausführende Elemente, Datenelemente oder verknüpfende Elemente sein. Ausführende Elemente sind diejenigen Elemente der Architektur, die Datenelemente verändern. Datenelemente sind Elemente, welche die Informationen enthalten, die im Programm verwendet und verändert werden. Verknüpfende Elemente sind solche Elemente, welche die Architektur zusammenhalten. Sie können wahlweise ausführende Elemente, Datenelemente oder auch beides sein. Beispiele für verknüpfende Elemente sind Funktionsaufrufe, Nachrichten oder gemeinsam genutzte Speicherbereiche.

Die *Form* besteht aus gewichteten Eigenschaften sowie Beziehungen. Die Gewichtung bestimmt dabei entweder die Wichtigkeit eines Elements oder die Notwendigkeit, sich für eine von mehreren Alternativen zu entscheiden. Ersteres erlaubt dem Architekten zwischen funktional wichtigen und dekorativen Elementen zu unterscheiden, letzteres, einen Spielraum für die Implementierung zu lassen, aber trotzdem die Entscheidungsrichtung vorzugeben. Die Eigenschaften selbst dienen dazu, die Auswahl von Elementen einzuschränken. Alles, was vom Architekten nicht durch Eigenschaften eingeschränkt worden ist, liegt im Verantwortungsbereich des Designers. Beziehungen schließlich dienen dazu, die Interaktionen und die Organisation der Elemente zu beschränken. Auch hier gilt: Was nicht spezifiziert ist, steht dem Designer frei zu entscheiden.

In der *Begründung* werden schließlich die Hintergründe für die getroffenen Entscheidungen explizit dargestellt. Dabei können sowohl grundlegende funktionale Anforderungen eine Rolle spielen als auch nichtfunktionale Aspekte wie Qualitätsanforderungen.

Als *Architekturstil* bezeichnen Perry und Wolf eine Abstraktion von Elementen und formalen Aspekten verschiedener Architekturen. Ein Architekturstil enthält weniger Informationen und Beschränkungen als eine konkrete Architektur, z. B. weil er sich auf bestimmte Aspekte der Interaktion von Elementen konzentriert, ohne die Elemente selbst konkret festzulegen. Dennoch existiert keine scharfe Trennlinie zwischen den Konzepten *Architektur* und *Architekturstil*. Im Rahmen der vorliegenden Arbeit wird SOA als Architekturstil bezeichnet.

Das Zachman-Framework [148] ist ein Ansatz, um die Architektur von Informationssystemen strukturiert zu beschreiben. Zentrales Argument dabei ist, dass es nicht *eine* Architektur gibt, sondern dass für jede beteiligte Rolle verschiedene Aspekte der Architektur von Bedeutung sind, so dass verschiedene Repräsentationen erforderlich werden. Die Architektur von Systemen ist also ebenso wie deren Qualität ein multidimensionales Konzept, für das abhängig von der jeweiligen Zielsetzung, beteiligten Personen und anderen Rahmenbedingungen eine geeignete Projektion gewählt werden muss.

Svahnberg und Wohlin [132] sehen in der Architektur ein zentrales Mittel, um die Qualität einer Software über deren Lebenszyklus hinweg sicher zu stellen. Dazu muss die Architektur allerdings von allen Entwicklern verstanden werden, und alle Änderungen am System müssen sich an die in der Architektur festgelegten Vorgaben halten. Ändern sich Anforderungen an das System, muss die Architektur entsprechend angepasst und aktuell gehalten werden. Da dies in der Praxis oft nicht der Fall ist, existiert im *Software Engineering (SE)* eine ganze Reihe Forschungsarbeiten darüber, wie man aus einer Software die Ist-Architektur in einer Darstellung extrahieren kann, die einen einfachen Abgleich mit der Soll-Architektur erlaubt [53, 84, 86, 125].

Im Folgenden wird Softwarearchitektur vor allem hinsichtlich der Struktur von Software verstanden, weshalb die Definition nach Bass, Clements und Kazman [25] respektive Vogel u. a. [138] zugrunde gelegt wird. Demnach besteht die Architektur der Software aus Software-Bausteinen, deren von außen sichtbare Eigenschaften sowie ihren Beziehungen untereinander und zu ihrer Umwelt. Dabei kann die Granularität der Beschreibung je nach Perspektive variieren, so dass Größe und Art der Bausteine vorrangig davon abhängen, zu welchem Zweck die Architektur betrachtet wird.

2.3.2 Architekturmodellierung

Als prominentester Vertreter von Sprachen zur Modellierung von Softwarearchitektur ist die *Unified Modeling Language (UML)* zu nennen. Diese bildet ein Metamodell zur Modellierung von Software und basiert ihrerseits auf der *Meta Object Facility (MOF)* der *Object Management Group (OMG)* [110]. Gemäß dem Konzept der *vierschichtigen Modellierungsarchitektur* ist dabei die MOF auf der Modellierungsebene M3 angesiedelt. Die UML als eine mögliche Instanz dieses Metametamodells wird der Ebene

M2 zugeschrieben, konkrete UML-Modelle bilden die Ebene M1, und Instanzen dieser Modelle (etwa Objekte vom Typ der dort modellierten Klassen) liegen auf der Ebene M0.

Die *Fundamental Modeling Concepts (FMC)* [89] stellen eine Modellierungssprache für Systemarchitekturen dar. Diese wird von den Autoren als leicht zu erlernendes und anzuwendendes Mittel beschrieben, um die Struktur komplexer Systeme zu beschreiben und zu kommunizieren. Die FMC definiert drei Diagrammtypen: Blockdiagramme, Petrinetze und Entity-Relationship-Diagramme. Blockdiagramme dienen zur Visualisierung der Struktur von Softwaresystemen und verwenden vor allem die Konzepte *Agent*, *Storage* und *Channel*. Das dynamische Verhalten eines Systems wird durch Petrinetze beschrieben, die im Wesentlichen aus *Stellen* und *Transitionen* bestehen. Entity-Relationship-Diagramme beschreiben einerseits Daten und deren Beziehungen, werden andererseits aber auch verwendet, um jegliche Art von Beziehungen zwischen interessanten Aspekten eines Systems („*Topics*“) darzustellen.

Grundsätzlich haben Modelle die Aufgabe, komplexe Sachverhalte vereinfacht darzustellen und vermittelbar zu machen. Mit allgemeinen Sprachen wie der UML ist dies aufgrund ihrer Mächtigkeit nur eingeschränkt möglich, so dass entsprechende Modelle oft nur unwesentlich einfacher erscheinen als der Programmcode, den sie repräsentieren. Um diesem Umstand Rechnung zu tragen, werden Architekturmodelle häufig mittels einer *Domain-Specific Language (DSL)* beschrieben. Eine solche Sprache ist speziell auf die Verwendung in einer bestimmten Domäne zugeschnitten, so dass ihre Sprach-Elemente über eine stärkere Semantik verfügen als die Elemente allgemeiner Sprachen. Diese zusätzliche Semantik ermöglicht es, Sachverhalte aus der spezifischen Domäne einfacher darzustellen, so dass die entsprechenden Modelle weniger komplex sind.

Eine Art domänenspezifischer Sprachen für die Domäne *Softwarearchitektur* stellen die *Architecture Description Languages (ADLs)* dar. Diese enthalten im Wesentlichen die Konzepte *Komponente*, *Konnektoren* zur Beschreibung der Kommunikation zwischen diesen Komponenten, sowie der *Architektur-Konfiguration*, die beschreibt, welche Komponenten über welche Konnektoren verbunden sind [138]. Eine Sprache speziell für die Modellierung serviceorientierter Architekturen ist die *Service oriented architecture Modeling Language (SoaML)* [111]. Sie wurde von der OMG als UML-Profil entwickelt und im März 2012 in Version 1.0 freigegeben. Da SoaML aufgrund ihres geringen Alters bisher noch keine weite Verbreitung gefunden hat und insbesondere während der Arbeiten am SOA-Qualitätsmodell noch nicht veröffentlicht war, findet sie in dieser Dissertation keine Anwendung.

Die SAP AG als Vertreter der Softwarebranche hat mit *Technical Architecture Modeling (TAM)* [124] einen eigenen Standard für die technische Modellierung von Softwarearchitekturen entwickelt. Dabei handelt es sich im Wesentlichen um eine Kombination von Elementen aus FMC und UML, welche einem *Best-Practice*-Ansatz folgt. So werden verschiedene Diagrammtypen aus den beiden Sprachen zu einem Gesamtkonzept kombiniert: Das Blockdiagramm aus FMC sowie Anwendungsfall-, Paket-, Klassen-, Aktivitäts-, Zustands- und Sequenzdiagramme aus UML. Zusätzlich zu dieser Einschränkung der Diagrammtypen wurden auch die zu verwendenden Elemente

innerhalb der Diagramme eingeschränkt, um die Darstellung der Modelle zu vereinheitlichen und die Modellierung zu vereinfachen.

Werden Architekturmodelle nicht nur zur Dokumentation, sondern als zentrale Artefakte des Entwicklungsprozesses verwendet, spricht man von *modellgetriebener Softwareentwicklung* (*Model-Driven Software Development*, MDSD). Hier sind formale, maschinenlesbare Architekturmodelle erforderlich, um automatisiert Programmcode zu generieren. Dabei kommen vor allem domänenspezifische Sprachen (textuell oder grafisch) oder UML zum Einsatz. Zur Anwendbarkeit von UML sowie deren Erweiterungen merkt etwa Wickel [143] an, dass die Vielzahl an Elementen und Notationsergänzungen zu einer gesteigerten Komplexität führen, so dass eine praktische Anwendung nicht in allen Fällen zu empfehlen sei – insbesondere dann nicht, wenn die Modelle ausschließlich zu Dokumentations- und Kommunikationszwecken verwendet werden. Auch im Hinblick auf die steigende Popularität agiler Methoden zur Entwicklung von Software und der damit verbundenen inkrementellen und oft kollaborativen Arbeitsweise seien heutige Modellierungswerkzeuge nur sehr eingeschränkt geeignet.

2.3.3 Architekturbewertung

Die Softwarearchitektur ist entscheidend für die Qualität einer Software. Das bedeutet, dass die Entscheidung für eine bestimmte Architektur maßgeblich von den Qualitätszielen der beteiligten Akteure abhängt. Clements, Kazman und Klein [42] formulieren diesen Zusammenhang besonders prägnant:

> If the sponsor of a system cannot tell you what any of the quality goals are for the system, then any architecture will do.

Wenn Architekturentscheidungen also entscheidend zur Qualität eines Softwaresystems beitragen, dann erscheint die Bewertung der Softwarearchitektur als probates Mittel um Rückschlüsse auf dessen Qualität zu ziehen. Zu diesem Zweck wurden Methoden zur Architekturbewertung entwickelt, welche klaren Regeln folgen und ein gewisses Maß an Wiederholbarkeit gewährleisten. Diese lassen sich in Anlehnung an o. g. Arbeit nach den verwendeten Werkzeugen unterscheiden: *Fragende* Methoden basieren auf Szenarien, Fragebögen oder Checklisten zur Analyse eines Systems, häufig in Verbindung mit einer Dokumentation oder Spezifikation desselben, das im Allgemeinen noch nicht oder zumindest noch nicht vollständig implementiert ist. Im Gegensatz dazu analysieren *messende* Methoden direkt das zu bewertende System mit Hilfe von automatisierten Werkzeugen, z. B. zur Erhebung von Softwaremetriken oder zur Simulation des Systemverhaltens. Dies erfordert also in jedem Fall das Vorhandensein von Softwareartefakten und kann deshalb im Allgemeinen nicht so früh angewendet werden wie etwa szenariobasierte Methoden.

Die Arbeit von Clements, Kazman und Klein [42] stellt die drei Methoden SAAM, ARID und ATAM vor, die alle zur Gruppe der szenariobasierten Bewertungsmethoden gehören, also die Bewertung einer Softwarearchitektur an Szenarien knüpfen. Diese werden in einem ersten Schritt in Workshops erarbeitet; ein entsprechendes Szenario

könnte lauten: „Das System soll zukünftig alternativ zur lokalen Installation auch auf einer Cloud-Plattform betrieben werden können“. Basierend auf einer priorisierten Liste solcher Szenarien werden verschiedene Architektur-Alternativen daraufhin untersucht, wie gut sie die Durchführung der Szenarien unterstützen. Dabei kommen je nach Methode und Anwendungsfall unterschiedliche Verfahren zum Einsatz, z. B. Sensitivitäts- und Tradeoff-Analyse bei ATAM, Szenario-Durchgänge bei SAAM oder *Active Design Reviews* [117] bei ARID. Eine Gegenüberstellung dieser drei Verfahren findet sich in [42, S. 256]. Für eine mittelgroße Architekturbewertung nach ATAM geben die Autoren den zu erwartenden Aufwand mit etwa 70 Personentagen an, wobei sie von fünf Mitgliedern des Evaluationsteams, drei Entscheidungsträgern und acht weiteren Interessenshaltern ausgehen.

Ein allgemeiner Überblick über Methoden zur Beurteilung von Architekturen ist außerdem bei Vogel u. a. [138] zu finden. Darüber hinaus liefert Zhao [149] Verweise auf weiterführende Literatur. Generell ist zu beobachten, dass die überwiegende Mehrheit der Methoden zur Architekturanalyse nach o. g. Klassifikation dem Bereich der fragenden Methoden zuzuordnen und folglich mit großem manuellen Aufwand verbunden ist. Dies mag der Tatsache geschuldet sein, dass die Analyse einer Architektur grundsätzlich im Kontext projektspezifischer Anforderungen durchgeführt wird. Für domänen- und projektunabhängige Bereiche der Analyse von Architekturen existieren allerdings werkzeuggestützte Verfahren. So können etwa mit ConQAT[6] tatsächliche Abhängigkeiten zwischen Komponenten innerhalb einer Software automatisch mit den Vorgaben der Architekturspezifikation verglichen, Abweichungen identifiziert und grafisch dargestellt werden.

2.3.4 Serviceorientierte Architektur

Der Architekturstil, der im weiteren Verlauf der vorliegenden Arbeit betrachtet wird, ist die SOA. In Literatur und industrieller Praxis existieren unzählige Definitionen von SOA, welche jeweils unterschiedliche Aspekte betonen. Dies hat dazu geführt, dass SOA unter anderem als *Seriously Overloaded Acronym* (etwa: ernsthaft überfrachtetes Akronym) bezeichnet wird. Manche Experten vertreten die Ansicht, SOA sei lediglich eine Menge von Design-Richtlinien, die seit vielen Jahren in jeder guten Architektur berücksichtigt sein sollten. In diesen Kreisen ist die (scherzhafte) Bezeichnung *Same Old Architecture* gebräuchlich.

Diese Beobachtungen verdeutlichen, dass eine klare Abgrenzung des Begriffs SOA von entscheidender Bedeutung ist. Aus diesem Grund wird im Folgenden ausführlich dargestellt, wie SOA im Kontext dieser Dissertation zu verstehen ist und welche Aspekte besonders relevant für die nachfolgenden Kapitel sind.

Das *SOA Glossary*[7], welches die terminologische Grundlage für die gesamte *Prentice-Hall Service-Oriented Computing Series from Thomas Erl* bildet, definiert SOA als ein *technologisches Architekturmodell*, welches auf dem *Design-Paradigma* der Serviceori-

[6] `http://www.conqat.org/`
[7] `http://www.soaglossary.com/`

entierung fußt. Ausgehend von der allgemeinen Betrachtung von Softwarearchitektur im vorigen Abschnitt wird SOA im Folgenden weiterhin als *Architekturstil* bezeichnet.

Nach Definition des W3C ist eine serviceorientierte Architektur *eine Menge von ausführbaren Komponenten, deren Schnittstellenbeschreibungen veröffentlicht und gefunden werden können*:

> A set of components that can be invoked and whose interface descriptions can be published and discovered. [140]

Neben der *Ausführung* von Diensten werden also insbesondere die *Veröffentlichung* und die *Suche* von Schnittstellenbeschreibungen betont. Allerdings bedient sich obige Definition auf unvorteilhafte Art und Weise des Komponentenbegriffs, so dass keine klare Abgrenzung zwischen serviceorientierten und Komponentenarchitekturen gegeben ist. Hierzu sei auf Vogel u. a. verwiesen, welche den Begriff des Dienstes klar von dem einer Komponente abgrenzen:

> Services sind generell grober granular als Komponentenschnittstellen und hinsichtlich ihrer Geschäftsrelevanz stärker strukturiert als Komponenten. [138]

Hier wird vor allem der stärkere Bezug von Diensten zur Geschäftsrelevanz, also der fachlichen Sicht auf die angebotene Funktionalität, hervorgehoben, während Komponenten grundsätzlich nach technischen Gesichtspunkten strukturiert sind. Auch Jones [79] argumentiert ausgehend von der fachlichen Sicht auf Funktionalität und definiert in diesem Zusammenhang einen Dienst als eine diskrete Kontrolleinheit, die eine Sammlung von Aufgaben enthält um verwandte Ziele zu erreichen.

Aus diesen Betrachtungen lässt sich die Schlussfolgerung ableiten, dass eine neue Schnittstelle allein aus einer Komponente noch keinen Service macht. Insbesondere ist dabei von Bedeutung, dass Dienste in einer SOA wie oben erwähnt stärker nach fachlichen Gesichtspunkten strukturiert sind, also Aspekte der Anwendungsdomäne repräsentieren. Dienste, welche in diesem Sinne fachlicher Natur sind, besitzen im Allgemeinen einen starken Bezug zu Geschäftsprozessen. Aus einer technischen Perspektive (etwa der eines Softwareentwicklers) kann diese Fachlichkeit auch als Abstraktion von technischen Konzepten oder als eine Ausprägung von Granularität angesehen werden, weshalb diese Begriffe in der Praxis je nach Kontext synonym gebraucht werden.

Die fachliche Orientierung von Diensten führt auf der Ebene der Softwarearchitektur ebenfalls zu einer fachlich geprägten Struktur, wohingegen etwa Komponentenarchitekturen nach technischen Gesichtspunkten strukturiert sind. Somit ist der Aspekt der Fachlichkeit ein Kennzeichen von SOA gegenüber anderen Architekturstilen. Einer der häufigsten Kritikpunkte bei der Einführung von SOA in Unternehmen ist, dass Service-Schnittstellen zur Anbindung von Altsystemen erstellt werden, die in ihrer Struktur und ihrer Granularität stark an den zumeist technisch begründeten, bestehenden Schnittstellen dieser Systeme orientieren, um diese mit möglichst geringem Aufwand zu kapseln und innerhalb der SOA weiter zu verwenden. Ein solches

Vorgehen wird üblicherweise damit begründet, die Kosten der SOA-Einführung zu minimieren und Investitionen zu sichern, die in die Entwicklung dieser Altsysteme geflossen sind [138]. Dienste, welche auf diese Weise entstehen, sind allerdings in aller Regel eher technischer als fachlicher Natur und verletzen deshalb im Allgemeinen dieses zentrale SOA-Prinzip.

Auch bezüglich der zentralen Elemente einer SOA ist die Terminologie nicht einheitlich. So führt Burbeck [33] in diesem Zusammenhang *Service Provider*, *Service Requester* und *Service Broker* an, während Krafzig, Banke und Slama [90] von *Application Front End*, *Service*, *Service Repository* und *Service Bus* sprechen. Solchen scheinbar grundverschiedenen Begriffen liegt im Allgemeinen eine unterschiedliche Betrachtungsweise zugrunde. Im genannten Fall bezieht sich Burbeck vor allem auf die organisatorischen Rollen, welche in einer SOA zu finden sind, während Krafzig, Banke und Slama technische Artefakte beschreiben, aus denen diese besteht.

Andere Definitionen von SOA beschreiben nicht nur den Aufbau einer Architektur, sondern auch ihre Ziele oder Vorteile, z. B. verbesserte Interoperabilität oder Wiederverwendbarkeit. Ebenfalls enthalten Definitionen von SOA zuweilen ihr vorherrschendes Anwendungsgebiet: Entwurf, Konzeption, Erstellung und Ausführung von Geschäftsprozessen [107, 139].

SOA kann sowohl die Grundlage für eine Anwendungsarchitektur sein als auch für eine Unternehmensarchitektur. In den meisten Fällen wird SOA allerdings mit einer Architektur auf einer unternehmensweiten und sogar -übergreifenden Ebene assoziiert. Der Umstand, dass SOA vor allem zur Implementierung von Geschäftsprozessen verwendet wird und damit gewissermaßen ein Bindeglied zwischen der fachlichen Domäne und ihrer technischen Repräsentation bildet, begünstigt diese Vielfalt an Perspektiven und Terminologien. Für eine Abgrenzung verwendeter Begriffe muss also grundsätzlich die Perspektive berücksichtigt werden. Die in den betrachteten Definitionen verwendeten Begriffe beschreiben zumeist Konzepte aus den Bereichen *Bestandteile der technischen Architektur*, *Rollen und Akteure*, *Ziele* und *Anwendungsgebiete* von SOA. Abbildung 2.4 fasst überblicksartig gebräuchliche Begriffe aus der SOA-Literatur zusammen und ordnet diese in die hier erwähnten Kategorien ein.

Einige SOA-Definitionen legen ihren Schwerpunkt auf spezifische Eigenschaften der Architektur. Das W3C [139] stellt dabei etwa den idealen Abstraktionsgrad der Dienstbeschreibung in den Vordergrund. Erl [56] liefert eine differenziertere Darstellung und unterteilt das *Paradigma* der Serviceorientierung in die folgenden acht *Design-Prinzipien*, denen er jeweils ein Kapitel widmet: Standardized Service Contract, Service Loose Coupling, Service Abstraction, Service Reusability, Service Autonomy, Service Statelessness, Service Discoverability, Service Composability.

Serviceorientierung sei dabei ein Mittel zur Erstellung technischer Einheiten (Services), welche so gestaltet sind, dass sie gemeinsam und wiederholt verwendet werden können, um strategische Ziele zu erreichen [56]. Auch diese Definition betont explizit den unternehmerischen Aspekt und schließt folglich Aspekte ein, welche über die Architektur von Software hinaus gehen.

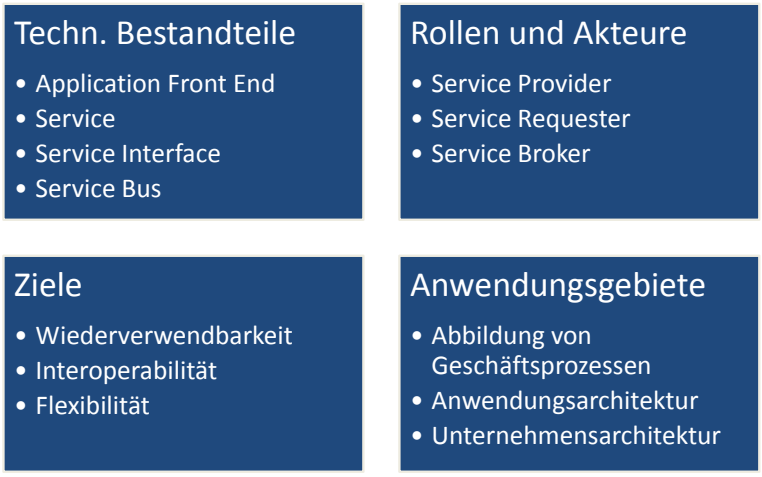

Abbildung 2.4: Einordnung gebräuchlicher SOA-Terminologie

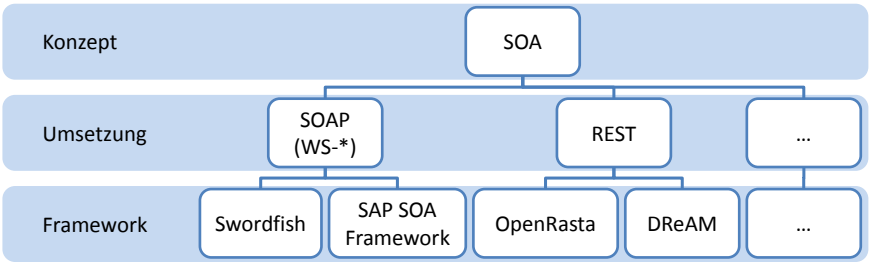

Abbildung 2.5: Beschreibungsebenen von SOA

2.3.5 SOA im Kontext dieser Dissertation

Wie eingangs erwähnt lassen sich Architekturen auf verschiedenen Abstraktionsebenen beschreiben: Auf der Ebene von *Konzepten* lassen sich generelle Eigenschaften und Prinzipien beschreiben; im Falle von SOA sind dies etwa das Vorhandensein standardisierter Schnittstellen. Es wird noch keine Aussage darüber getroffen, welcher konkrete Standard zur Schnittstellenbeschreibung zum Einsatz kommt. Auf der *Umsetzungs*-Ebene werden die Konzepte instanziiert, so ist SOAP/WS-* eine mögliche Umsetzung des Konzeptes SOA. Als weitere Abstraktionsebene existieren *Frameworks*, welche wiederum eine konkrete SOA-Umsetzung instanziieren, z. B. ist das *Swordfish*-Framework[8] eine Implementierung der SOAP/WS-*-Umsetzung von SOA. Abbildung 2.5 veranschaulicht diese Einteilung. Sie ist keinesfalls SOA-spezifisch, sondern dient der Einordnung der im Folgenden eingenommenen Perspektive.

Ziel der vorliegenden Arbeit ist es, *konzeptionell* inhärente Eigenschaften von SOA aus Sicht der Produktqualität eines Softwaresystems zu untersuchen und in einem Qualitätsmodell zu beschreiben. Für die automatisierte Erhebung von Messwerten in

[8] `http://www.eclipse.org/swordfish/`

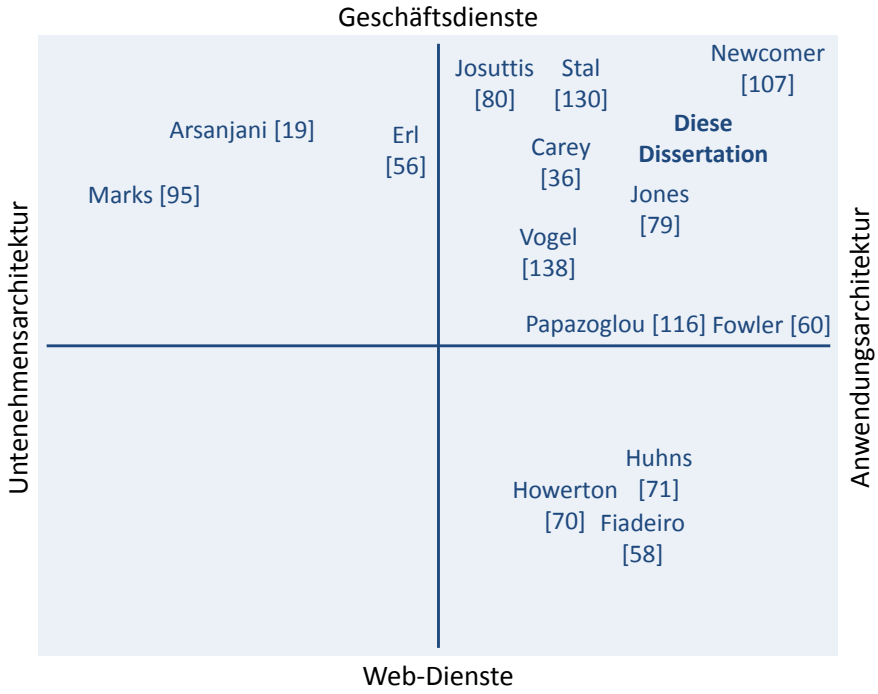

Abbildung 2.6: Einordnung des SOA-Begriffs dieser Dissertation

einem bestimmten System muss natürlich eine konkrete Umsetzung oder ein entsprechendes Framework zugrunde liegen. Ansätze zur werkzeugbasierten Messung werden in Abschnitt 5.3.2 auf Seite 136 beschrieben; eine vollständige Automatisierung der Qualitätsanalyse ist jedoch nicht Gegenstand dieser Dissertation. Da im Folgenden vor allem der Bezug von SOA zur Softwarequalität untersucht wird, fokussiert diese Dissertation auch die Betrachtung von SOA auf Aspekte mit klarem Softwarebezug. In dieser Hinsicht weicht sie etwa von Erl [56] ab, dessen SOA-Definition explizit strategische Ziele einschließt.

Dienste innerhalb einer SOA können verschiedene Granularitäten aufweisen. So gibt es Dienste, die grundlegende technische Aufgaben wie das Auslesen eines Datensatzes aus einer Datenbank erfüllen. Andere Dienste repräsentieren gesamte Geschäftsabläufe wie das Anlegen einer Bestellung inklusive aller dazugehörigen Teilschritte, z. B. der Prüfung der Kreditwürdigkeit oder einem Adressabgleich. Einzelheiten zur Definition dieser Granularitätsebenen sowie weiterer Konzepte der softwarezentrischen Sicht auf SOA werden im Rahmen des SOA-Referenzmodells in Abschnitt 4.3.2 vorgestellt.

Abbildung 2.6 stellt die zwei zentralen Dimensionen von SOA-Definitionen grafisch gegenüber und ordnet die vorliegende Dissertation in diese Darstellung ein. Horizontal wird zwischen der Granularität der Architektur zur Beschreibung eines Unternehmens oder einer Anwendung unterschieden, vertikal zwischen der Art der in der Architektur enthaltenen Dienste. Diese können entweder einen deutlichen Bezug zu Geschäftsprozessen haben und damit im Allgemeinen eher statisch verknüpft sein, oder offen

im Internet angeboten werden (z. B. zur Abfrage von Aktienkursen oder Wetterinformationen) und damit vor allem für dynamische Selektion von Bedeutung sein. Das Hauptaugenmerk dieser Dissertation liegt auf der Softwarearchitektur von betrieblichen Anwendungen, was dem rechten oberen Quadranten in Abbildung 2.6 entspricht. Dieses Verständnis von SOA ähnelt also vor allem dem in den Arbeiten von Newcomer und Lomow [107], Josuttis [80], Stal [130], Carey [36] oder Jones [79].

Zusammengefasst wird im weiteren Verlauf dieser Dissertation SOA als ein *Architekturstil für Software* verstanden, welcher die in Abschnitt 4.5 beschriebenen Prinzipien vereinigt. Im Einzelnen sind dies *Verschachtelte Wiederverwendung und Komposition*, *Verteilung von Funktionalität*, *Geschäftsprozessbezug und Fachlichkeit*, *Standardisierte Formate und Schnittstellen* sowie *Lose Kopplung und dynamische Dienstkomposition*. Diese Prinzipien betrachten gemäß der Fokussierung des SOA-Begriffs auf Software ausschließlich Aspekte, die auf der Ebene der *Softwarearchitektur* in Erscheinung treten und *bewertbar* sind. Die Verwendung dieses SOA-Begriffs ergibt sich aus dem Anspruch der Arbeit, einen Mehrwert für die industrielle Praxis, insbesondere die Softwareentwicklung betrieblicher Informationssysteme, zu bieten. Letztere sind im Allgemeinen stark an Geschäftsprozessen orientiert und bilden eine Anwendung innerhalb eines Unternehmens.

2.4 Qualitätsaspekte serviceorientierter Architekturen

Die Betrachtung von Qualität in serviceorientierten Architekturen erfordert eine strukturierte Auseinandersetzung mit den Prinzipien der Serviceorientierung und daraus abgeleiteten Schlussfolgerungen im Hinblick auf Qualität.

Welche Einflüsse SOA als Ganzes auf bestimmte Qualitätsattribute hat, wurde unter anderem von O'Brien, Bass und Merson [112] untersucht. Für jedes betrachtete Attribut wurde eine Gesamtbewertung in Form einer Ampelfarbe vorgenommen, wobei Rot für einen negativen und Grün für einen positiven Einfluss steht. Mit Gelb bewertete Qualitätsattribute sind mit Einschränkungen versehen, etwa dass sich ein positiver Einfluss nur unter bestimmten Voraussetzungen erreichen lässt. Die Autoren diskutieren die einzelnen Attribute strukturiert und begründen ihre Einschätzung. Allerdings geben sie weder eine Klassifizierung der beschriebenen Attribute an, noch schlagen sie Methoden vor, um die Qualität einer SOA hinsichtlich dieser Attribute zu bewerten. Darüber hinaus existieren zahlreiche Publikationen, die Einflüsse einzelner SOA-spezifischer Systemeigenschaften auf die Qualität eines Systems untersuchen. Diese werden in den folgenden Abschnitten detaillierter betrachtet und bei der Konstruktion des Qualitätsmodells berücksichtigt.

Abschnitt 2.4.1 beschreibt charakteristische Merkmale serviceorientierter Architekturen und identifiziert deren mögliche Auswirkungen auf die Qualität von SOA-Systemen. Abschnitt 2.4.2 fasst die Argumente zusammen, die oft herangezogen werden, um eine bessere Qualität von IT-Systemen durch die Einführung von SOA zu begründen. Abschnitt 2.4.3 stellt verwandte Qualitätsmodelle für SOA überblicksartig vor und bewertet diese im Hinblick auf die Forschungsfragen. Abschnitt 2.4.4 fasst das Kapitel

zusammen und präsentiert eine abschließende Bewertung der betrachteten verwandten Arbeiten.

2.4.1 Qualitätsrelevante Charakteristiken von SOA

Um die besonderen Merkmale von SOA im Vergleich zu anderen Architekturstilen zu identifizieren und ihre Bedeutung für die Qualität eines Systems zu verstehen, wurden die in der Literatur am häufigsten als charakteristisch für SOA bezeichneten Merkmale analysiert. Dabei wurden allein Eigenschaften des Architekturstils betrachtet, ohne von vornherein einen Bezug zur Qualität von SOA-Systemen zu fordern. eine Betrachtung von Arbeiten, die diesen Bezug mit Hilfe von Qualitätsmodellen herstellen, ist in Abschnitt 2.4.3 zu finden. Die folgenden Abschnitte beschreiben jeweils ein charakteristisches Merkmal von SOA und stellen dessen Bedeutung für die Qualitätssicherung eines serviceorientierten Softwaresystems heraus. Tabelle 2.1 zeigt eine Übersicht über ausgewählte Literatur und die dort erwähnten Merkmale von SOA. Abschnitte, welche sich auf technische Eigenschaften der Architektur beziehen, fließen im Rahmen der Operationalisierung der SOA-Prinzipien in das SOA-Qualitätsmodell ein und werden in Abschnitt 4.5 (ab Seite 80) aufgegriffen, insbesondere hinsichtlich der dort beschriebenen Einflüsse auf die Softwarequalität.

Verschachtelte Wiederverwendung und Komposition

Wiederverwendung ist ein zentrales Konzept von SOA. Ausgehend von einzelnen Diensten, werden Anwendungen durch Komposition dieser Dienste erstellt. Fehlende Funktionalität wird dabei neu implementiert und wiederum als Dienst zur Verfügung gestellt, so dass sie auch von anderen Anwendungen verwendet werden kann [36].

Wiederverwendung ist dabei keine neue Idee, sondern findet auch in traditionellen Komponentenarchitekturen ihre Anwendung. Während dies allerdings hauptsächlich aus Gründen der Vereinfachung und Effizienzsteigerung des Entwicklungsprozesses geschah, wird die Wiederverwendung in SOA als zentral angesehen, vor allem im Kontext unternehmensübergreifender Dienste [71, 130]. Ein komponierter Dienst kann entweder als komplette Lösung einem Kunden angeboten werden, oder er kann als Grundlage für weitere Komposition dienen [116]. Die Möglichkeit der verschachtelten Wiederverwendung durch Komposition hat zur Bezeichnung der *fraktalen Nutzung von SOA* geführt [19].

Dieses neue Wiederverwendungskonzept hat einige Auswirkungen auf die Qualitätssicherung: Insbesondere entsteht durch verschachtelte Komposition im Allgemeinen eine große Anzahl von Ebenen. Dies bietet einerseits die Möglichkeit von einzelnen Basisdiensten bis hin zu kompletten Anwendungen die gleichen Testwerkzeuge zu nutzen. Andererseits müssen zur Lokalisierung von Fehlern auch all diese Ebenen untersucht werden.

	[36]	[130]	[71]	[79]	[70]	[58]	[19]	[116]
Verschachtelte Wiederverwendung und Komposition	•	•	•			•	•	•
Auslagerung von Funktionalität	•		•					•
Geschäftsprozessbezug und Fachlichkeit		•	•	•				•
Standardisierte Formate und Schnittstellen	•		•	•	•			•
Automatisierung			•			•		•
Mangelnde Werkzeugunterstützung	•			•			•	
Methodenwandel	•				•		•	•
Dynamische Dientsuche und -komposition			•			•		•

Tabelle 2.1: SOA Charakteristiken – Übersicht

Auslagerung von Funktionalität

Wie im vorigen Abschnitt erwähnt ermöglicht SOA unter anderem eine Wiederverwendung von Diensten über Unternehmensgrenzen hinaus. Dies bedeutet insbesondere, dass Anwendungen verschiedener Unternehmen miteinander interagieren [71] und Dienste verwenden können, die über das Internet zur Verfügung gestellt werden [36].

Verglichen mit herkömmlichen Architekturen steigt also die durchschnittliche Anzahl der an einem Geschäftsprozess beteiligten Unternehmen. Nimmt man an, dass jeder Beteiligte spezifische Anforderungen an die Qualität der angebotenen Dienste stellt, muss ein Qualitätsmodell für SOA in der Lage sein, diese unterschiedlichen Anforderungen und damit verknüpften Bewertungsmethoden zu konsolidieren, um in einem solchen verteilten Umfeld sinnvolle Qualitätsaussagen treffen zu können.

Darüber hinaus bedingt eine verstärkte Nutzung von Netzwerkinfrastrukturen und speziell dem Internet nicht nur ein steigendes Datentransfervolumen, sondern auch die Wahrscheinlichkeit, dass sensible Daten von Dritten verarbeitet werden. Datensicherheit ist also ein wichtiger Qualitätsaspekt serviceorientierter Systeme. Die Mittel diese zu gewährleisten (z. B. Verschlüsselung) unterscheiden sich allerdings nicht von anderen verteilten Systemen.

Geschäftsprozessbezug und Fachlichkeit

Ein weiteres zentrales SOA-Konzept ist der hohe Abstraktionsgrad. Im Gegensatz zur Objektorientierung (*Object-Orientation*, OO) wird SOA zugeschrieben, einen stärker geschäftsgetriebenen Blickwinkel einzunehmen. Während ein Objekt in OO dafür steht, was etwas *ist*, beschreibt ein Dienst in SOA, wie etwas *genutzt* werden sollte [79]. Dienste, welche auf einer hohen Abstraktionsebene definiert sind, verwenden im Allgemeinen dieselbe Terminologie wie die damit abgebildeten Geschäftsprozesse, so dass sich Anwendungsentwickler auf die Spezifika der jeweiligen Domäne konzentrieren können [130]. Betrachtet man darüber hinaus Geschäftsprozesse als Orchestrierung von Diensten, lassen sich Änderungen im Geschäftsprozess durch Änderungen an den entsprechenden Diensten im System abbilden.

Diesem Abstraktionsgrad werden Vorteile bei der Organisation großer Geschäftsanwendungen und der Reduktion von Abhängigkeiten zwischen Teilnehmern zugeschrieben [71].

Standardisierte Formate und Schnittstellen

Für eine SOA ist es essenziell, geltende Standards einzuhalten, um Interoperabilität und Erweiterbarkeit sicherzustellen [71]. Nur so könnten SOAs zum Erfolg werden [79].

In Bezug auf Qualitätssicherung ist Interoperabilität jedoch nicht die einzige Folge von Standardisierung. Weitere Folgen sind Flexibilität [116] und die Verfügbarkeit von Werkzeugen Dritter. Letzteres gilt allerdings auch umgekehrt: Existieren mehrere vergleichbare Standards, werden diejenigen erfolgreich sein, die von den Herstellern entsprechender Werkzeuge umgesetzt werden [36]. Daraus lassen sich folgende Schlussfolgerungen für ein SOA-Qualitätsmodell ableiten: Offensichtlich spielt Standardisie-

rung eine wichtige Rolle bezüglich der Entwicklung universeller und automatischer Test- und Messwerkzeuge. Darüber hinaus sollte Interoperabilität ein weniger großes Problem darstellen, weil eine standardisierte Architektur selbst sich bereits positiv auf Interoperabilität auswirkt – unabhängig von der technischen Umsetzung.

Andererseits muss die Einhaltung von Standards einer ständigen Prüfung (*Governance Process*) unterliegen, was zusätzliche Prozesskomplexität bedingt.

Automatisierung

Oft wird Automatisierung im Zusammenhang mit Standardisierung genannt. Außer den Vorteilen, die im vorigen Abschnitt beschrieben wurden, sieht die Literatur weiteres Potenzial hinsichtlich Selbstorganisation und automatischer Komposition von Diensten [58]. Das Internet und besonders die darauf aufbauenden Anwendungen wurden ursprünglich für die Nutzung durch Menschen entworfen. SOA erlaubt nun die Anpassung dieser Zielsetzung hin zur automatisierten Nutzung Web-basierter Systeme [71].

Aus Qualitätssicht muss eine automatische Komposition von Diensten durch Mechanismen zur automatischen Qualitätsbewertung dieser begleitet werden. Wenn Dienste automatisch komponiert werden, muss die Möglichkeit bestehen, sämtliche relevanten Qualitätsaspekte aller in Frage kommender Basisdienste zu bewerten, um diejenigen auszuwählen, welche die Anforderungen des komponierten Dienstes am besten erfüllen.

Mangelnde Werkzeugunterstützung

Werkzeuge sind ein zentraler Bestandteil des modernen Software-Engineerings. Bestehende Werkzeuge sind allerdings für die Entwicklung einer SOA nur bedingt geeignet. Werkzeuge mit SOA-spezifischen Anpassungen befinden sich in der Entwicklung, insbesondere für den Bereich der Komposition von Diensten. Neue Paradigmen wie z. B. *Data service modeling* werden allerdings noch nicht adäquat in heutigen Werkzeugen abgebildet [36]. Tatsächlich existieren Werkzeuge für beinahe alle Phasen eines SOA-Entwicklungsprozesses. Oft ist allerdings die Kommunikation und Integration dieser Werkzeuge nicht ausreichend sichergestellt, da kein gemeinsames Informationsmodell existiert. Eine erfolgreiche, integrierte SOA-Werkzeugplattform wird dadurch beschrieben, dass sie nicht nur vorhandene Ansätze verbessert, sondern verschiedene Ebenen der Systembeschreibung anbietet, abhängig von den verschiedenen Akteuren in einem SOA-Projekt [79]. Darüber hinaus werden Architekturvalidierung und Kapazitätsplanung als Themengebiete dargestellt, in denen keine angemessenen Werkzeuge zur Verfügung stehen [19].

Werkzeugbasierte Entwicklung ist eine Möglichkeit zur Sicherung der Produktqualität. Deshalb schlägt sich ein Mangel an angemessenen Werkzeugen entweder in geringer Produktqualität oder in zusätzlichem Aufwand für andere Qualitätssicherungsmaßnahmen nieder. Projekte ohne eine integrierte Werkzeug-Infrastruktur benötigen deshalb besonders zuverlässige Qualitätsmodelle.

Methodenwandel

Zusätzlich zu der Forderung nach neuen Werkzeugen wird in der Literatur die These vertreten, dass SOA neue Methoden benötigt. Die Entwicklung von SOA-Lösungen wird als schwierig bewertet, weshalb Architekten ihre Art zu denken auf die Möglichkeiten, die SOA eröffnet, ausrichten müssen [19].

Darüber hinaus hat SOA verschiedene neue Disziplinen hervorgebracht, für die entsprechende Methoden sich noch entwickeln und reifen müssen, z. B. die Modellierung von Datendiensten [36]. Zusammengefasst müssen die Methoden des traditionellen Software-Engineerings mit Wissen über Geschäftsprozesse angereichert werden, um in der Folge „serviceorientierte Software-Ingenieure“ hervorzubringen [26].

Sich entwickelnde Methoden und Prozesse besitzen die gleichen Schwächen wie sich entwickelnde Werkzeuge: Solange sie noch nicht die nötige Reife besitzen, werden die resultierenden Produkte von geringerer Qualität sein. Der Methodenwandel durch SOA wird also zumindest temporär die Qualitätssicherung erschweren.

Dynamische Dienstsuche und -komposition

Dynamisches Auffinden und Aufrufen von Diensten sind zwei der Kernkonzepte von SOA. Diese ermöglichen es, zur Laufzeit Dienste zu komponieren, um nach Bedarf neue Dienste bereitzustellen. Die Möglichkeit, Dienste zur Laufzeit aufzufinden und aufzurufen wird oft als *Dynamic Binding* oder *Ultra-late Binding* bezeichnet. Sie führt zu zusätzlicher Flexibilität und kann z. B. dazu genutzt werden, die Dienstqualität zu optimieren oder Fehlertoleranz auf Anwendungsebene zu erreichen, indem Transaktionszustände dynamisch verwaltet werden [71].

Höhere Dienstqualität und Fehlertoleranz klingen zunächst vielversprechend. Allerdings gehen mit diesen Eigenschaften Risiken durch zusätzliche Komplexität einher. So ist z. B. die Reproduzierbarkeit von Testergebnissen in einem Szenario mit Laufzeit-Binding schwer sicherzustellen. Nicht zuletzt deshalb sind diese Laufzeit-Eigenschaften und darauf abgestimmte Werkzeuge ein intensiv adressiertes Thema im Forschungsfeld.

Zusammenfassung

Da SOA mittlerweile seit einigen Jahren erforscht und in der Praxis eingesetzt wird, herrscht weitgehende Einigkeit über die Merkmale, die SOA von anderen Architekturkonzepten unterscheiden. Die Einflüsse dieser Merkmale auf Qualitätsaspekte von SOA-Systemen sind allerdings oft nicht in gleichem Maße eindeutig beschrieben.

Einige dieser Einflüsse werden intensiv erforscht, z. B. die der Dienstbindung zur Laufzeit [147], welches u. a. Fehlertoleranz auf Anwendungsebene ermöglicht, aber auch zusätzliche Systemkomplexität mit sich bringt. Im Bereich der Standards, Werkzeuge und Methoden existieren bereits praktisch anwendbare Ansätze. Viele davon befinden sich allerdings noch in der Entwicklung und weisen derzeit noch keine ausreichende Reife für den industriellen Einsatz auf [36]. Insbesondere wird ein gemeinsames Informationsmodell für alle beteiligten Werkzeuge als unentbehrlich für den Erfolg von

SOA bezeichnet. Abstraktion, Automatisierung und Wiederverwendung über mehrere Ebenen eröffnen einerseits dem Softwaretest neue Möglichkeiten. Andererseits bergen diese Konzepte Risiken bezüglich Performanz, Fehlersuche oder dynamischer *Quality of Service (QoS)*-Bewertung. Diese Risiken müssen bei der Erstellung von Testwerkzeugen, die von den neuen Möglichkeiten profitieren sollen, in Betracht gezogen werden. In einer Literaturanalyse über Publikationen von SOA-Testumgebungen [136] kommen die Autoren zu dem Schluss, dass traditionelle Verfahren des Softwaretests die Spezifika serviceorientierter Architekturen nur ungenügend adressieren. Insbesondere dynamische Komposition von Diensten erfordere einen adaptiven Ansatz, den sie in Form einer Prozessbeschreibung vorschlagen.

2.4.2 Ziele und Argumente zur Einführung von SOA

Neue Technologien werden üblicherweise einer Kosten-Nutzen-Analyse unterzogen, bevor über ihre Einführung in einem Unternehmen entschieden wird. Für SOA werden im Zuge dessen üblicherweise drei Argumente ins Feld geführt: Steigerung der Flexibilität, um schneller auf Veränderungen in Geschäftsprozessen reagieren zu können, Steigerung der Interoperabilität, um die Integration unterschiedlicher Systeme zu vereinfachen, sowie eine einheitliche Beschreibung der Systemlandschaft durch das gemeinsam zugrundeliegende Architekturprinzip.

Interoperabilität

Ein immer wieder aufgegriffenes Argument für die Einführung von SOA in Unternehmen ist eine gesteigerte Interoperabilität. Diese soll durch den hohen Abstraktionsgrad und vor allem gemeinsame Standards für implementierungsunabhängige Schnittstellenbeschreibungen erreicht werden. Inwieweit dieses Ziel jedoch tatsächlich erreicht wird, ist nicht abschließend geklärt. Laut Lawler und Howell-Barber [91] ergab eine Umfrage von *Information Week Research*, dass 55 % der Befragten deshalb in ihren Unternehmen SOA eingeführt hätten, um eine bessere Integration mit Geschäftspartnern zu erreichen. Dieselbe Umfrage ergab allerdings auch, dass bei 35 % der Befragten dieses Ziel nicht erreicht worden sei.

Überdies wird an vielen Stellen die semantische Interoperabilität als das zentrale Problem von SOA dargestellt, so dass der positive Effekt von SOA auf Interoperabilität durchaus infrage gestellt werden kann.

Flexibilität

Heutige Geschäftsprozesse sind keine starren Gebilde, sondern unterliegen kontinuierlicher Anpassung an aktuelle Gegebenheiten. In schnelllebigen Branchen hängt der Unternehmenserfolg davon ab, wie schnell die eigenen Prozesse sich an veränderte Rahmenbedingungen anpassen lassen. Anpassungen können viele Ursachen haben, z. B. Veränderungen der gesetzlichen Anforderungen, die Konsolidierung von Prozessen nach Firmenübernahmen oder Zusammenschlüssen, oder die Reaktion auf allgemeine Veränderungen im Markt. Neben der Anpassung bestehender Funktionalität ist auch

das Hinzufügen neuer Funktionalität von entscheidender Bedeutung. Nur wenn sich IT-Systeme effizient anpassen und erweitern lassen, können Unternehmen flexibel auf solche Anforderungsänderungen reagieren.

SOA verspricht hier durch die Abbildbarkeit von Geschäftsprozessen auf Dienste eine analoge Abbildbarkeit von Änderungen in Geschäftsprozessen auf kleine Anpassungen, den Austausch oder die Neuentwicklung einzelner Dienste, ohne das Gesamtsystem zu beeinträchtigen.

Uniformität der Beschreibung

Nicht zuletzt spielt SOA bei der Beschreibung von IT-Infrastrukturen eine Rolle (siehe Abschnitt 2.3.1). Wenn allen Bausteinen einer Unternehmensarchitektur SOA-Prinzipien zugrunde liegen, können sie auf die gleiche Art und Weise beschrieben werden. Dies ermöglicht leichtere Prüfung auf Konsistenz der Unternehmensarchitektur sowie das Auffinden von Konsolidierungspotenzial.

2.4.3 SOA-Qualitätsmodelle aus der Literatur

In der Literatur finden sich einige Publikationen, welche als Qualitätsmodelle aus dem Bereich SOA verstanden werden können. Die folgenden Abschnitte stellen einige dieser Publikationen vor, beschreiben die Kernkonzepte und unterziehen die Modelle einer kritischen Würdigung, insbesondere in Bezug auf die Zielsetzung der vorliegenden Arbeit. Dadurch soll einerseits das breite Spektrum existierender Arbeiten aufgezeigt werden, andererseits wird deutlich, wo die vorliegende Dissertation Beiträge leistet, welche über jene der vorhandenen Arbeiten hinausgehen. Kapitel 3 komplementiert diese Betrachtung mit einer Literaturanalyse qualitätsrelevanter Faktoren und Maße aus anderen Bereichen des Software Engineerings.

S-Cube Referenz-Qualitätsmodell für SBA

Eine Publikation, die ein spezielles Qualitätsmodell für den Bereich SOA vorschlägt, ist das *Quality Reference Model for Service Based Applications* [133]. Dieses Qualitätsmodell basiert auf einer umfassenden Literaturrecherche und enthält sämtliche Qualitätsattribute, die für dienstbasierte Anwendungen als wichtig identifiziert wurden. Im Rahmen der Literaturarbeit wurden Qualitätsmodelle aus den Bereichen SE, *Service-oriented Computing (SOC)*, *Business Process Management (BPM)* und *Grid Computing* untersucht und klassifiziert. Für den Bereich SE wurden insbesondere ISO/IEC 9126 und verschiedene UML-Profile betrachtet. Für die Schnittmenge zwischen SE und SOC fanden Ansätze Beachtung, welche durch statische Analyse von Artefakten auf die Bewertung von QoS-Attribute schließen, sowie verschiedene *Design-by-Contract*-Modelle. Aus dem Bereich des SOC werden funktionale QoS-Kompositionsmodelle betrachtet, für den BPM-Bereich Qualitätsmodelle für Service-Netzwerke. Außerdem flossen Qualitätsmodelle aus dem *Grid Computing* in die Analyse ein.

Der Begriff des Qualitätsmodells wird sehr eng definiert als ein Modell, das eine Menge von Qualitätsattributen strukturiert. Hier wird also lediglich der beschreibende Aspekt von Qualitätsmodellen betrachtet. Das vorgeschlagene Qualitätsmodell ist eine hierarchische Struktur von knapp 90 Qualitätsattributen auf vier Ebenen, inklusive Definitionen und Beschreibungen. Die Elemente der ersten Ebene sind: *Dependability*, *Security*, *Data-Related*, *Network & Infrastructure Related*, *Quality of Use Context*, *Usability*, *Configuration & Management*, *Cost*, *Performance* und *Other*.

Das Referenzmodell wird von den Autoren als ein erstes gemeinsames Verständnis von Qualitätsattributen bezeichnet, die für dienstbasierte Anwendungen von Bedeutung sind. Abhängigkeiten zwischen diesen Attributen sollen im weiteren Verlauf des S-Cube-Projekts modelliert werden. Ein entsprechendes Metamodell für die Darstellung dieser Abhängigkeiten ist bereits veröffentlicht. Allerdings ist bisher weder eine Operationalisierung verfügbar, mit deren Hilfe man die Qualität eines SOA-basierten Systems bewerten könnte, noch die entsprechende Werkzeugunterstützung.

Das Modell ist mit 89 Attributen sehr umfangreich und an einigen Stellen nicht überschneidungsfrei definiert, wodurch es zu komplex ist, um intuitiv verwendet werden zu können. So finden sich etwa die Attribute *Availability* und *Continuous Availability*, deren Abgrenzung nicht intuitiv klar ist, weil üblicherweise die Verfügbarkeit eines Systems ohnehin prozentual angegeben wird. Die *ständige* Verfügbarkeit eines Systems wäre demnach gleichzusetzen mit einer Verfügbarkeit von 100 %. An einigen Stellen wird außerdem erwähnt, die Klassifikation der Attribute sei an die der ISO/IEC 9126 angelehnt, was jedoch nicht immer zutrifft. So ist beispielsweise *Safety* als Teil von *Security* modelliert, wohingegen ISO/IEC 9126 dieses Attribut ausdrücklich als Subcharakteristik von *Quality in Use* einordnet, mit der Begründung, dass für *Safety* (also die Vermeidung von Gefahr für den Menschen, Sachwerte und die Umwelt) nicht die Software allein maßgeblich sei, sondern insbesondere der Kontext, in dem sie verwendet werde. Diese Einschätzung hat auch im Nachfolgestandard ISO/IEC 25010 Bestand. Das S-Cube-Modell bleibt jedoch weitgehend Begründungen für Abweichungen dieser Art schuldig.

OASIS Quality Model for Web Services

Einen ähnlichen Ansatz wie das Referenz-Qualitätsmodell des S-Cube-Konsortiums verfolgt das *OASIS Quality Model for Web Services, Working Draft* [109], das ebenfalls eine strukturierte Menge von Qualitätsattributen mit besonderer Bedeutung im Bereich *Web Services* präsentiert. Diese Qualitätsattribute werden dabei *Quality Factors* genannt und stehen in Wechselwirkung mit *Quality Activities* und *Quality Associates*. Bei letzteren handelt es sich um spezifische Rollen oder Aufgaben innerhalb einer Organisation, welche über Aktivitäten mit Artefakten oder untereinander interagieren. Dabei werden sämtliche Rollen berücksichtigt, die mit dem Service in Berührung kommen, insbesondere auch Entwickler, Provider oder Benutzer. Abbildung 2.7 veranschaulicht diese Zusammenhänge grafisch. *Quality Associates* sind als grüne Rechtecke dargestellt, *Quality Activities* als Pfeile.

Die betrachteten Qualitätsattribute sind *Business Value*, *Service Level Measure-*

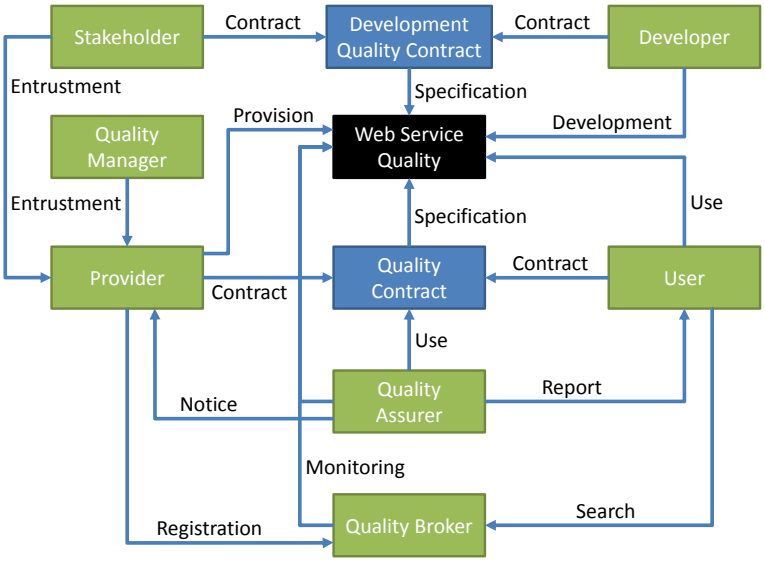

Abbildung 2.7: Konzepte des WSQM (Quelle: [109])

ment, *Interoperability*, *Business Processing*, *Manageability* sowie *Security*. Diese besitzen meist eine Menge an Unterfaktoren und sind jeweils einer der Beschreibungsebenen *Business*, *Service* oder *System* zugeordnet. Dabei enthält die Business-Ebene solche Konzepte, die den Mehrwert beschreiben, der dem Nutzer durch den Dienst entsteht. Auf der Service-Ebene werden messbare Qualitätsaspekte beschrieben, die während der Benutzung des Dienstes erhoben werden, sowie Verfügbarkeits- und Performance-Aspekte, was der üblichen *Quality-in-Use*-Terminologie entspricht. Die System-Ebene enthält Konzepte, welche die Interoperabilität von Services beschreiben, sowohl bezüglich verwendeter Datenformate, als auch bezüglich auszuführender Funktionalität. Darüber hinaus werden hier Aspekte der Handhabbarkeit und Sicherheit betrachtet.

Insgesamt ist das Dokument in einem unreifen Stadium. Viele der verwendeten Definitionen sind nicht klar gegeneinander abgegrenzt oder lassen Präzision vermissen [73]. Darüber hinaus enthält das Modell weder konkrete Maße noch Messvorschriften, mit deren Hilfe sich die vorgestellten Qualitätsattribute bestimmen ließen. Es kann somit allenfalls als Einordnungshilfe von Konzepten anderer Arbeiten dienen, stellt aber keine ernst zu nehmende Grundlage für die vorliegende Arbeit dar. Das Dokument befindet sich im Status *Committee Draft* und wurde seit 2005 nicht mehr aktualisiert. Eine Anfrage bei der Vorsitzenden des Komitees, Eunju Kim, am 2. März 2011 ergab, dass das Komitee die Arbeit am Qualitätsmodell im Laufe des Jahres 2011 wieder aufnehmen wird, nachdem zwischenzeitlich die Prioritäten zugunsten der Spezifikation *Web Service Quality Factors (WSQF)* verschoben worden seien, da diese von stärkerer praktischer Relevanz sei. Es handelt sich dabei im Wesentlichen um eine Sammlung von Definitionen der *Quality Factors*, die oben bereits beschrieben wurden.

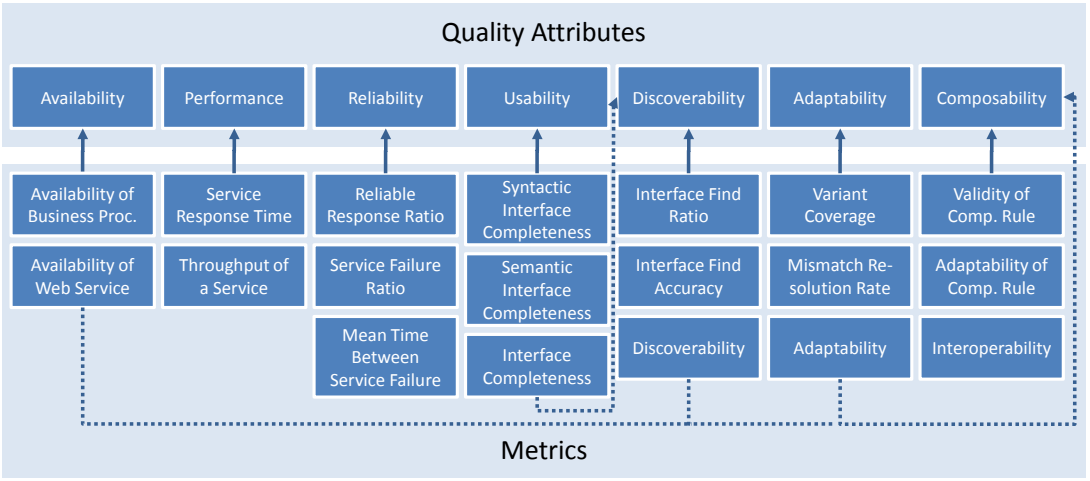

Abbildung 2.8: Qualitätsmodell nach Choi, Her und Kim (Quelle: [41])

SOA-Qualitätsmodell aus Nutzersicht

Choi, Her und Kim [41] schlagen ein Qualitätsmodell für Services vor, welches die Qualitätsanforderungen des Nutzers in den Vordergrund stellt. Das typische Profil eines Nutzers beschreiben die Autoren wie folgt: Ein Nutzer sucht in einer *Registry*, um einen passenden Dienst ausfindig zu machen. Anschließend wählt er einen Dienst eines bestimmten Anbieters aus, bindet diesen in seine Anwendung ein und ruft ihn schließlich auf. Darüber hinaus kann ein Nutzer mehrere Dienste komponieren, um die gewünschte Funktionalität zu bekommen, oder einen Dienst entsprechend der eigenen Anforderungen anpassen. Die wichtigsten Anliegen eines Nutzers sind demnach die effiziente Auffindbarkeit von Diensten, deren stabile und zuverlässige Ausführung, die Interoperabilität von Diensten im Sinne einer einfachen Komponierbarkeit sowie die Anpassbarkeit von Diensten an leicht veränderte Situationen.

Das vorgeschlagene Qualitätsmodell enthält die sieben Attribute *Availability*, *Performance*, *Reliability*, *Discoverability*, *Adaptability*, *Composability* und *Usability*. Diese werden neben der Begründung durch die Kernanliegen von Service-Nutzern außerdem argumentativ von den Charakteristiken von SOA hergeleitet. Pro Attribut schlagen die Autoren mehrere Metriken mit konkreten Messvorschriften vor und definieren Berechnungsformeln für die Aggregation der Messwerte zu je einer Gesamtbewertung pro Attribut. Da die Metriken speziell auf den Nutzer ausgerichtet sind, lassen sie sich erst nach der Entwicklung eines Dienstes anwenden, also sobald dieser verfügbar ist. Eine Übersicht über die vorgeschlagenen Attribute und Metriken ist in Abbildung 2.8 dargestellt.

Eine Sonderrolle nimmt *Composability* ein, da sich dieses Attribut nach Ansicht der Autoren aus dem Zusammenwirken anderer Attribute ergibt. Für einen zusammengesetzten Dienst ergibt sich die Bewertung aus den Bewertungen für *Availability*, *Discoverability* und *Adaptibility* der verwendeten Dienste, der Validität und Anpass-

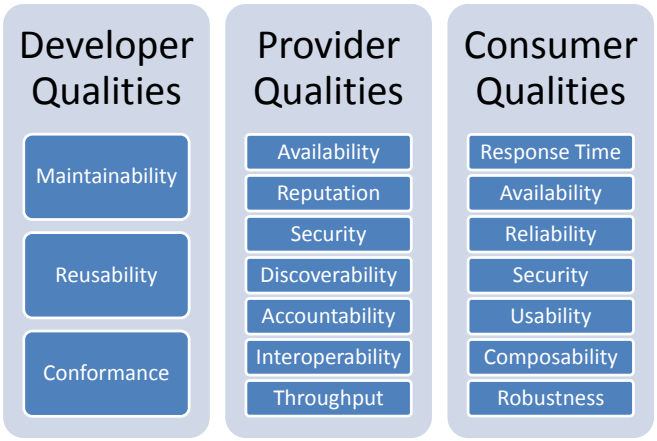

Abbildung 2.9: Qualitätsattribute nach Balfagih und Hassan (Quelle: [21])

barkeit der Kompositionslogik sowie der Interoperabilität zwischen den verwendeten Diensten.

Für die Attribute *Availibility* und *Performance* haben Choi, Her und Kim eine Fallstudie an einem Flugreservierungssystem durchgeführt und die Metriken für sinnvoll befunden. Für andere Metriken argumentieren sie, dass derzeitige Ansätze zum Auffinden, Anpassen und Komponieren von Diensten noch nicht reif genug seien, um eine praktische Evaluation durchführen zu können, weshalb diese ausschließlich mittels theoretischer Konstrukte validiert wurden. Die Autoren verweisen jedoch darauf, dass diese Metriken die Kernkonzepte eines Dienstes beschreiben und deshalb sinnvoll seien.

Multi-Stakeholder-Qualitätsmodell für Web Services

Das Qualitätsmodell von Balfagih und Hassan [21] basiert auf der Beobachtung, dass dienstbasierte Anwendungen stärker von verschiedenen Interessensgruppen abhängen als traditionelle Softwareprodukte. Eine Analyse verschiedener anderer Qualitätsmodelle ergab, dass dieser Aspekt oft nur unzureichend betrachtet werde, weshalb die Autoren die Zuordnung von Qualitätsattributen zu den beteiligten Rollen als oberste Prämisse ihrer Modellierung definieren.

Insbesondere werden *Provider*, *Consumer*, *Developer* und *Broker* genannt, wobei letzterer im Modell selbst keine Beachtung findet. Für die übrigen drei Rollen werden spezifische Charakteristiken vorgeschlagen, welche aus der jeweiligen Perspektive für die Qualität der Software von Bedeutung sind. Abbildung 2.9 zeigt diese Charakteristiken und ihre Aufteilung zu den beschriebenen Blickwinkeln. Zu beachten ist, dass *Availability* und *Security* sowohl dem Anbieter als auch dem Konsumenten zugeordnet sind.

Für manche dieser Charakteristiken werden außerdem einfache Metriken vorgeschla-

gen, ohne jedoch Anspruch auf Vollständigkeit zu erheben. Beispiele für solche Metriken sind etwa:

$$\text{Availability} = \text{Uptime} \,/\, (\text{Uptime} + \text{Downtime})$$

$$\text{Throughput} = \text{Number of served requests} \,/\, \text{Unit of time}$$

Die Autoren sehen den größten Mehrwert des Modells in der expliziten Trennung der Blickwinkel und dem daraus resultierenden besseren Verständnis der verschiedenen Qualitätsanforderungen aus diesen Blickwinkeln. Die Berücksichtigung dieses Aspektes als Kernelement des Modells ist tatsächlich ein Unterscheidungsmerkmal zu vielen anderen Arbeiten. Die vorgeschlagenen Metriken sind jedoch äußerst abstrakt und deshalb nur mit zusätzlichem Konkretisierungs- und Implementierungsaufwand praktisch anzuwenden.

QM für serviceorientierte Infrastrukturen nach Rud

Dmytro Rud propagiert in seiner Dissertation ein emphQualitätsmodell für serviceorientierte Infrastrukturen [123]. Das Modell ist metrikbasiert und unterscheidet zwischen Produkt-, Prozess-, und Ressourcenmetriken. Jede Metrik wird durch einen eindeutigen Namen charakterisiert und enthält daneben Angaben zur verwendeten Skala, Wertebereich, Maßeinheit und Optimalwert, sowie eine Beschreibung und eine Messvorschrift.

Die Produktmetriken sind in Komplexitäts-, Kritikalitäts- und Granularitätsmetriken unterteilt. Erstere machen Angaben zur Kohäsion, zur Anzahl von Diensten in zusammengesetzten Diensten und zur Abhängigkeit zwischen Diensten im System. Kritikalitätsmetriken bestimmen u. a. die Anzahl semantisch äquivalenter Dienste im System, die Wichtigkeit und Abhängigkeit einzelner Dienste oder die Zuverlässigkeit zusammengesetzter Dienste. Granularität wird einerseits auf funktionaler Ebene für Dienste und Operationen bestimmt, andererseits auch aus Datensicht.

Als Ressourcen werden sowohl Kapazitäten des Netzwerks, der Middleware und beteiligter Rechner verstanden als auch Metadaten. Metriken aus diesem Bereich sind u. a. Anzahl und Existenzdauer verschiedener Versionen desselben Dienstes, die Häufigkeit von Änderungen in den Metadaten eines Dienstes, die Einhaltung von sowie das Risiko eines Verstoßes gegen geltende *Service Level Agreements (SLAs)*, Fehlerraten von Diensten und die Anzahl von Geschäftsprozessen im System.

Um den SOA-Prozess zu bewerten, werden vorhandene SOA-Reifegradmodelle herangezogen. Dabei wird einerseits die Prozessqualität der SOA-Einführung durch ein bestehendes Reifegradmodell bewertet, andererseits auch dessen Qualität. Für letzteres wurden 19 SOA-Reifegradmodelle hinsichtlich festgelegter Kriterien untersucht und Qualitätskennzahlen berechnet. Unternehmen, die eines der untersuchten Reifegradmodelle einsetzen, können also die entsprechende Kennzahl direkt übernehmen, während anderenfalls die Bewertung manuell erfolgen muss.

Insgesamt werden 26 Metriken vorgeschlagen und abschließend hinsichtlich paarweiser Zusammenhänge untersucht. Die Metriken sind technologieunabhängig definiert.

Viele der betrachteten Grundlagen und verwandten Arbeiten basieren allerdings auf SOAP und BPEL.

Welche der vorgeschlagenen Metriken sich für die Bewertung welcher Qualitätseigenschaften eignen, bleibt weitgehend unbeantwortet. Vereinzelt finden sich Hinweise auf solche Einflüsse in den textuellen Beschreibungen der Metriken, ohne dass im Verlauf der Arbeit eine konsolidierte Sicht präsentiert wird. Es handelt sich also um eine Sammlung von Metriken, die den Anspruch erhebt, für SOA relevante Konzepte angemessen abzubilden und für die Qualitätsbewertung zugänglich zu machen. Die Interpretation und Aussagekraft der Messwerte wird allerdings dem Anwender des Modells überlassen. Die Arbeit enthält zwar als weiteres Element ein Performance-Modell für orchestrierte Service-Angebote, dieses wird jedoch unabhängig von den vorgeschlagenen Metriken definiert. Die angeführten Fallstudien wiederum beziehen sich lediglich auf dieses Modell, dessen zentrale Zielgrößen die voraussichtliche Ausführungszeit sowie die voraussichtliche Anzahl von Aufrufen jeder im System vorhandenen Service-Operation sind. Das Modell liefert Vorhersagen zu diesen Größen und bietet so eine Hilfestellung bei der Integration von Diensten im Rahmen der technischen Umsetzung von Geschäftsprozessen.

Service-Qualitätsmodell von Hündling

In seiner Dissertation beschreibt Jens Hündling [73] die Modellierung von Qualitätsmerkmalen für Services. Dabei wird zunächst ein Service-Modell entwickelt, welches anschließend um Konzepte der Service-Qualität erweitert wird. Der Schwerpunkt der Arbeit liegt auf der formalen Beschreibung der Konzepte und der Integration derselben in bestehende Service-Modelle.

Als weiteren Bestandteil enthält die Arbeit Überlegungen zur Aggregation von Qualitätsbewertungen einzelner Dienste zu Aussagen über aus diesen Diensten zusammengesetzte Geschäftsprozesse. Dabei werden rekursive Aggregationsmuster definiert, die entlang der Hierarchie von Kontrollflussblöcken innerhalb eines Geschäftsprozesses eine zusammenfassende Bewertung erlauben. Eine Grundannahme dabei ist, dass ein Prozessschritt auf unterster Ebene durch einen Service implementiert wird. Die vorgeschlagenen Aggregationsmuster sind Summe, Produkt, Minimum, Maximum, geometrisches sowie arithmetisches Mittel, wahrscheinlichkeitsbehaftete Aggregation, die Aggregation der ersten N Prozessschritte sowie die Übernahme der Bewertung eines einzelnen (besonders wichtigen) Prozessschrittes für einen gesamten Kontrollflussblock. Außerdem ist die Aggregation verschiedener Qualitätsaspekte eines solchen Blocks zu einem neuen Qualitätsaspekt vorgesehen.

Konkrete Qualitätsaspekte werden lediglich in Beispielen genannt und dienen der Illustration der Konzepte. Im Rahmen einer Fallstudie wurden insgesamt sieben Qualitätsmerkmale aus den Kategorien Performance, Stabilität und Kosten verwendet. Das entwickelte Modell ist jedoch allgemeingültig und erlaubt die Bewertung von Diensten hinsichtlich weiterer Qualitätsaspekte, um etwa zwischen verschiedenen Alternativen mit gleicher oder ähnlicher Funktionalität auszuwählen. Die Bewertung dienstbasierter Systeme hinsichtlich standardisierter Qualitätsaspekte sieht der Ansatz allerdings

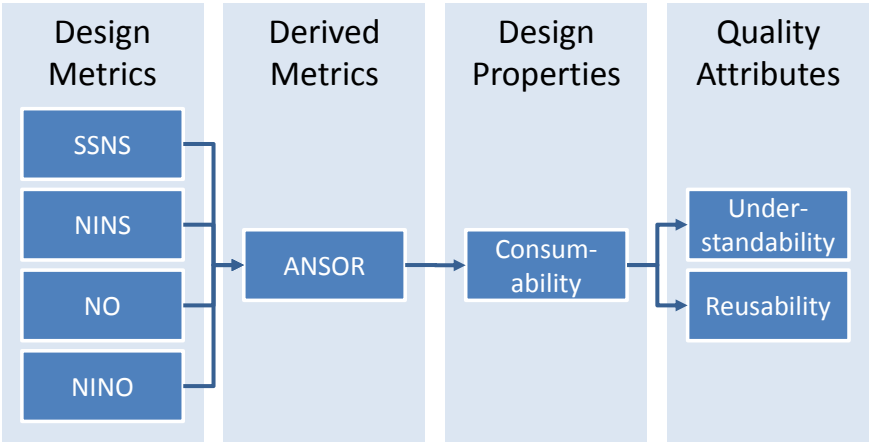

Abbildung 2.10: Qualitätsmodell nach Shim u. a. – Ausschnitt (Quelle: [128])

nicht vor. Die Einbeziehung von Qualitätsaspekten, „die einem allgemein akzeptierten und standardisierten Grundwortschatz entstammen“, ist als mögliche Erweiterung des Modells erwähnt. Das Objekt der Bewertung ist dabei in allen Fällen ein Dienst oder ein Geschäftsprozess, kein Softwaresystem.

Web Service Design Quality Model

Ein Qualitätsmodell für das Design von *Web Services* stellen Shim u. a. [128] vor. Es beschreibt verschiedene Maße, die je nach benötigter Information und Granularitätsebene als dienstintern, dienstextern oder systemweit klassifiziert werden. Aus diesen (einfachen) Maßen werden in einem nächsten Schritt komplexere Maße abgeleitet, welche wiederum direkt mit Design-Eigenschaften des Systems verknüpft werden. Diesen Eigenschaften werden anschließend Einflüsse auf die Qualitätsattribute *Flexibilität*, *Effektivität*, *Wiederverwendbarkeit* und *Verständlichkeit* des Systems zugeordnet.

Exemplarisch wird der Ansatz im Folgenden für die Design-Eigenschaft *Consumability* nachvollzogen. Um Consumability zu messen, wird das abgeleitete Maß ANSOR (*Adequately Named Service and Operation Ratio*) definiert. Dieses setzt sich aus den Designmaßen SSNS (*System Size in Number of Services*), NINS (*Number of Inadequately Named Services*), NO (*Number of Operations*) und NINO (*Number of Inadequately Named Operations*) nach folgender Formel zusammen:

$$\text{ANSOR} = 0.5\frac{\text{SSNS} - \text{NINS}}{\text{SSNS}} + 0.5\frac{\text{NO} - \text{NINO}}{\text{NO}}$$

Es wird nicht explizit zwischen automatisch und manuell zu erhebenden Maßen unterschieden, wobei im Falle dieses Beispiels die Werte für NO und SSNS leicht automatisch zu bestimmen sind, während die Angemessenheit von Bezeichnern (für NINS und NINO) eine manuelle Einschätzung erfordert. Eine schematische Darstellung der Zusammenhänge enthält Abbildung 2.10.

Analog dazu werden für die Design-Eigenschaften *Coupling*, *Cohesion*, *Complexity*, *Design Size*, *Service Granularity* und *Parameter Granularity* abgeleitete Maße definiert und deren Einflüsse auf die o. g. Qualitätsattribute beschrieben. Das so gewonnene Qualitätsmodell wurde von den Autoren auf ein SOA-System vor und nach einer umfangreichen Restrukturierung angewendet. Die daraus ermittelten Differenzen der Kennzahlen wurden mit der beabsichtigten Qualitätsverbesserung abgeglichen. Dabei konnten alle erwarteten Veränderungen von Qualitätsattributen durch das Modell erklärt werden.

Da insgesamt nur eine geringe Anzahl an Maßen und Qualitätsattributen berücksichtigt wird, kann das Modell jedoch nicht als allgemeines Qualitätsmodell für SOA verstanden werden.

2.4.4 Fazit

Die betrachteten Arbeiten enthalten wichtige Ansätze und Konzepte für die Modellierung SOA-spezifischer Qualitätsaspekte. So liegen den Modellen von Balfagih und Hassan [21] sowie The European Network of Excellence in Software Services and Systems (S-Cube) [133] Analysen anderer Qualitätsmodelle zugrunde, wodurch sie bereits eine konsolidierte Sicht bieten.

Allerdings besitzen die betrachteten Arbeiten Schwachstellen. So sind z. B. im *Quality Reference Model for SBA* [133] die vorgeschlagenen Qualitätsattribute weder überschneidungsfrei definiert noch klar gegeneinander abgegrenzt. Darüber hinaus stehen sie in einigen Punkten nicht mit anderen Standards im Einklang, ohne dafür schlüssige Begründungen zu liefern.

Das OASIS *Quality Model for Web Services* [109] sieht durch das Konzept der *Quality Associates* explizit den Blickwinkel vor, unter dem Qualität betrachtet wird. Es strukturiert alle als relevant für SOA befundenen Qualitätsattribute in Form von *Quality Factors*. Allerdings ist weder eine klare Abgrenzung dieser Faktoren untereinander gegeben noch eine Möglichkeit, diese zu messen oder aus Messwerten zu berechnen.

Verschiedene Perspektiven auf die Qualität von SOA-Systemen werden auch in anderen Arbeiten präsentiert, etwa die des Nutzers im Modell von Choi, Her und Kim [41]. Ein Schwachpunkt des Modells ist die mangelnde praktische Evaluierung vieler verwendeter Konstrukte. Da das Modell ausschließlich die Nutzersicht beschreibt, lässt es sich erst dann anwenden, wenn das betreffende System bereits genutzt werden kann, also insbesondere nachdem die entsprechenden Dienste bereits entwickelt wurden und zur Verfügung stehen.

Balfagih und Hassan [21] gehen deshalb einen Schritt weiter und beschreiben SOA-Qualität sowohl für die Nutzer von Diensten als auch für deren Entwickler und Anbieter. So ist sichergestellt, dass auch zu früheren Zeitpunkten im Entwicklungszyklus bereits Qualitätsbewertungen durchgeführt werden können. Um dies tatsächlich zu tun, enthält das Modell allerdings keine ausreichend spezifizierten Maße, weshalb die Autoren es selbst auch nur als Schritt zu einem besseren Verständnis von SOA-Qualität bezeichnen.

Rud [123] adressiert das Problem der praktischen Anwendbarkeit, indem er ein

Qualitätsmodell auf Basis von Metriken definiert, die unabhängig von der konkreten technischen SOA-Umsetzung sind, aber dennoch Grundlagen für die automatisierte Erhebung von Messwerten liefern. Die Auswirkungen dieser Metriken werden jedoch lediglich in Form einer Analyse paarweiser Zusammenhänge untersucht. Einflüsse auf die Qualitätsattribute eines SOA-Systems werden nur am Rande betrachtet und nicht konsolidiert, weshalb es dem Anwender des Modells überlassen bleibt, die Messwerte zu interpretieren.

Auch Hündling [73] betrachtet keine standardisierten Qualitätsattribute, sondern definiert ein abstraktes Modell zur Einbindung von Qualitätsaussagen in bestehende SOA-Modelle. Allerdings enthält das vorgeschlagene Modell Konzepte zur Aggregation von Bewertungsergebnissen entlang von Kompositionshierarchien und zeigt damit einen Weg zur Qualitätsbewertung IT-gestützter Geschäftsprozesse auf. Die Möglichkeit, eine Qualitätsaussage über ein serviceorientiertes IT-System zu treffen, ist damit allerdings noch nicht unmittelbar gegeben.

Shim u. a. [128] präsentieren ein durchgehendes Qualitätsmodell, ausgehend von einfachen Maßen auf Service- und Systemebene über daraus abgeleitete Maße bis hin zur Darstellung von Einflüssen auf Qualitätsattribute. Damit ist es hinsichtlich der Modellierungstiefe unter den betrachteten Modellen weit entwickelt. Allerdings werden nur wenige Maße und Qualitätsattribute betrachtet, so dass keine zufriedenstellende Abdeckung in der Breite erreicht wird. Darüber hinaus wird zur Evaluierung das Modell lediglich angewendet, so dass die Relevanz und Aussagekraft der modellierten Konzepte nicht in ausreichendem Maße sichergestellt ist.

Insgesamt lassen viele Arbeiten im Bereich Qualitätsmodellierung für SOA eine ganzheitliche Betrachtung vermissen. Manche präsentieren hierarchische Dekompositionen relevanter Qualitätsattribute, andere erklären die Einflüsse von SOA auf bestimmte Attribute. Es existieren Modelle, die Maße zur Qualitätsbewertung bezüglich kleiner Teilmengen von Qualitätsattributen enthalten. Allerdings präsentiert keine der bisher betrachteten Arbeiten ein Modell, das sowohl wissenschaftlich belegt, als auch praktisch anwendbar ist, um die Qualität von SOA-Systemen zu bewerten.

2.5 Zusammenfassung

Die grundlegenden Konzepte aus den Bereichen SOA und Softwarequalität wurden wie folgt definiert: Der Qualitätsbegriff wird im Folgenden in Anlehnung an Garvin [62] verwendet und als multidimensionales Konstrukt verstanden. Insbesondere muss bei der Verwendung des Begriffs immer der Blickwinkel berücksichtigt werden, aus dem das Produkt betrachtet wird. Soweit nicht anders angegeben, bezieht sich die vorliegende Arbeit auf die Produktsicht, also die Bewertung des Produkts anhand seiner beobachtbaren Eigenschaften. In Anlehnung an die Klassifikation von Perry und Wolf [119] wird SOA dabei als Architekturstil verstanden, ohne dass eine konkrete technische Implementierung vorgeschrieben wird. Dabei unterscheidet sich die hier verwendete Definition von SOA dahingehend von vielen aus der Literatur bekannten, dass eine klare Abgrenzung zur Serviceorientierung als allgemeinem Paradigma gegeben ist.

SOA bezeichnet damit gewissermaßen die Projektion der Serviceorientierung auf die Dimension der Softwarearchitektur. Insbesondere werden damit organisatorische und unternehmerische Aspekte der Serviceorientierung von der weiteren Betrachtung ausgeschlossen, da diese nicht Gegenstand des zu entwickelnden SOA-Qualitätsmodells sind.

Als wiederkehrende Argumente für die Einführung von SOA wurden Flexibilität, Interoperabilität und Uniformität identifiziert und beschrieben. Inwieweit diese Eigenschaften von SOA unmittelbar erfüllt werden, ist allerdings nicht direkt ersichtlich. Es wurden sowohl die charakteristischen Merkmale von SOA und mögliche Auswirkungen auf die Softwarequalität betrachtet als auch existierende Arbeiten, die Qualitätsmodelle für SOA vorschlagen. Eine Analyse der Stärken und Schwächen dieser Modelle hat gezeigt, dass dort wichtige Konzepte betrachtet werden, allerdings nicht in einer Art, die sowohl eine vollständige Beschreibung als auch eine praktische Anwendung zur Architektur-Qualitätsbewertung erlaubt.

3 Literaturanalyse

3.1 Einleitung

In diesem Kapitel wird eine Literaturanalyse vorgestellt, welche zur Identifikation von Eigenschaften einer Softwarearchitektur durchgeführt wurde, die für das SOA-Qualitätsmodell von Bedeutung sind. Abschnitt 3.2 beschreibt die Vorgehensweise, welche der Literaturanalyse zugrunde liegt. In Abschnitt 3.3 werden die statistischen Kennzahlen der Analyse präsentiert. Abschnitt 3.4 gibt einen Überblick über Arbeiten, welche bezogen auf die Fragestellung besonders relevant erschienen. Die identifizierten Faktoren und deren Bedeutung im Kontext von SOA werden in Abschnitt 3.5 vorgestellt. Abschnitt 3.6 fasst das Kapitel zusammen.

3.2 Methodik

Für eine stichhaltige Literaturanalyse ist ein striktes Vorgehen erforderlich. Nachfolgend wird das genaue Vorgehen beschrieben, welches der Literaturanalyse zugrunde liegt.

Aufgrund der Vielzahl an Publikationen im Bereich der Softwarequalität wurde die Analyse auf wissenschaftliche Journale und Magazine eingeschränkt. Deren Auswahl wurde aufgrund anderer im SE durchgeführten Literaturstudien getroffen. Konkret wurden die Arbeiten von Glass, Ramesh und Vessey [64], Wohlin [145], Wong u. a. [146], Cai und Card [35] sowie Geist u. a. [63] als Grundlage verwendet, um eine umfangreiche Liste relevanter Publikationen zu erstellen. Eine detaillierte Aufstellung dieser Arbeiten ist in Tabelle 3.1 zu finden.

Daraus ergibt sich die in Tabelle 3.2 dargestellte Menge potenziell relevanter Journale und Magazine. Einige Publikationen wurden aufgrund ihrer inhaltlichen Ausrichtung von der weiteren Analyse ausgeschlossen, was durch entsprechende Fußnoten kenntlich gemacht wurde. Außerdem wurde das relativ neue Journal *IEEE Transactions on Services Computing* zusätzlich aufgenommen, da es eine besondere thematische Nähe aufweist und aufgrund seines Erscheinungsjahres 2008 für die o. g. Studien noch nicht verfügbar war.

Für die weitere Analyse wurden also 21 Journale und Magazine betrachtet. Aufgrund der Reife des Themas Softwarequalität wurde bewusst ein weiter Zeitraum gewählt, um keine relevanten Artikel allein aufgrund ihres Alters auszuschließen. Es wurden alle Veröffentlichungen ab dem Jahr 1986 einbezogen, sofern das betreffende Journal in diesem Jahr bereits existierte.

Ref.	Autor(en)	Titel
[35]	Cai und Card	*„An analysis of research topics in software engineering - 2006“*
[63]	Geist u. a.	*„Computing research programs in the U.S.“*
[64]	Glass, Ramesh und Vessey	*„An analysis of research in computing disciplines“*
[145]	Wohlin	*„An analysis of the most cited articles in software engineering journals - 2000“*
[146]	Wong u. a.	*„An assessment of systems and software engineering scholars and institutions (2002-2006)“*

Tabelle 3.1: Meta-Quellen der Literaturanalyse

3.3 Umfang und Auswahl

Die 21 betrachteten Journale und Magazine enthielten zwischen 1986 und 2010 insgesamt 17 294 Artikel. Der erste Schritt der Analyse bestand darin, diejenigen Artikel von der weiteren Betrachtung auszuschließen, die bereits aufgrund ihres Titels als themenfremd eingestuft werden konnten, was für 16 150 Artikel der Fall war. Im zweiten Schritt dienten die Abstracts der verbliebenen 1144 Artikel als Grundlage für die Zuweisung von Prioritäten von Null bis Neun, wobei Null für *komplett irrelevant* und Neun für *sehr wichtig* steht. Ein Histogramm dieser Analyse ist in Abbildung 3.1 dargestellt. Da allerdings bereits Artikel der Priorität Sieben maximal einzelne Aspekte von Softwarequalität betrachteten und somit pro Artikel nur einen sehr kleinen Beitrag für das Qualitätsmodell hätten leisten können, wurden im Folgenden ausschließlich Artikel der Prioritätenklassen Acht und Neun verwendet. Diesen wurde nach eingehender inhaltlicher Analyse basierend auf den betrachteten Eigenschaften von Software, den Qualitätsattributen und der verwendeten Methodologie ein Relevanzwert zwischen Eins und Neun zugeordnet. Eine Aufstellung dieser Werte ist in Abbildung 3.2 dargestellt.

Aufgrund der deutlichen Abgrenzung zwischen hohen und eher niedrigen Bewertungen der Relevanz, insbesondere durch Ausbleiben von Bewertungen im Bereich dazwischen, dienten die Artikel mit Werten zwischen Sieben und Neun als Grundlage für die detaillierte inhaltliche Analyse.

Tabelle 3.2: Für die Literaturanalyse betrachtete Journale

Titel	#
ACM Transactions on Computer Systems	1
ACM Transactions on Computer-Human Interaction	1
ACM Transactions on Database Systems	1
ACM Transactions on Graphics [1]	1
ACM Transactions on Information Systems	1
ACM Transactions on Modeling and Computer Simulation	1
ACM Transactions on Programming Languages and Systems	1
ACM Transactions on Software Engineering and Methodologies	5
Annals of Software Engineering	1
Automated Software Engineering	1
Empirical Software Engineering	2
IEE proceedings / Software	1
IEEE Software	4
IEEE Transactions on Computers [1]	1
IEEE Transactions on Knowledge and Data Engineering [1]	1
IEEE Transactions on Parallel and Distributed Systems [1]	1
IEEE Transactions on Pattern Analysis and Machine Intelligence [1]	1
IEEE Transactions on Services Computing [2]	0
IEEE Transactions on Software Engineering	5
IEEE Transactions on VLSI Systems [1]	1
IEEE/ACM Transactions on Networking [1]	1
Information and Software Technology	4
Int. Journal of Software Engineering and Knowledge Engineering	1
Journal of Software Maintenance – Research and Practice	1
Journal of Systems and Software	4
Journal of Systems Architecture [1]	1
Journal of the ACM [1]	1
Requirements Engineering [1]	1
Software – Concepts and Tools [1]	1
Software Practice and Experience	3
Software Process – Improvement and Practice [1]	1
Software Quality Journal	1
Software Testing, Verification and Reliability	2

[1] themenfremd
[2] neu

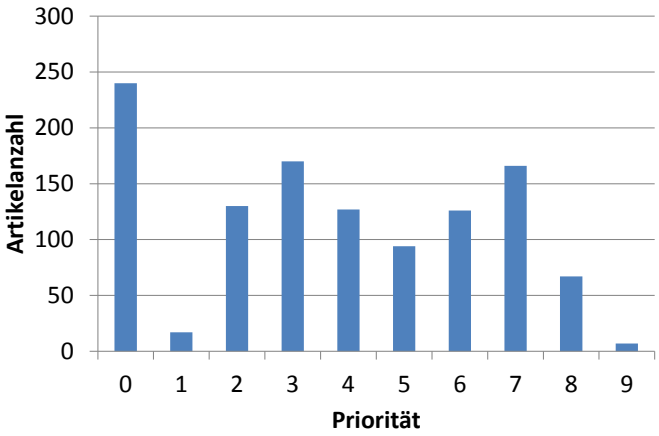

Abbildung 3.1: Artikel-Prioritäten der Literaturanalyse

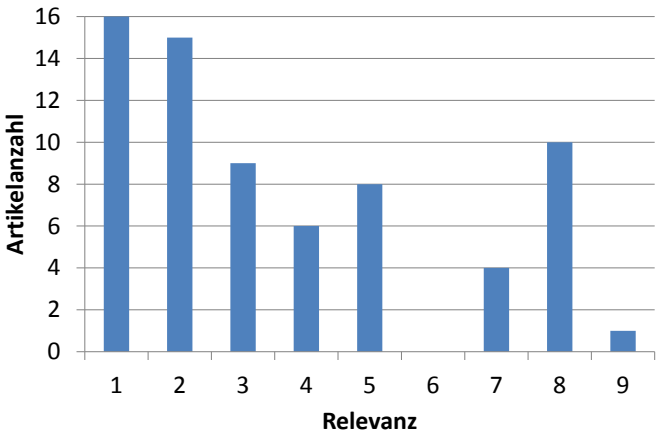

Abbildung 3.2: Artikel-Relevanz der Literaturanalyse

3.4 Zentrale Arbeiten

Im Folgenden werden einige der identifizierten relevanten Arbeiten einer eingehenderen inhaltlichen Analyse unterzogen. Im Gegensatz zu Abschnitt 3.5 werden dabei die Gesamtheit der Arbeit betrachtet und der verfolgte Ansatz zur Modellierung von Softwarequalität sowie die verwendete Methode zur Evaluierung der Ergebnisse vorgestellt und bewertet. Dabei werden auch solche Arbeiten berücksichtigt, die zwar zunächst als relevant erschienen, letztendlich aber keine umfangreichen Erkenntnisse zum SOA-Qualitätsmodell beitragen konnten.

3.4.1 Analyse der Änderbarkeit objektorientierter Software

Arisholm [17] analysiert in seinem Beitrag *„Empirical assessment of the impact of structural properties on the changeability of object-oriented software“* die Auswirkungen struktureller Eigenschaften objektorientierter Software auf deren Änderbarkeit. Damit ist die Perspektive eine ähnliche wie sie für das SOA-Qualitätsmodell eingenommen wird.

Die Arbeit enthält eine Reihe von Metriken zur Bestimmung struktureller Eigenschaften von Klassen sowie Vorschriften zur Aggregation von diesen klassenbasierten Messwerten auf das Gesamtsystem. Darüber hinaus wird eine Anpassung dieser System-Metriken unter Einbeziehung von Änderungen an diesen Klassen über die Zeit vorgeschlagen. Diese Berechnungsvorschriften sollen hier am Beispiel des Maßes *Class Size* veranschaulicht werden. Dabei ist $\mathrm{CS}(c)$ definiert als die Größe einer Klasse c in *Source Lines of Code (SLOC)*, woraus sich für das Gesamtsystem die Maße *TotCS* und *AvgCS* als Summe bzw. Mittelwert über alle Klassen ableiten. Das angepasste Maß $\mathrm{CS\_CP}_j$ für eine konkrete Änderung (etwa von einer Version zur folgenden) berechnet sich als

$$\mathrm{CS\_CP}_j = \sum_c \mathrm{CS}_j(c) \cdot \mathrm{CP}_j(c)$$

wobei das Änderungsprofil CP_j einer Klasse den Anteil der Modifikationen am System im Rahmen einer Änderung j beschreibt, die auf diese Klasse entfallen. Durch die Anpassung werden also diejenigen Klassen höher gewichtet, die im Rahmen der Änderung j stärker modifiziert wurden. Insbesondere tragen solche Klassen nicht zum Messwert bei, die unverändert blieben. Für die Korrelationsanalyse wurde der tatsächliche zeitliche Änderungsaufwand als abhängige Variable definiert. Für eine Änderung j bedeutet dies, dass im Falle der angepassten Metriken der Zusammenhang zwischen dem Änderungsaufwand und den Eigenschaften der von dieser Änderung betroffenen Klassen untersucht wurde.

Als generelles Fazit führt der Autor an, vor allem Größe und Kopplung der Software seien gute Indikatoren für deren Änderbarkeit. Die Metriken mit signifikanter Korrelation waren CS_CP, also die angepasste Klassengröße, sowie OMAEC_CP, der Zugriff auf Attribute von Klassen durch andere Klassen außerhalb ihrer Vererbungshierarchie, ebenfalls in der angepassten Form. Ersteres ist zwar prinzipiell interessant für

das SOA-Qualitätsmodell, allerdings stehen in der Schnittstellensicht keine Informationen über Quelltextzeilen zur Verfügung – andere Größenmaße wie etwa die Anzahl der Methoden zeigten jedoch im Experiment keine Signifikanz und sind deshalb nicht als Annäherung für einen Einflussfaktor für Änderbarkeit im SOA-Modell verwendbar. Welche Attribute ein Service intern definiert und verwendet, ist ebenfalls nicht an seiner Schnittstelle erkenntlich, so dass auch diese Metrik keine sinnvolle Anpassung an SOA erfahren kann und deshalb nicht weiter berücksichtigt wird.

Generell ist zu beobachten, dass viele der verwendeten Metriken detaillierte Informationen zur Implementierung enthalten. Aus diesem Grund lassen sie sich schlecht auf das SOA-Qualitätsmodell übertragen, das bewusst von der Programmiersprache abstrahiert und ausschließlich strukturelle Eigenschaften der Architektur beschreibt.

3.4.2 Qualitätsanalysemodell für objektorientierte Software

Die Arbeit von Bansiya und Davis [22] mit dem Titel *„A hierarchical model for object-oriented design quality assessment“* beschreibt ein Modell zur Qualitätsanalyse objektorientierter Software. Sie bildet die Grundlage für das Web-Service-Qualitätsmodell von Shim u. a. [128], welches in Abschnitt 2.4.3 beschrieben wurde.

Grundsätzlich argumentieren die Autoren, dass sich mit Einführung der Objektorientierung die Art und Weise verändert hat, wie Qualität bewertet wird. Insbesondere könne man Qualität früher im Entwicklungsprozess bewerten, weil bereits der Designprozess eines objektorientierten Systems dessen Struktur bestimme, und eine sinnvolle Qualitätsbewertung damit bereits auf Grundlage des Designs durchführbar sei. Man müsse allerdings bestehende Metriken für die Verwendung in OO-Systemen anpassen.

Zur Bestimmung der Komplexität objektorientierter Software schlagen die Autoren deshalb die Metrik *Number of Methods* vor, welche alle Methoden zählt, die innerhalb einer Klasse definiert werden. Im Gegensatz dazu beschreibt *Class interface size* die Anzahl der öffentlichen Methoden einer Klasse und wird zur Bestimmung der *Messaging*-Eigenschaft verwendet, mit deren Hilfe ausgedrückt wird, wie viel Funktionalität eine Klasse anderen Klassen zur Verfügung stellt. Da das SOA-Qualitätsmodell von der Implementierung abstrahiert, kann dort lediglich letztere Metrik Anwendung finden. Sie wird als M18: *Anzahl Operationen* in das Modell aufgenommen.

Die Abhängigkeit zwischen Klassen operationalisieren die Autoren durch *Direct Class Coupling*: *„[…] number of classes that a class is directly related to*“ [22], wobei der Begriff der Beziehung zwischen Klassen weit gefasst wird und explizit Klassen einschließt, welche innerhalb von Attributen der Klasse oder als Methodenparameter verwendet werden. Da sich in SOA die Konzepte *Dienst* und *komplexer Datentyp* fundamental unterscheiden, wird diese gemischte Definition im SOA-Qualitätsmodell nicht verwendet.

Die Argumentation der Autoren, dass Metriken angepasst werden müssen, um in einem neuen Programmierparadigma nutzbringend eingesetzt werden zu können, gilt für OO gleichermaßen wie für SOA. Auch die Möglichkeit, diese angepassten Metriken früher im Entwicklungsprozess einsetzen zu können als bisher, ist im Falle von SOA gegeben, da Dienste üblicherweise aus den Anforderungen von Geschäftsprozessen abge-

leitet werden („Service-Identifikation“, vgl. [69]), welche erst in einem nächsten Schritt in ein (objektorientiertes) Design überführt und schließlich implementiert werden.

3.4.3 Sensitivitätsanalyse von Softwarearchitekturen

Die Robustheit von Softwarearchitekturen ist das zentrale Thema der Arbeit mit dem Titel *„An Approach to Quantitative Software Architecture Sensitivity Analysis“* von Lung und Kalaichelvan [94]. Die Autoren schlagen eine strukturierte Vorgehensweise basierend auf verschiedenen Metriken vor, da in der Praxis Architekturanalysen oft ad-hoc durchgeführt würden und dies nicht zu vergleichbaren Ergebnissen führe.

Die für die Analyse vorgeschlagenen Maße werden von den Autoren in vier Gruppen zur Bestimmung von Größe, Struktur, Nutzung oder Abstraktionsgrad unterteilt. Die eigentliche Sensitivitätsanalyse setzt nicht die Nutzung spezieller Metriken voraus. Vielmehr wird eine Methode beschrieben, welche auf Basis zuvor festgelegter Metriken operiert. Welche Metriken verwendet werden, sei von mehreren Faktoren abhängig, inklusive dem analysierten Projekt selbst. Darüber hinaus sei das Thema der Architekturmetriken noch ein sehr unreifes Feld, so dass die Autoren nicht den Anspruch erheben, einen Satz universell geeigneter Metriken für ihre Methode zur Verfügung zu stellen.

Zur vorgeschlagenen Sensitivitätsanalyse wird ein szenariobasiertes Vorgehen gewählt: Zunächst werden Zielsetzungen analysiert, die sich entweder auf die Interessenshalter, die Architektur selbst oder die Qualitätsattribute der Software beziehen können. Anschließend werden diese Zielsetzungen in eine Menge von Szenarien überführt, welche sowohl die erwartete Nutzung des Systems als auch dessen Verhalten und mögliche zukünftige Änderungen einschließen. Basierend auf diesen Szenarien werden im nächsten Schritt geeignete Metriken ausgewählt. Anschließend werden für jede zuvor festgelegte Zielsetzung die notwendigen Änderungen am System identifiziert. Zum Abschluss wird analysiert, hinsichtlich wie vieler der ausgewählten Metriken sich das geänderte System vom ursprünglichen unterscheidet. Eine Änderung bezüglich vieler Metriken bedeutet dabei, dass die bestehende Architektur nicht robust gegenüber den geänderten Anforderungen ist.

Eine Analyse bezüglich sich ändernder Qualitätsattribute im Rahmen der zur Sensitivitätsanalyse durchgeführten Änderungen wird zwar als mögliche Erweiterung der beschriebenen Methode erwähnt, aber nicht weiter ausgeführt. Insbesondere machen die Autoren keine Angaben dazu, wie sich Qualitätsattribute überhaupt berechnen lassen – stattdessen wird eine Methode vorgeschlagen, in der Domänenexperten basierend auf verschiedenen Alternativen zu einer Änderung der Software abschätzen sollen, inwieweit sich einzelne Qualitätsattribute bei Durchführung dieser Änderung verändern würden.

Zusammenfassend erklären die Autoren, dass die Robustheit einer Architektur in großem Maße von ihrer Struktur, insbesondere der Kopplung, abhängt.

3.4.4 Heuristiken zur Wartbarkeit objektorientierter Software

Aus der Forschergruppe um Deligiannis fanden sich gleich zwei Arbeiten unter den in der Literaturanalyse ausgewählten [50, 51], wobei beiden das gleiche Experiment zu Grunde liegt: Es sollte ein Java-Programm auf seine Wartbarkeit hin untersucht werden. Dabei wurden den Teilnehmern sowohl der Quellcode (Java & AWT) als auch die dazugehörige Dokumentation (Textdokumente und UML-Diagramme) zur Verfügung gestellt. Das Programm lag in zwei Varianten vor, von denen eine aus 16 Klassen bestand und das *Anti-Pattern* der *God Class* enthielt, die andere aus 18 Klassen ohne eine solche „entartete“ Klasse. Daneben erhielten die Teilnehmer eine Aufgabenstellung zur Erweiterung des Programms um eine bestimmte Funktion. Die Wartbarkeit des Programms wurde dabei als abhängige Variable am Ergebnis dieser Änderung gemessen, insbesondere an dessen Vollständigkeit, Korrektheit und Konsistenz. Darüber hinaus wurde mittels eines Fragebogens die Verständlichkeit des Programms als zweite abhängige Variable bestimmt. Als unabhängige Variablen für das Experiment kamen verschiedene Design-Metriken zum Einsatz.

In Deligiannis u. a. [50] wurde das Experiment mit vier wissenschaftlichen Assistenten durchgeführt, in Deligiannis u. a. [51] mit 20 Studenten im zweiten Semester eines Informatikstudiengangs im Rahmen einer Vorlesung zur objektorientierten Programmierung. Als generelle Tendenz zeigten die Experimente, dass das Programm ohne *God Class* als etwas schwerer verständlich bewertet wurde, allerdings wesentlich besser wartbar war. Letzteres führen die Autoren vor allem auf eine stärkere Kohäsion der Klassen und eine einfachere Vererbungshierarchie zurück.

3.4.5 Dienstkohäsion und Analysierbarkeit

Perepletchikov, Ryan und Tari [118] führen in ihrem Artikel *„The Impact of Service Cohesion on the Analyzability of Service-Oriented Software“* an, Wartbarkeit sei ein großes Problem von SOA, was unter anderem der Tatsache geschuldet sei, dass SOAs oft ad-hoc entwickelt würden, was grundsätzlich zu Lasten der Wartbarkeit geschehe.

Die zentrale Frage, die die Autoren beantworten möchten, ist die nach dem Zusammenhang zwischen der Kohäsion von Diensten und der Analysierbarkeit des Softwaresystems, welches von diesen Diensten gebildet wird. Dazu werden verschiedene Kohäsionsmaße definiert, die unterschiedliche Formen der Kohäsion messen. Das Maß *service interface data cohesion* bestimmt dabei die sogenannte kommunikative Kohäsion, *service interface usage cohesion* die externe, *service interface implementation cohesion* die implementierte und *service interface sequential cohesion* die sequentielle Kohäsion. Darüber hinaus definieren die Autoren die Begriffe der zufälligen, der temporalen, der logischen und der konzeptuellen Kohäsion, die durch die präsentierten Maße jedoch nur indirekt bestimmt werden. Eine Gesamtaussage über die Kohäsion von Diensten in einem System wird durch Mittelwertbildung aus den vier Maßen abgeleitet.

Zur Evaluierung dieses Konstrukts wurde in einem Experiment die Analysierbarkeit einer Software mittels *Failure Analysis Efficiency* bestimmt, wobei die Teilnehmer Ur-

sachen für Fehler im System finden mussten. Die genaue Definition dieser Metrik ist im ISO/IEC-Standard 9126-4 [78] zu finden. Außerdem wurden die Studienteilnehmer befragt, in welchem Maße sie die betrachteten Dienste als konzeptuell kohäsiv empfinden. Mit dem Nachweis positiver Korrelation zwischen dem vorgestellten Kohäsionsmaß und beiden Vergleichsmetriken wurde sichergestellt, dass das vorgeschlagene Maß sowohl zur Beschreibung konzeptueller Kohäsion geeignet ist als auch Rückschlüsse auf die Analysierbarkeit der Software erlaubt. Die Autoren merken jedoch an, dass weitere Evaluierungen nötig seien, bevor diese Ergebnisse als allgemeingültig betrachtet werden könnten.

Im SOA-Qualitätsmodell sind die in dieser Arbeit vorgeschlagenen Maße als M24: *Gemeinsame Datentypen für Parameter*, M25: *Nutzung von Operationen durch Dienstkonsumenten* und M26: *Abfolge-Zusammenhang zwischen Operationen* repräsentiert. Das Maß zur implementierten Kohäsion wurde aufgrund seines immensen Erhebungsaufwands nicht in das SOA-Qualitätsmodell aufgenommen.

3.5 Einzelne relevante Faktoren

In Ergänzung zu den zuvor genannten, umfänglich für das SOA-Qualitätsmodell relevanten Artikeln wurde eine Reihe weiterer Arbeiten identifiziert, aus denen sich Faktoren für das Qualitätsmodell ableiten lassen. Diese sind im Allgemeinen nicht spezifisch für SOA, sondern stammen aus anderen Bereichen des Software Engineerings. In der Bestandsaufnahme zu qualitätsrelevanten Eigenschaften von SOA (Abschnitt 2.4 auf Seite 21) wurde bereits deutlich, dass die Umsetzung der SOA-Prinzipien allein nicht ausreicht, um eine hohe Produktqualität der Software sicherzustellen. Weitere beachtenswerte Faktoren werden im Folgenden thematisch gruppiert dargestellt und – sofern erforderlich – auf SOA angepasst.

3.5.1 Konsistente und präzise Bezeichner

Ein zentraler Bestandteil eines jeden Softwaresystems sind Bezeichner. Deißenböck und Pizka [48] argumentieren, eine Analyse des Quellcodes der Entwicklungsumgebung Eclipse habe gezeigt, dass 33 % aller Tokens Bezeichner seien und aufgrund ihrer Länge über zwei Drittel des gesamten Codes ausmachen. Dabei wachse die Bedeutung angemessener Bezeichner mit steigender Komplexität der Software. Die Autoren beschreiben verschiedene Methoden, um automatisierte Analysen von Bezeichnern zu ermöglichen, weisen aber zugleich darauf hin, dass eine ausschließlich automatische Analyse nicht möglich sei. Die beschriebenen Ansätze stellen folglich Hilfsmittel dar, um eine manuelle Analyse zu erleichtern. Unter anderem werden dabei Synonyme und Homonyme sowie zusammengesetzte Bezeichnungen identifiziert.

Auch Barnard [23] beschreibt die wichtige Rolle von Bezeichnern, vor allem in Bezug auf die Wiederverwendbarkeit von Software. Ihr Modell zur Bewertung von Wiederverwendbarkeit enthält entsprechende Maße sowohl für Klassen als auch für Methoden und deren Parameter. Im Gegensatz zum zuvor beschriebenen Ansatz werden dabei

allerdings keine Angaben zur Automatisierbarkeit der Analysen gemacht. Das Ergebnis der Analyse ist in jedem Fall eine ganze Zahl zwischen Eins (wenig bedeutsam) und Fünf (sehr bedeutsam).

Aufgrund der zentralen Rolle von Schnittstellenbeschreibungen und der darin enthaltenen Bezeichner wird deren Informationsgehalt auch im SOA-Qualitätsmodell untersucht. Dies wird durch die Maße M11: *Angemessenheit des Dienstnamens*, M12: *Angemessenheit des Operationsnamens* und M13: *Bedeutsame Parameter-Namen* ausgedrückt.

3.5.2 Angemessene Dokumentation

Ein wichtiger Aspekt für die Verständlichkeit von Software ist neben der aussagekräftigen Benennung der relevanten Konstrukte auch deren sinnvolle Dokumentation – sowohl bezogen auf externe Schnittstellen für die Benutzer der Software, als auch auf der internen, technischen Ebene, um die Wartung und Weiterentwicklung zu erleichtern.

Chen und Huang [40] präsentieren eine Literaturanalyse über Problemfaktoren in der Softwareentwicklung, wobei verstärkt auf Probleme im Entwicklungsprozess eingegangen wird. Bezogen auf die Dokumentation von Software identifizieren die Autoren drei zentrale Probleme: Mit *Documentation is obscure and untrustworthy (DOC1)* wird der Umstand beschrieben, dass Software weder klar verständlich noch zuverlässig dokumentiert ist. *Lack of integrity and consistency (DOC5)* bezieht sich ebenfalls auf den Informationsgehalt der Dokumentation und beschreibt die Tatsache, dass diese zuweilen unvollständig oder widersprüchlich ist. Schließlich wird mit *inadequacy of source code comments (PGM2)* auch der internen Dokumentation ein häufiger Mangel bescheinigt: Kommentare innerhalb des Quellcodes seien oft unzulänglich. Vor allem die beiden erstgenannten Probleme sind direkt auf serviceorientierte Systeme übertragbar, da auch die richtige und sinnvolle Verwendung von Diensten eine zuverlässige, verständliche und konsistente Dokumentation erfordert, auch wenn das Prinzip der gesteigerten Abstraktion von Schnittstellenbeschreibungen diese an die Begriffswelt der Anwendungsdomäne annähert.

Darüber hinaus wird auch in der Arbeit von Barnard [23] die Dokumentation auf ihre Auswirkungen hinsichtlich der Wiederverwendbarkeit von Software untersucht. Dabei werden unter anderem Designdokumente, Informationen zu Einschränkungen sowie Nutzungsinformationen und -beispiele genannt. Prinzipiell gilt dabei der Zusammenhang, dass die Wiederverwendbarkeit steigt, je mehr Dokumentation verfügbar ist. Im Speziellen wurde die angemessene Beschreibung von Klassen, Methoden und Parametern als wichtig identifiziert, die bloße Textmenge jedoch nicht. Dies legt den Schluss nahe, dass Dokumentation zumindest auch auf ihren Informationsgehalt und ihre Verständlichkeit hin untersucht werden sollte. Das Fehlen jeglicher Dokumentation sei jedoch in jedem Fall schlecht für die Wiederverwendbarkeit.

Das Dokumentation von Schnittstellen ist also auch für serviceorientierte Systeme von Bedeutung und wird als M10: *Bedeutsame Beschreibung* in das Qualitätsmodell aufgenommen.

3.5.3 Schnittstellenkomplexität

Komplexität von Artefakten in der Softwareentwicklung ist seit jeher ein aktiv erforschtes Thema, nicht zuletzt wegen des offensichtlichen Zusammenhangs zwischen Komplexität und Verständlichkeit. Aber auch die unzähligen Möglichkeiten, Komplexität zu definieren und zu messen, tragen dazu bei, dass diese Eigenschaft von Software in nahezu jede Betrachtung von Qualitätsaspekten einbezogen wird. Aus diesem Grund enthalten viele der analysierten Artikel Aussagen zur Komplexität von Software [22, 39, 50, 51, 57, 92, 106, 126].

Die Arbeit von Mouchawrab, Briand und Labiche [106] untersucht Einflüsse von Maßen unterschiedlicher Artefakte aus dem Entwicklungsprozess auf die Testbarkeit von Software. Zur Beschreibung der (von außen sichtbaren) Komplexität von Systemen verwenden die Autoren unter anderem das Maß *Number of system level operations*. Dabei werden öffentliche Operationen innerhalb von Klassen an der Systemgrenze gezählt. Die übrigen Maße für die Systemkomplexität erfordern das Vorhandensein einer Verhaltensbeschreibung in Form von Sequenzdiagrammen, so dass sie im Folgenden unberücksichtigt bleiben. Die Außensicht eines Systems ist allerdings auch für SOA bedeutsam, so dass dieses Maß als M16: *Operationen an der Systemgrenze* in das Qualitätsmodell aufgenommen wird.

Sharma, Kumar und Grover [126] präsentieren eine Definition für die Komplexität von Komponentenschnittstellen. Als Anwendungsbeispiel dienen dabei Java Beans, deren Schnittstellen neben Methoden auch Attribute aufweisen. Folglich sieht der Ansatz zur Komplexitätsbestimmung eine Kombination aus der Gesamtkomplexität der Methoden und jener der Attribute vor, wobei diese beiden Teile unterschiedlich gewichtet werden können. Die Komplexität einer Methode bestimmt sich demnach aus der Anzahl und Komplexität ihrer Parameter und ihrer Rückgabetypen. Dem entsprechenden Maß werden anschließend unter anderem Korrelationen mit Anpassbarkeit und Lesbarkeit der Software nachgewiesen.

Der größte Teil der analysierten Literatur enthält Angaben zur Komplexität von Klassen oder Artefakten ähnlicher Granularität, z. B. *Units* im Sinne von Quellcode-Einheiten, die im Rahmen eines Unit-Tests getestet werden. Li und Henry [92] beschreiben die Komplexität einer Klasse über die Anzahl ihrer Methoden.

Mouchawrab, Briand und Labiche [106] ziehen zur Bestimmung der Größe einer *Unit* die Anzahl der *Features* heran. Dabei unterscheiden sie zwischen geerbten und lokalen Features. Als Features werden unter anderem Methoden verstanden, wobei die Autoren auch hier differenzieren, insbesondere zwischen öffentlichen und privaten Methoden, aber auch zwischen solchen, die eine Schnittstelle implementieren, und anderen, die vererbte Methoden überschreiben. Außerdem fließen noch weitere Informationen in die Berechnung ein, etwa die Anzahl der implementierten Schnittstellen oder die Anzahl der deklarierten und der geerbten Attribute. Die Autoren geben an, dass die Signifikanz dieser Indikatoren für unterschiedliche Teststrategien unterschiedlich ausfallen kann. Für SOA kommen üblicherweise Tests der Dienst-Schnittstelle in Frage, so dass in diesem Fall die entsprechenden Maße der an der Schnittstelle beobachtbaren Faktoren eine wichtige Rolle spielen dürften, etwa die Anzahl der bereitgestellten Operationen,

welche am ehesten mit den öffentlichen Methoden einer Klasse vergleichbar sind. Auch Etzkorn, Hughes und Davis [57] beschreiben die Wichtigkeit dieser Kenngröße, wobei ihre Arbeit auf die Analyse der Wiederverwendbarkeit ausgerichtet ist. Darüber hinaus wurden bereits in den Abschnitten 3.4.2 und 3.4.4 die Arbeiten von Bansiya und Davis [22] bzw. Deligiannis u. a. [50] im Detail beschrieben, welche ebenfalls die Anzahl öffentlicher Methoden als Maß verwenden. Im SOA-Qualitätsmodell ist dieses Maß als M18: *Anzahl Operationen* vertreten.

Komplexität lässt sich auch von Artefakten feinerer Granularität bestimmen. So bestimmen Chen und Lu [39] die Komplexität von Operationen mittels ihrer *operation complexity metric*, welche durch einfache Addition zur Operationskomplexität einer Klasse aggregiert wird. Zur Bestimmung dieser Metrik wird jeder Operation anhand einer vorgegebenen Tabelle mit sieben Komplexitätsklassen ein Wert zwischen Null und 100 zugewiesen, wobei sich die Autoren auf Boehm [27] beziehen. Sharma, Kumar und Grover [126] beschreiben wie bereits erwähnt die Methodenkomplexität über Anzahl und Komplexität der Parameter- und Rückgabetypen.

Auch zur Bestimmung der Komplexität von Methodenparametern existieren mehrere Ansätze innerhalb der analysierten Literatur. Barnard [23] schlägt eine auf Wiederverwendbarkeit spezialisierte Metrik vor, nach der solche Parameter als besonders komplex angesehen werden, die viele Subtypen besitzen. In der weiteren Argumentation werden dann Methoden mit komplexen Parametern als schlecht wiederverwendbar bewertet. Dies ist besonders deshalb interessant, weil z. B. im Java-Umfeld viele auf Wiederverwendbarkeit ausgelegte Bibliotheken (etwa das Java-Collections-Framework) verstärkt Gebrauch von abstrakten Parametertypen machen, gerade um Wiederverwendbarkeit sicherzustellen. Chen und Lu [39] wählen einen Ansatz zur Bestimmung der Komplexität von Parametern, der auf Ebene ganzer Klassen ausgewertet wird, ohne Aussagen über einzelne Methoden zu treffen. Dazu wird jedem Methodenparameter einer Klasse mittels einer Tabelle ein Wert zwischen Null und Zehn zugewiesen, wobei klar definiert wird, welche Datentypen auf welche Werte abgebildet werden. Parameter vom Typ *Boolean* oder *Integer* werden z. B. mit Null bewertet, *Pointer* mit Fünf und Dateien mit Zehn. Anschließend werden die Werte aller Parameter einer Klasse zur Gesamt-Parameterkomplexität dieser Klasse addiert. Einen detaillierteren Ansatz beschreiben Sharma, Kumar und Grover [126]: Auch sie legen der Bewertung eine Tabelle zugrunde, die jedoch pro Methode einen Wert liefert, welcher sich sowohl aus der Anzahl als auch der Komplexität der einzelnen Parameter bestimmt. Über konkrete Datentypen werden allerdings keine Aussagen getroffen; es stehen lediglich Kategorien zwischen *simple* und *highly complex* zur Verfügung. Auch bleibt unklar, wie genau verfahren wird, wenn sich die Parameter einer Methode stark in ihrer Komplexität unterscheiden. Aufgrund der Unabhängigkeit von konkreten Datentypen und der Ermittlung von Messwerten pro Methode wird dieser Ansatz dennoch für das SOA-Qualitätsmodell adaptiert und im Maß M19: *Operationskomplexität* verwendet.

3.5.4 Granularität

Gerade im Hinblick auf Dienste spielt Granularität eine entscheidende Rolle, so dass dieses Konzept auch in anderen SOA-Qualitätsmodellen beschrieben wird (z. B. [123, 128], siehe Abschnitt 2.4.3). Auch in anderen Bereichen des Software Engineerings wurde der Einfluss von Granularität auf die Softwarequalität untersucht. So identifiziert Barnard [23] die *funktionale* Granularität als förderlich für die Wiederverwendbarkeit von Software. Dieses Konzept operationalisiert sie mittels der auf Methoden anzuwendenden Metrik *Number of functions performed*, wobei gefordert wird, dass Methoden exakt eine Funktion ausführen. Diese Metrik ist direkt in das SOA-Qualitätsmodell übertragbar und wird dort als M22: *Anzahl ausgeführter Funktionen* berücksichtigt.

Weitere analysierte Arbeiten betrachten vor allem die Granularität von Datentypen, die innerhalb der Software verwendet werden. Bansiya und Davis [22] ermitteln in ihrem Maß *Measure of Aggregation (MOA)* die Anzahl der benutzerdefinierten Klassen, die als Datentyp von Klassenattributen verwendet werden. In Schnittstellen von Diensten werden üblicherweise keine Attribute exponiert, sodass eine direkte Übertragbarkeit dieses Konzeptes nicht gegeben ist. In ähnlicher Weise, wenn auch detaillierter, beschreiben Li und Henry [92] mit dem Maß *data abstraction coupling (DAC)* die Anzahl abstrakter Datentypen (womit sie insbesondere Klassen meinen), welche in einer Klasse innerhalb von Variablendefinitionen verwendet werden. An einer anderen Stelle verweisen die Autoren in diesem Zusammenhang auf die Anzahl der Variablendefinitionen innerhalb einer Klasse, welche vom Typ einer anderen Klasse sind. Es wird also nicht klar, ob die Metrik mehrfache Verwendungen derselben Klasse berücksichtigt oder nicht. Letztendlich folgern die Autoren, dass die Metrik in Kombination mit weiteren Metriken zur Bestimmung der Wartbarkeit von Software geeignet ist. Auch hier ist aufgrund der Schnittstellensicht auf SOA keine direkte Übertragbarkeit der Metrik gegeben. Benutzerdefinierte Datentypen sind jedoch gerade in SOA von entscheidender Bedeutung, um Ein- und Ausgabedaten gemäß der Anwendungsdomäne zu definieren. Aus diesem Grund findet die Granularität von Parametern im Qualitätsmodell als M23: *Datengranularität* Beachtung. Dort wird allerdings eine angepasste Definition zugrunde gelegt, welche neben der bloßen Anzahl von Datentypen auch deren Gruppierbarkeit einschließt.

3.5.5 Abhängigkeiten

Bezüglich der Abhängigkeiten zwischen Diensten haben Analysen gezeigt, dass entgegen dem SOA-Prinzip der dynamischen Dienstbindung in der Praxis oft statische Abhängigkeiten zu beobachten sind. Aus diesem Grund wurden Studien dazu im Rahmen der Literaturanalyse näher betrachtet und deren Ergebnisse in das Qualitätsmodell übertragen. Aus der Menge der ausgewählten Publikationen sind hier vor allem Mouchawrab, Briand und Labiche [106], Barnard [23] sowie Bansiya und Davis [22] zu erwähnen, die Maße zur Bestimmung statischer Abhängigkeiten vorschlagen und diesen eine signifikante Bedeutung für die Produktqualität nachweisen.

Mouchawrab, Briand und Labiche schlagen dazu die Bestimmung von Abhängig-

keitspfaden vor, welche sie mit dem Attribut *Dependency Paths* beschreiben. Die entsprechende Metrik wird wie folgt definiert: „*Number of direct (or by transitive closure) dependent classes, with or without accounting for child classes*“ [106]. Da die Vererbung ein Konzept der objektorientierten Programmierung ist, wurde von der explizit erwähnten Möglichkeit der Betrachtung ohne Subklassen Gebrauch gemacht.

Auch innerhalb Barnards Studie zur Wiederverwendbarkeit von Software spielen Abhängigkeiten eine wichtige Rolle. Dort wird die Metrik „*number of calls made to other classes within a method*“ [23] definiert, welche sich von der zuvor genannten in einem wichtigen Punkt unterscheidet: Hier werden Methoden*aufrufe* gezählt, so dass Abhängigkeiten zu anderen Klassen stärker gewichtet werden, je häufiger von diesen Gebrauch gemacht wird. Dieser Unterschied erscheint für SOA nicht angebracht, da auf Schnittstellenebene die tatsächliche Anzahl von Aufrufen eine untergeordnete Rolle spielt.

Wie in Abschnitt 3.4.2 beschrieben wurde, definieren auch Bansiya und Davis [22] ein Maß zur Quantifizierung von Abhängigkeiten zwischen Klassen, welches jedoch aus den dort genannten Gründen nicht für eine Berücksichtigung im SOA-Qualitätsmodell geeignet ist. Insgesamt ergeben sich für das SOA-Qualitätsmodell also zwei Maße, welche die direkten (statischen) Abhängigkeiten eines Dienstes beschreiben: M1: *Fan-Out*, M30: *Abhängigkeitssumme*. Die durch transitive Betrachtung dieser Abhängigkeiten gewonnenen Abhängigkeitspfade gehen als M31: *Abhängigkeitstiefe* in das Qualitätsmodell ein.

3.6 Zusammenfassung

Für die Beantwortung der Frage, welche strukturellen Eigenschaften von Software relevant für die Produktqualität sind, wurden Artikel aus insgesamt 21 wissenschaftlichen Journalen der letzten 25 Jahre analysiert. Aus den über 17 000 Artikeln wurden in einem ersten Schritt 1144 als möglicherweise relevant ausgewählt und anschließend unter Einbeziehung des Abstracts priorisiert. Eine weitere Klassifizierung aufgrund von Einleitung und Zusammenfassung führte zu 15 Arbeiten, welche in diesem Kapitel beschrieben wurden und die Grundlage für den in Abschnitt 4.6 beschriebenen Teil des SOA-Qualitätsmodells bilden.

Schwerpunkte der analysierten Arbeiten waren die Namensgebung von Bezeichnern, Dokumentation, Komplexität, Granularität sowie Abhängigkeiten. Diese Kategorien finden sich in ähnlicher Weise im Qualitätsmodell wieder. Trotz unterschiedlicher Ausrichtung der analysierten Arbeiten, etwa in Bezug auf untersuchte Qualitätsaspekte wie Änderbarkeit, Wartbarkeit oder Wiederverwendbarkeit, lagen häufig ähnliche Maße zu Grunde, welche diese Qualitätsaspekte angemessen (und zumeist mit nachgewiesener Signifikanz) erklären konnten. Dies steigert die Bedeutung der betreffenden Maße für ein allgemeines Qualitätsmodell.

Eine Anpassung der verwendeten Maße, welche im Allgemeinen für unterschiedliche Programmiersprachen und -paradigmen definiert wurden, auf SOA war zumeist in eindeutiger Weise möglich. In jedem Fall wurden dabei getroffene Annahmen dargelegt.

ID	Bezeichnung
M1	*Fan-Out*
M10	*Bedeutsame Beschreibung*
M11	*Angemessenheit des Dienstnamens*
M12	*Angemessenheit des Operationsnamens*
M13	*Bedeutsame Parameter-Namen*
M16	*Operationen an der Systemgrenze*
M18	*Anzahl Operationen*
M19	*Operationskomplexität*
M22	*Anzahl ausgeführter Funktionen*
M23	*Datengranularität*
M24	*Gemeinsame Datentypen für Parameter*
M25	*Nutzung von Operationen durch Dienstkonsumenten*
M26	*Abfolge-Zusammenhang zwischen Operationen*
M30	*Abhängigkeitssumme*
M31	*Abhängigkeitstiefe*

Tabelle 3.3: Maße als Ergebnis der Literaturanalyse

Insgesamt werden aufgrund dieser Literaturanalyse die in Tabelle 3.3 dargestellten Maße im Qualitätsmodell verwendet.

4 Ein Qualitätsmodell für SOA

4.1 Einleitung

In den vorigen Kapiteln wurden bestehende und neu gewonnene Erkenntnisse über Softwarequalität in SOA-Systemen beschrieben. Das Ziel dieses Kapitels ist es, diese Ergebnisse in einem Qualitätsmodell zusammen zu fassen. Im Vergleich zu bestehenden, generischen Qualitätsmodellen gilt dabei besonderes Augenmerk den spezifischen Merkmalen von SOA.

Der Rest dieses Kapitels ist wie folgt gegliedert: Abschnitt 4.2 präsentiert grundlegende Definitionen hinsichtlich der Struktur des Qualitätsmodells. Die Elemente, welche eine SOA aus technischer Sicht konstituieren, werden in Abschnitt 4.3 in Form eines SOA-Referenzmodells beschrieben. Abschnitt 4.4 beschreibt und strukturiert Qualitätsziele innerhalb des Qualitätsmodells, welche in Anlehnung an die aktivitätenbasierte Qualitätsmodellierung mit Hilfe von Aktivitäten formuliert werden. Während Qualitätsziele eine übergeordnete, anwendungsnahe Strukturierungshilfe darstellen, sind Faktoren als Hauptbestandteil des vorgestellten Qualitätsmodells zu verstehen. Diese werden mit Hilfe von geeigneten Maßen operationalisiert, so dass die modellbasierte Bewertung der Qualität einer SOA ermöglicht wird. Abschnitt 4.5 enthält solche Faktoren und Maße, die zur Beschreibung und Operationalisierung der SOA-Prinzipien benötigt werden. Darauf aufbauend werden in Abschnitt 4.6 weitere Faktoren und Maße vorgestellt, um über die bloße Erfüllung der SOA-Prinzipien hinaus für die Softwarequalität relevante Eigenschaften der Architektur abzubilden. In Abschnitt 4.7 werden die zentralen Inhalte des Qualitätsmodells zusammengefasst und bewertet.

4.2 Strukturelle Aspekte

Um die Inhalte eines Qualitätsmodells möglichst konsistent und logisch zu präsentieren, bedarf es klarer struktureller Vorgaben bei der Modellierung. Dieses Ziel der möglichst eindeutigen Spezifikation wird oft in Form eines expliziten Metamodells verfolgt. Für Software-Qualitätsmodelle wurden deshalb u. a. das ABQM [49] und das Squale-Metamodell [105] entwickelt.

Ersteres verfolgt dabei den Ansatz, Qualitätsaspekte in Form von Aktivitäten auszudrücken, weil diese eine natürliche und eindeutige Semantik für die Dekomposition in Teilaktivitäten ermöglichen. Zur Bewertung der Qualität eines Softwaresystems wird die Kosteneffizienz der Durchführung dieser Aktivitäten herangezogen. Einzelheiten zum ABQM-Ansatz sind in Abschnitt 2.2.3 auf Seite 8 beschrieben.

Das Squale-Metamodell verwendet für die Beschreibung der Qualitätsattribute die Konzepte der ISO/IEC 9126, die es mit Hilfe von *Practices* operationalisiert. Innerhalb dieser Practices werden messbare Eigenschaften der Software zu den Qualitätsattributen in Bezug gesetzt. Ein detaillierterer Überblick über die verwendeten Konzepte ist in Abschnitt 2.2.3 auf Seite 10 zu finden.

Aus diesen und den spezifischen Gegebenheiten der SOA-Domäne lassen sich für die vorliegende Arbeit drei Modellierungsanforderungen von besonderer Bedeutung ableiten: Erstens soll das Modell eine klare Dekompositionssemantik aufweisen, um intuitiv verständlich zu sein und Mehrdeutigkeiten zu vermeiden. Konkret bedeutet dies, dass die hierarchische Struktur von Modellelementen eindeutig sein und einem klar definierten Muster folgen soll. Zweitens muss das Modell die spezifischen Eigenschaften von SOA berücksichtigen, um die relevanten Konzepte angemessen zu beschreiben. Die Unterscheidung zwischen generischen und SOA-basierten Softwaresystemen erfolgt demnach nicht ausschließlich auf inhaltlicher Ebene, sondern ist bereits im verwendeten Metamodell reflektiert. Drittens soll das Qualitätsmodell an die spezifischen Gegebenheiten bestimmter Anwendungsdomänen, Unternehmen oder Projekte anpassbar sein und deshalb über eine modulare Struktur verfügen. Viertens soll die Möglichkeit einer Operationalisierung, also der Anbindung von Maßen zur Qualitätsbewertung mit Hilfe des Qualitätsmodells, gegeben sein. Die Messbarkeit der Qualität von SOA-Softwaresystemen ist neben der Formalisierung und Strukturierung des Qualitätsbegriffs im Kontext von SOA ein wesentliches Ergebnis und Erfolgskriterium des im Rahmen dieser Dissertation entwickelten Qualitätsmodells. Die vier hier überblicksartig eingeführten Anforderungen werden in den folgenden Abschnitten sukzessive adressiert.

4.2.1 Allgemeine Modellelemente

Prinzipiell lässt sich eine Vielzahl von Systematisierungsstrategien zur Strukturierung von Qualitätsmodellen verwenden. So ist es z. B. denkbar, Modellelemente durch das mit dem Web 2.0 beliebt gewordene Zuweisen von Tags zu strukturieren. Neben den Vorteilen der Flexibilität und intuitiven Verständlichkeit birgt dies aber die Gefahr, dass eine semi-formale, flache und vor allem überlappende Strukturierung erfolgt, die die Domäne zwar möglicherweise detailliert beschreibt, dabei aber keine echte Gliederungshilfe ist. Dementsprechend ist auch heute in der Modellierung von Softwarequalität eine hierarchische Unterteilung in Qualitätsattribute und Teilattribute das Standardvorgehen. Dieser Ansatz wurde bereits in den Arbeiten von McCall, Richards und Walters [100] sowie Boehm u. a. [28] eingesetzt und ist auch in aktuellen internationalen Standards zu finden (z. B. ISO/IEC 25010 [77]).

Eine solche Unterteilung wurde mit dem Argument kritisiert, sie folge oft keiner eindeutigen Semantik. So könne die hierarchische Untergliederung nach verschiedensten (impliziten) Kriterien erfolgen, was sich nachteilig auf die Klarheit der Strukturierung auswirke [82]. Dieser Kritik wurde durch die Einführung aktivitätenbasierter Qualitätsmodelle entgegengewirkt [49]. Bei Aktivitäten, wie z. B. *Programmierung* oder *Wartung* von Software, ergibt sich aufgrund der möglichen Unterteilung in ein-

zelne Teilaktivitäten eine natürliche Gliederungssemantik und damit eine eindeutige Dekomposition. Das im Rahmen dieser Dissertation entwickelte Qualitätsmodell für SOA schließt sich diesem Stand der Wissenschaft an und verwendet eine aktivitätenbasierte Darstellung. Im Gegensatz zur dabei getroffenen Annahme, das Qualitätsziel sei immer die effiziente Durchführung einer Aktivität, wird allerdings mit dem SOA-Qualitätsmodell die explizite Charakterisierung der Aktivität mit einer gewünschten Eigenschaft gefordert. Diese Verallgemeinerung des ABQM-Konzeptes begründet sich auf der Beobachtung, dass die Forderung nach effizienter Durchführung im Allgemeinen nicht ausreicht, um sämtliche Aspekte von Softwarequalität zu beschreiben. Die gleiche Verallgemeinerung wird u. a. auch in der Arbeit von Lochmann [93] vorgenommen.

Einige Qualitätsmodelle (z. B. das S-Cube-Referenzmodell [133]) besitzen rein beschreibenden Charakter und verzichten auf die Möglichkeit der Bewertung von Systemen mit Hilfe des Modells. Um praktische Relevanz und Anwendbarkeit zu erreichen, ist in der vorliegenden Arbeit die Möglichkeit der Qualitätsbewertung explizit vorgesehen. Dabei werden die Beschreibungsebenen aus Abschnitt 2.2.3 (vgl. Abbildung 2.1 auf Seite 8) als Grundlage verwendet, also die konzeptuelle Dreiteilung in Ziele, Faktoren und Maße. Wie dort beschrieben ist es ein in der Qualitätsmodellierung übliches Vorgehen, zwischen Zielen (z. B. in Form von Qualitäts-Charakteristiken oder Anforderungen) und Faktoren (also beobachtbaren Eigenschaften bestimmter Systembestandteile) zu unterscheiden. Um Qualitätsmodelle praktisch anzuwenden sind außerdem Maße erforderlich, mit deren Hilfe man auf das Vorhandensein dieser Eigenschaften schließen kann. Zusammengefasst bedeutet dies, dass die Erreichung von Zielen beurteilt wird, indem Eigenschaften des Systems betrachtet werden, welche wiederum durch Maße quantifiziert werden. Dies entspricht dem GQM-Prinzip von Basili und Weiss [24].

Dieser Überblick über die verwendeten Modellelemente soll im Folgenden jeweils durch eine formelle Definition ergänzt und vertieft werden. Die Terminologie orientiert sich dabei an den früheren Arbeiten im Bereich der aktivitätenbasierten Qualitätsmodelle ([49, 141, 144]) sowie der im Forschungsprojekt Quamoco entwickelten Struktur für Qualitätsmodelle ([10, 11, 12]).

Definition 4.1 (Aktivität) *Eine* Aktivität *im Sinne des Qualitätsmodells ist eine Tätigkeit, die am oder mit dem System durchgeführt wird, und an deren Durchführung Qualitätsziele geknüpft werden können. Aktivitäten werden in serifenloser Schrift und eckigen Klammern dargestellt:* [Aktivität]*; die Menge aller Aktivitäten im Qualitätsmodell wird mit* $\mathbb{A}$ *bezeichnet.*

Definition 4.2 (Aktivitätsverfeinerung) *Eine Aktivität, die einen Teil einer anderen Aktivität darstellt, steht in einer* Verfeinerungsrelation *zu dieser:* $\mathbb{V}_A \subseteq \mathbb{A} \times \mathbb{A}$

Konkret werden Qualitätsziele durch die o. g. Aktivitäten beschrieben, wobei ein Ziel immer aus einer Aktivität und einem Attribut gebildet wird. In früheren Arbeiten zu aktivitätenbasierten Qualitätsmodellen sind diese Attribute lediglich implizit vorhanden, können allerdings zumeist aus der textuellen Beschreibung abgeleitet werden.

Im Fall des *Maintainability*-Modells von Deißenböck u. a. [49] liegt die Forderung nach *Kosteneffizienz* zugrunde; das *Security*-Modell von Wagner u. a. [141] fordert maximale *Ineffektivität*.

Definition 4.3 (Attribut) *Ein* Attribut *beschreibt eine beobachtbare Eigenschaft. Attribute werden in serifenloser Schrift und Kapitälchen dargestellt:* ATTRIBUT.

Definition 4.4 (Qualitätsziel) *Das Tupel aus Aktivität und Attribut bildet ein* Qualitätsziel. *Die Erreichung eines solchen Qualitätsziels wird auf dem Intervall [0;1] bewertet. Qualitätsziele werden in eckigen Klammern dargestellt, wobei Aktivität und Attribut durch eine senkrechte Linie voneinander getrennt sind:* [Aktivität | ATTRIBUT]*; die Menge aller Qualitätsziele im Qualitätsmodell wird mit* $\mathbb{Z}$ *bezeichnet.*

Definition 4.5 (Verfeinerung von Qualitätszielen) *Zum Zwecke der Konkretisierung können Qualitätsziele* verfeinert *werden. Ein Qualitätsziel* [X | Y] *kann dabei bezüglich seiner Aktivität zu* [X' | Y] *verfeinert werden, wobei* X' *eine Teilaktivität von* X *ist:* $(X, X') \in \mathbb{V}_A$. *Alternativ kann durch Spezifizierung des Attributs eine Verfeinerung zu* [X | Y'] *erreicht werden. Die Strukturierung aller Qualitätsziele im Qualitätsmodell bezüglich dieser Verfeinerung wird ausgedrückt als* $\mathbb{V}_Z \subseteq \mathbb{Z} \times \mathbb{Z}$.

Für die Beschreibung einzelner Elemente innerhalb eines Systems oder eines Dienstes werden diese dekomponiert. So ist die Schnittstelle Teil eines Dienstes, und eine Operation wiederum ein Teil der Schnittstelle. Diese Konzepte werden im Qualitätsmodell als Entitäten bezeichnet.

Definition 4.6 (Entität) *Eine* Entität *im Sinne des Qualitätsmodells ist ein Bestandteil des betrachteten Systems, dem eine Eigenschaft zugeordnet werden kann. Entitäten werden in serifenloser Schrift dargestellt:* Entität.

Eigenschaften, welche man am System oder dessen Entitäten beobachten kann, werden im Qualitätsmodell durch Faktoren beschrieben. Im Rahmen der Qualitätsbewertung wird für diese Faktoren bestimmt, zu welchem Grad eine Entität eine solche Eigenschaft besitzt. Die Granularität von Diensten wird somit im Qualitätsmodell durch den Faktor [Dienst | GRANULARITÄT] beschrieben.

Definition 4.7 (Faktor) *Ein Tupel aus Entität und Attribut wird als* Faktor *bezeichnet. Gültige Faktoren im Sinne des Qualitätsmodells sind solche, die sich in Form eines Erfüllungsgrades aus dem Intervall* $[0; 1]$ *bewerten lassen. Faktoren werden in eckigen Klammern dargestellt, wobei Entität und Attribut durch eine senkrechte Linie voneinander getrennt sind:* [Entität | ATTRIBUT]*; die Menge aller Faktoren im Qualitätsmodell wird mit* $\mathbb{F}$ *bezeichnet.*

Durch Einführung einer Verfeinerung können Faktoren zu höherwertigen Faktoren zusammengefasst werden. Diese Verfeinerung kann sowohl bezüglich des Attributs (z. B. FUNKTIONALE GRANULARITÄT als Verfeinerung von GRANULARITÄT) als auch bezüglich der Entität (z. B. Operation als Verfeinerung von Schnittstelle) erfolgen.

Definition 4.8 (Faktorverfeinerung) *Ein Faktor* verfeinert *einen anderen Faktor, wenn er ein spezifischeres Attribut derselben Entität oder dieselbe Eigenschaft einer spezifischeren Entität beschreibt als dieser. Die Menge aller Faktorverfeinerungen im Qualitätsmodell wird mit* $\mathbb{V}_F$ *bezeichnet:* $\mathbb{V}_F \subseteq \mathbb{F} \times \mathbb{F}$.

Analog zum ABQM können sich Faktoren entweder positiv oder negativ auf die Erfüllung von Qualitätszielen auswirken: [Schnittstellenbeschreibung | AKTUALITÄT] $\xrightarrow{+}$ [Wiederverwendung | EFFIZIENZ].

Definition 4.9 (Einfluss) *Ein* Einfluss *im Sinne des Qualitätsmodells beschreibt einen Zusammenhang zwischen einem Faktor und einem Qualitätsziel, welcher entweder positiv oder negativ für dessen Erfüllung sein kann. Einflüsse werden mittels Pfeilen dargestellt, die mit einem Plus- bzw. Minuszeichen versehen sind:* [Entität | ATTRIBUT] $\xrightarrow{+}$ [Aktivität | ATTRIBUT]. *Die Menge aller Einflüsse im Qualitätsmodell wird mit* $\mathbb{E}$ *bezeichnet.*

Maße dienen schließlich der Erhebung von Kenngrößen. Im Unterschied zu Faktoren haben Maße keine einheitliche Semantik und können auf unterschiedlichen Skalen und Wertebereichen definiert sein. Der Rückgabewert eines Maßes ist im Allgemeinen eine reelle Zahl; die Abbildung dieses Wertes für die Bewertung von Faktoren oder Qualitätszielen wird in Abschnitt 4.2.4 beschrieben.

Definition 4.10 (Maß) *Als* Maß *im Sinne des Qualitätsmodells wird eine Abbildung gewisser Charakteristiken einer Entität in eine quantitative Darstellung verstanden, die entweder durch ein Werkzeug oder manuell erfolgen kann. Maße werden kursiv in serifenloser Schrift dargestellt: Maß; Die Menge aller Maße im System wird mit* $\mathbb{M}$ *bezeichnet.*

4.2.2 SOA-spezifische Modellelemente

Neben den im vorigen Abschnitt eingeführten generischen Modellelementen bedarf es gewisser spezifischer Modellerweiterungen, um die Anwendungsdomäne SOA angemessen abzubilden. Dies ist vor allem der Tatsache geschuldet, dass sich Qualität in serviceorientierten Architekturen teilweise durch konsequente Anwendung der SOA-Prinzipien sicherstellen lässt, darüber hinaus aber auch weitere Faktoren eine Rolle spielen. In welcher Art und Weise sich die SOA-Prinzipien auf Qualitätsziele eines Systems auswirken, wurde bereits in Abschnitt 2.4 beschrieben. Dort ist darüber hinaus ersichtlich, dass nicht alle Qualitätsaspekte von den SOA-Prinzipien adressiert werden. Um die Qualität eines SOA-basierten IT-Systems zu beschreiben, sind also weitere Faktoren und Maße nötig. Diese greifen die Ergebnisse der Literaturanalyse aus Kapitel 3 auf und übertragen die dort für verschiedene Architekturstile und Programmierparadigmen identifizierten Faktoren auf SOA.

Im Rahmen der Forschungen zum SOA-Qualitätsmodell wurden verschiedene Ansätze verfolgt, um das Modell zu strukturieren. Zunächst wurde dabei die Anforderung in den Vordergrund gestellt, zwischen Aussagen über einen einzelnen Dienst und solchen

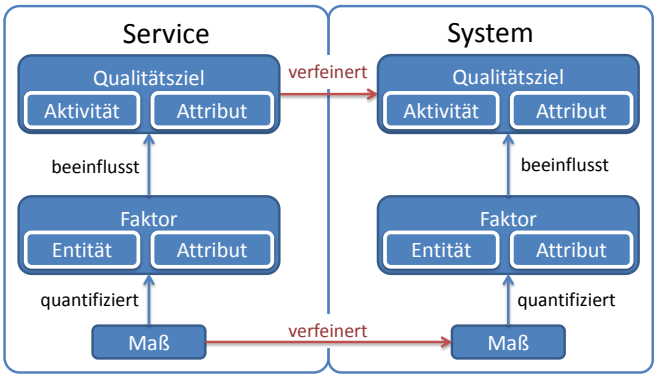

Abbildung 4.1: Alternative (verworfene) Modellstruktur

über ein gesamtes System unterscheiden zu können. Letzteres ist üblicherweise für den Anbieter der Infrastruktur von Bedeutung, während der Nutzer im Allgemeinen an der Qualität einzelner Dienste interessiert ist. Deshalb sah der ursprüngliche Modellierungsansatz explizit diese beiden Definitions- und Bewertungsbereiche vor, sodass Maße, Faktoren und Qualitätsziele jeweils einem dieser Bereiche zugeordnet wurden. Um die Auffindbarkeit eines Dienstes zu bewerten, kann etwa dessen Name auf seine Angemessenheit überprüft werden. In diesem Fall wäre *Service-Name* die Entität und *Service* der Geltungsbereich, dem der Faktor [Service-Name | ANGEMESSENHEIT] zugeordnet wurde. Abbildung 4.1 gibt einen Überblick über das Konzept; weitere Einzelheiten sind in Göb und Lochmann [3] veröffentlicht.

Im weiteren Verlauf der Forschung wurde dieser Ansatz jedoch verworfen, da zum Einen eine klare Trennung der Geltungsbereiche nicht in allen Fällen möglich war, zum Anderen die Struktur nicht unmittelbar die Beantwortung der Forschungsfrage unterstützte, welche spezifischen Qualitätseigenschaften SOA besitzt. Aus diesen Gründen wurde das Modell um das Konzept der *SOA-Prinzipien* erweitert und gleichzeitig die explizite Trennung von Service und System aufgegeben.

Definition 4.11 (SOA-Prinzip) *Ein SOA-Prinzip ist eine Eigenschaft eines Systems oder Dienstes, welche für SOA essentiell ist. In Anlehnung an die Literatur werden die folgenden SOA-Prinzipien definiert: Verschachtelte Wiederverwendung und Komposition, Verteilung von Funktionalität, Geschäftsprozessbezug und Fachlichkeit, Standardisierte Formate und Schnittstellen sowie Lose Kopplung und dynamische Dienstkomposition. Die Menge dieser SOA-Prinzipien wird mit* $\mathbb{P}$ *bezeichnet.*

Dieser Definition folgend sind SOA-Prinzipien eine spezielle Klasse von Faktoren. Da sie für das Qualitätsmodell von besonderer Bedeutung sind, werden sie als eigenständige (logische) Modellelemente dargestellt. In einer technischen Implementierung können sie durchaus als Faktoren modelliert sein, so dass etwa der Quamoco-Modelleditor[1] und dessen Dateiformat für Qualitätsmodelle (vgl. [45]) nicht modifiziert werden müssen.

[1] `https://quamoco.in.tum.de/tools`

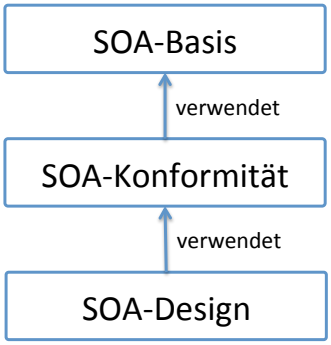

Abbildung 4.2: Modulare Struktur des Qualitätsmodells

4.2.3 Modularität des Qualitätsmodells

Die Unterscheidung zwischen Faktoren, die zur Erfüllung von SOA-Prinzipien beitragen, und solchen, die darüber hinaus Auswirkungen auf die Qualität eines Systems besitzen, bildet die Grundlage für eine Modularisierung des Qualitätsmodells. Das grundsätzliche Modularisierungskonzept folgt dem Ansatz, der innerhalb des Quamoco-Projekts entwickelt wurde und sich bereits in der Praxis bewährt hat [10].

Definition 4.12 (Modul eines Qualitätsmodells) *Ein* Modul *des Qualitätsmodells bezeichnet eine Untermenge seiner Elemente, die zur besseren Wartbarkeit des Modells oder aufgrund ihrer inhaltlichen Ähnlichkeit zusammengefasst werden sollen.*

Zwischen Modulen können Abhängigkeiten definiert werden: Hängt ein Modul A von einem Modul B ab, kann in der Definition seiner Faktoren und Maße auf die in Modul B definierten Faktoren und Maße verwiesen werden. Querverweise in der entgegengesetzten Richtung sind nicht zulässig.

Qualitätsziele wie z. B. Anpassbarkeit oder Effizienz werden als Anforderungen an einen Dienst oder ein System definiert und sind grundsätzlich von der Architektur unabhängig. Dies ist insbesondere bei Qualitätsstandards der Fall, die oft keine Annahmen über die Architektur der zu bewertenden Software enthalten (z. B. ISO/IEC 25010 [77]). Aus diesem Grund werden Qualitätsziele im übergeordneten Modul *SOA-Basis* definiert, so dass eine maximale Wiederverwendbarkeit gewährleistet ist. SOA-Prinzipien sowie Faktoren und Maße, die zu deren Beschreibung notwendig sind, werden im Modul *SOA-Konformität* zusammengefasst. Faktoren und Maße, die weitere Konzepte mit Einfluss auf die Softwarequalität beschreiben, bilden das Modul *SOA-Design*. Letzteres referenziert das Modul *SOA-Konformität*, da zur fundierten Bewertung der Qualität eines serviceorientierten Systems dessen Einhaltung der SOA-Prinzipien eine Voraussetzung ist. Allerdings kann diese auch überprüft werden, ohne dass darüber hinausgehende Qualitätsziele betrachtet werden. Diese Module und deren Abhängigkeiten sind in Abbildung 4.2 veranschaulicht. Die Aufteilung der Inhalte des Qualitätsmodells in diese beiden Module ist in Abbildung 4.3 schematisch dargestellt. Dieses Modularisierungskonzept kann außerdem zur Erweiterung und Anpassung des

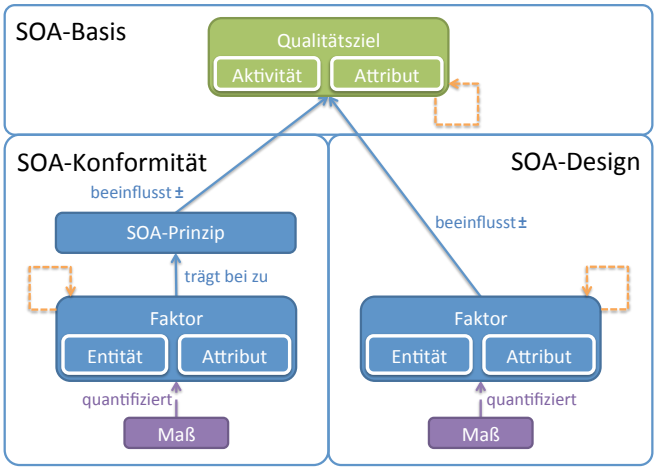

Abbildung 4.3: Schematischer Aufbau des Qualitätsmodells

Qualitätsmodells verwendet werden. Einerseits können projektspezifische Anforderungen in einem eigenen Modul ergänzt und durch Überschreiben von Bewertungsregeln auch projektspezifische Anpassungen in der Gewichtung von Einflüssen vorgenommen werden. Andererseits kann das Qualitätsmodell mit Hilfe weiterer Module für bestimmte Technologien konkretisiert werden, etwa indem für bestimmte Maße bei Kenntnis der verwendeten Technologie eine automatisierte Messung möglich ist.

Die im ursprünglichen Ansatz enthaltene und zugunsten der Klarheit des Modells aufgegebene Unterscheidung zwischen einzelnem Dienst und System kann zumindest bezogen auf die Qualitätsbewertung noch immer erreicht werden. Dazu wird das Objekt der Betrachtung entsprechend eingeschränkt, d. h. für die Analyse eines einzelnen Dienstes wird ein (virtuelles, weil nicht physisch vorhandenes) System definiert, welches nur aus diesem einen Dienst besteht. Die Bewertung erfolgt dann auf diesem System. Dieses Vorgehen erlaubt darüber hinaus auch die Bewertung von beliebigen Teilsystemen, etwa eines zusammengesetzten Dienstes und aller Dienste, welche bei dessen Nutzung zum Einsatz kommen, oder aller Dienste, die einen bestimmten Geschäftsprozess im System abbilden.

4.2.4 Modellelemente zur Qualitätsbewertung

Eine wesentliche Anforderung für die Praxistauglichkeit des entwickelten Qualitätsmodells ist es, Qualitätsattribute mit Hilfe von Maßen quantitativ zugänglich zu machen. Die bewusst offen gehaltene Definition der Maße, die keinerlei Einschränkungen bezüglich ihres Wertebereichs unterliegen, wird durch das Konzept der Bewertung vereinheitlicht und somit aggregationstauglich gemacht.

Die Bewertung erfolgt grundsätzlich auf dem Intervall $[0; 1]$, wobei Null für das Fehlen und Eins für das vollständige Vorhandensein der durch einen Faktor oder ein Qualitätsziel beschrieben Eigenschaft steht. Zur besseren Darstellung der Bewertungsergebnisse auf der Ebene von Qualitätszielen kann eine Transformation dieses Wertes

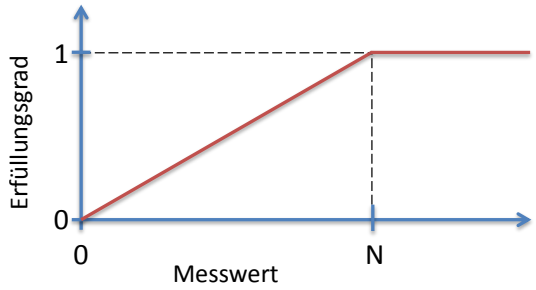

Abbildung 4.4: Abschnittsweise lineare Abbildung

auf eine andere Skala erfolgen, z. B. Punkte von 100, Ampelfarben oder andere Piktogramme.

Definition 4.13 (Bewertung) *Eine* Bewertung *ist eine Abbildung von zunächst beliebigen reellen Zahlen auf das Intervall* $[0;1]$. *Bewertungen können sowohl auf Qualitätszielen als auch auf Faktoren definiert werden. Die Menge aller Bewertungen im Qualitätsmodell wird mit* $\mathbb{B}$ *bezeichnet.*

Eine Bewertung kann z. B. mit einer abschnittsweise linearen Funktion realisiert werden, welche bei einem Messwert von Null auf den Wert Null abbildet und einen oberen Schwellenwert definiert, ab dem der Faktor mit Eins bewertet werden soll. Ein Beispiel für eine solche Funktion ist in Abbildung 4.4 dargestellt.

Eine Anpassung der Qualitätsbewertung an die Sichtweise verschiedener Personen und Rollen ist möglich, indem die Qualitätsziele unterschiedlich gegeneinander gewichtet werden. Hierzu kann das Qualitätsmodell durch ein weiteres Modul ergänzt werden, in dem für ein spezifisches Nutzungsszenario (z. B. die Bewertung eines Systems, für das gewisse Qualitätsziele besonders relevant sind) Bewertungen überschrieben werden. Auf diese Weise kann eine globale Hierarchie von Aktivitäten auf ein bestimmtes Rollenprofil angepasst werden, ohne tief greifende Änderungen im Modell vornehmen zu müssen. Soll in Zukunft eine weitere Perspektive auf die Produktqualität berücksichtigt werden, genügt es, einen neuen Satz an spezifischen Bewertungsvorschriften bereitzustellen, mit deren Hilfe die Bewertung des Systems vorgenommen wird. Eine detaillierte Beschreibung dieses Anpassungsvorgangs ist in Wagner u. a. [10] zu finden.

4.2.5 Zusammenfassung

Es wurde ein Metamodell für das SOA-Qualitätsmodell definiert, welches aus Qualitätszielen, SOA-Prinzipien, Faktoren, Maßen, Bewertungen und deren Beziehungen zueinander besteht.

$$\mathsf{QM} = (\mathbb{Z}, \mathbb{P}, \mathbb{F}, \mathbb{M}, \mathbb{B}, \mathbb{V}_Z, \mathbb{V}_F, \mathbb{E})$$

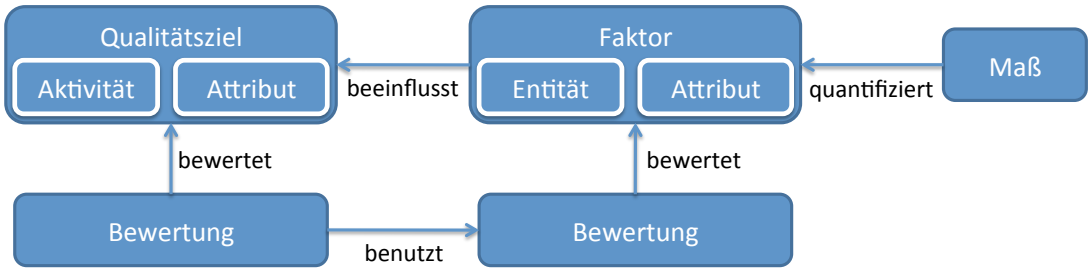

Abbildung 4.5: Elemente des SOA-Qualitätsmodells

Dabei sind Faktoren Tupel aus Entität und Attribut, während Qualitätsziele durch je eine Aktivität und ein Attribut bestimmt werden. Sowohl Qualitätsziele als auch Faktoren lassen sich durch sukzessive Verfeinerung konkretisieren.

Abbildung 4.5 veranschaulicht die beschriebenen Modellelemente und deren Beziehungen untereinander in einer vereinfachten schematischen Darstellung.

4.3 Entitäten und deren Beziehungen

Um die Elemente des Qualitätsmodells angemessen in den SOA-Kontext einzubetten, ist zunächst ein Referenzmodell der relevanten SOA-Konzepte erforderlich. Wie bereits in Abschnitt 2.3.4 auf Seite 16 erwähnt betrachtet die vorliegende Arbeit vor allem die technischen Aspekte von SOA, also insbesondere die Artefakte, die ein SOA-basiertes System aus IT-Sicht konstituieren. Innerhalb des Qualitätsmodells werden diese Artefakte als *Entitäten* repräsentiert, deren Granularität je nach Bedarf variiert. So wird etwa das gesamte IT-System als Entität dargestellt, ebenso wie ein Parameter einer Service-Operation.

Das hier vorgestellte SOA-Referenzmodell verwendet Teile der SOA-Ontologie der OpenGroup [134] sowie der Arbeiten von Méndez Fernández und Kuhrmann [102] und Hündling [73]. Aufgrund der unterschiedlichen Abstraktion der dort beschriebenen Konzepte werden diese nicht in ihrer Gesamtheit verwendet und einige weitere ergänzt.

Anzumerken ist außerdem, dass eine Entität im Modell immer ein *Konzept* beschreibt, keine konkrete Instanz innerhalb eines IT-Systems. Es ist demnach zwar möglich, Entitäten wie Service, Operation oder Parameter zu definieren, nicht aber einen konkreten Service *FlightReservation*, dessen Operation *Login* oder ein an diese übergebenes Passwort. Eine solche Instanziierung erfolgt erst während der Bewertung eines konkreten Systems, bei der die im Modell definierten Maße an einem konkreten System erhoben werden.

Aufgrund dieser technischen Sicht auf die Bestandteile einer SOA wurden verschiedene SOA-Modelle betrachtet und auf ihre Tauglichkeit bezüglich einer Wiederverwendung innerhalb der Entitäten-Hierarchie des Qualitätsmodells untersucht. Der folgende Abschnitt stellt einige solche Modelle vor und diskutiert die Kompatibilität der betrachteten Konzepte mit dem Entitätenmodell.

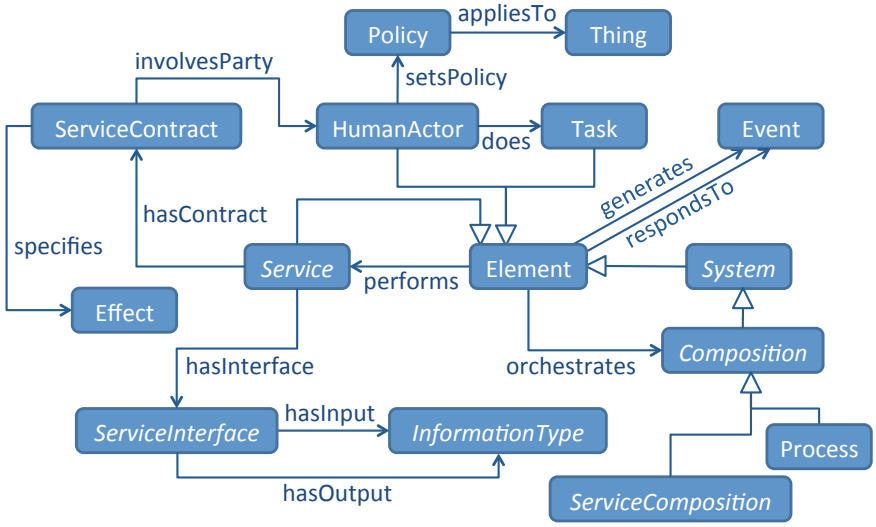

Abbildung 4.6: SOA-Ontology – schematische Darstellung (Quelle: [134])

4.3.1 SOA-Modelle

OpenGroup SOA Ontology

Um den zentralen Konzepten von SOA eine semantische Grundlage zu verleihen, hat die *SOA Work Group* der *OpenGroup* eine SOA-Ontologie [134] entwickelt und Ende 2010 herausgegeben.[2] Diese verfolgt das Ziel, die zentrale Terminologie im SOA-Umfeld festzuhalten und damit die Kommunikation zwischen der Geschäfts- und der IT-Welt zu verbessern sowie Missverständnisse bei der SOA-Einführung zu vermeiden. Nach Angabe der Autoren seien begriffliche Unklarheiten ein Grund für mangelnden Erfolg von SOA-Lösungen und für Interoperabilitätsprobleme. Die Ontologie richtet sich gleichermaßen an Geschäftsleute wie an Architekten und Softwareentwickler. Eine schematische Darstellung der Kernkonzepte der Ontologie ist Abbildung 4.6 zu entnehmen.

Der Abstraktionsgrad ist dabei ähnlich dem für das Qualitätsmodell vorgesehenen, so dass zumindest ein Teil der definierten Konzepte und Zusammenhänge direkt übertragbar sind, etwa hinsichtlich der Begriffe *System*, *Service* und *Service Interface*. Weitere Konzepte wie *Dienstkomposition* und *Geschäftsprozess* werden ebenfalls im Rahmen des SOA-Referenzmodells betrachtet, allerdings nur dann als Entitäten im Qualitätsmodell referenziert, wenn sie zur Bildung von Faktoren erforderlich sind.

Artefaktmodell der Technischen Universität München

Im Kontext der Anforderungserhebung für Software wurde an der Technischen Universität München ein artefaktorientiertes Vorgehen entwickelt (*Artefact-based Requirements Engineering and its Integration into a Process Framework*, Méndez Fernández

[2] siehe auch Pressemeldung unter `http://opengroup.org/press/08dec10.htm`

und Kuhrmann [102]). Im Rahmen dessen wurden Konzepte sowohl aus dem Umfeld von Qualitätsanforderungen als auch von IT-basierten Service-Infrastrukturen beschrieben und miteinander in Beziehung gesetzt. Es entstand eine umfangreiche Referenz, die im Folgenden überblicksartig beschrieben wird. Das vorgestellte *Business Information Systems' Analysis (BISA)*-Rahmenwerk führt vier Modelle ein, mit deren Hilfe Anforderungen an betriebliche Informationssysteme in den Entwicklungsprozess eingebunden werden: Dabei beschreibt das *Abstraktionsmodell* für Artefakte sämtliche für die Spezifikation von Anforderungen benötigten Konzepte und setzt diese entlang einer horizontalen und einer vertikalen Abstraktion miteinander in Beziehung. Die vertikale Abstraktion beschreibt den schrittweisen Übergang vom Kontext des Unternehmens und dessen Prinzipien und Zielsetzungen über Geschäftsprozesse bis hin zur Spezifikation eines IT-Systems, welches diese Prozesse abbildet. In horizontaler Richtung wird auf all diesen Ebenen zwischen Umgebung, Verhalten, Struktur und Information unterschieden. Das *Rollenmodell* definiert Verantwortlichkeiten für die verwendeten Artefakte, während das *Prozessmodell* die Artefakte durch Festlegung von Meilensteinen und Vorgehensweisen in den Entwicklungsprozess einbindet. Schließlich beschreibt das *Zustandsmodell* für jedes Artefakt, welche Zustände es annehmen kann, und welche Aktionen (etwa zur Qualitätssicherung) erforderlich sind, um es fertigzustellen. Mittels dieser Zustandsautomaten kann während des gesamten Entwicklungsprozesses der Fortschritt jedes Artefakts beobachtet werden.

Im Folgenden gilt besonderes Augenmerk dem im Vergleich zum SOA-Qualitätsmodell unterschiedlich verwendeten Konzept der *Aktivität*. Im Artefaktmodell ist eine Aktivität Teil eines Geschäftsprozesses und kann durch einen (IT-basierten) Dienst repräsentiert sein. Aktivitäten sind also Teil der Ausführung eines Prozesses und können durch Personen oder durch Services realisiert werden. Diese Definition unterscheidet sich grundlegend von jeder, die im Kontext aktivitätenbasierter Qualitätsmodelle verwendet wird. Dort – wie auch insbesondere in der vorliegenden Dissertation – bezeichnet eine Aktivität nicht notwendigerweise einen Vorgang innerhalb der Funktionalität des IT-gestützten Geschäftsprozesses. Vielmehr beschreibt sie die Interaktion mit dem entsprechenden IT-System, die über die Ausführung hinaus geht und Tätigkeiten wie die Analyse und Modifikation des Systems beschreibt. Darüber hinaus werden in dieser Definition Aktivitäten grundsätzlich von Personen ausgeführt, wodurch sie mit den *Actor Actions* aus dem Artefaktmodell vergleichbar sind.

Service-Modell nach Hündling

In der Dissertation von Hündling [73] wird ein Modell vorgestellt, das Konzepte aus dem Umfeld von Services miteinander in Beziehung setzt. Dabei werden die Konzepte *Schnittstelle*, *Operation*, *Nachrichtentyp*, *Kommunikationskanal*, *Endpunkt*, *Service* sowie *Service-Typ* definiert und gemäß Abbildung 4.7 verwendet.

Das Modell geht von einem Softwaresystem aus, welches aus Komponenten besteht. Die Kommunikation mit diesen Komponenten wird durch *Schnittstellen* spezifiziert, welche wiederum aus mehreren *Operationen* bestehen. Eine solche Operation besitzt mindestens einen Parameter zur Ein- oder Ausgabe von Daten, wobei Parameter im

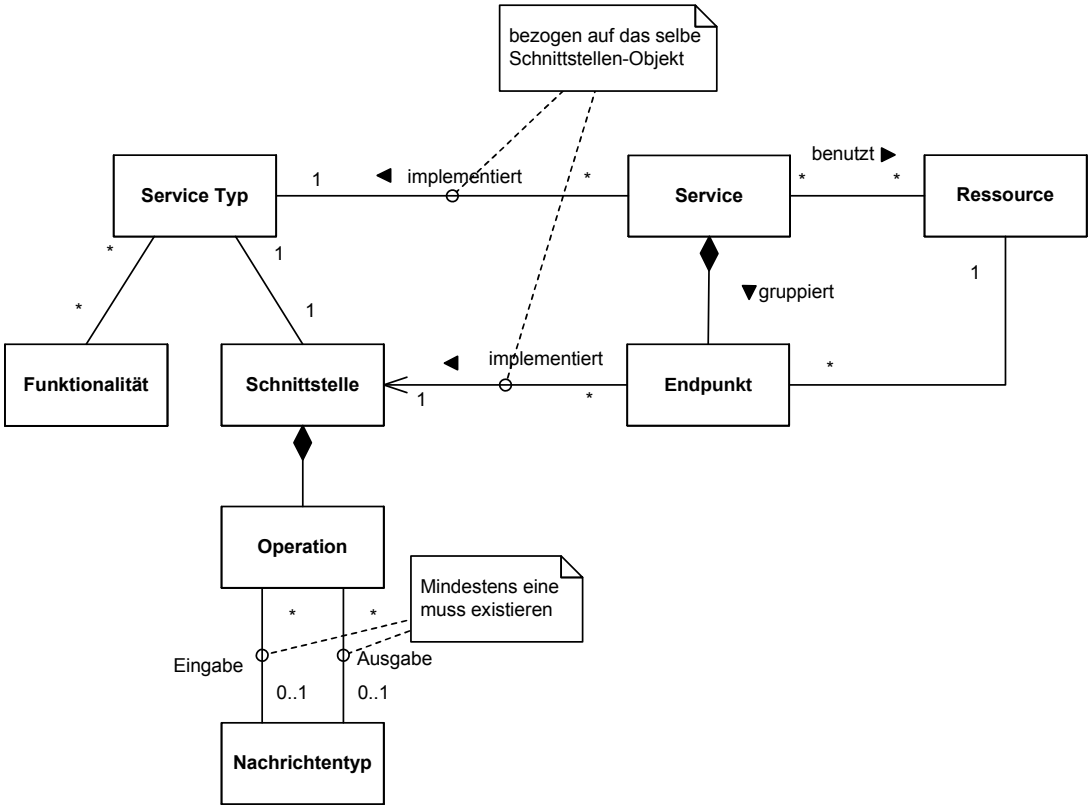

Abbildung 4.7: Service-Modell nach Hündling (Quelle: [73])

Service-Modell durch Nachrichten implementiert sind. Aufbau und Struktur dieser Nachrichten werden mittels des *Nachrichtentyps* definiert.

Der Übertragungsweg, über den mit einer Komponente interagiert wird, wird *Kommunikationskanal* genannt; die Anbindung einer Komponente an einen Kommunikationskanal wird als *Endpunkt* bezeichnet, wobei jede Komponente über mehrere Endpunkte verfügen kann.

Ein *Service* wird als eine in einem IT-System angebotene Leistung verstanden, für die eine Beschreibung veröffentlicht wird. Darin müssen sowohl die Funktionalität als auch die Kommunikation mit dem Anbieter spezifiziert sein. Da eine bestimmte Funktionalität typischerweise durch mehrere Services realisiert wird, führt Hündling darüber hinaus das Konzept des *Service-Typs* ein, welcher mehrere Services mit gleicher Funktionalität und gleicher Schnittstelle zusammenfasst. Anbieter von Services können standardisierte Service-Typen implementieren und so von potenziellen Kunden leichter gefunden werden – diese wiederum können bei Bedarf einen Service gegen einen anderen des gleichen Service-Typs austauschen, ohne dass sich Änderungen an der Funktionalität oder der Schnittstelle ergeben.

Diese Konzepte stimmen weitgehend mit den hier verwendeten Entitäten überein, so dass eine starke Kompatibilität gewährleistet ist. Einzig der Kommunikationsaspekt wird im folgenden Abschnitt weniger stark betrachtet und deshalb nicht weiter ausdifferenziert. Die Beschreibung der Funktionalität und der Schnittstelle wird allerdings als essentiell angesehen. Darüber hinaus könnten die Dokumentation von Services sowie Bezeichner von Services und Operationen aufgrund ihrer großen Bedeutung als eigene Entitäten repräsentiert werden.

4.3.2 SOA-Referenzmodell

Dieser Abschnitt beschreibt diejenigen SOA-Konzepte, die für das Qualitätsmodell von Bedeutung sind, und ihre Beziehungen untereinander. Eine Übersicht dieser Konzepte ist in Abbildung 4.8 und Abbildung 4.9 dargestellt; Details zu den einzelnen Konzepten werden in den folgenden Abschnitten erläutert.

IT-System Ein IT-System ist die Kombination aus Hardware und Software mit dem Ziel eine bestimmte Aufgabe zu erfüllen. Bezüglich der Hardware lässt sich ein SOA-basiertes IT-System in verschiedene Netzwerkknoten (siehe unten) unterteilen; bezüglich der Software erfolgt eine Untergliederung in Dienste.

Softwaresystem Das Softwaresystem (vereinfachend im Folgenden auch *System*) bezeichnet den in Software realisierten Teil eines IT-Systems. Damit repräsentiert es die Gesamtheit der Softwareartefakte, deren Qualität mit dem Qualitätsmodell beschrieben und analysiert werden soll. Da sich diese Arbeit mit der Softwarequalität serviceorientierter IT-Systeme befasst, handelt es sich bei diesen Softwareartefakten im Allgemeinen um Dienste (Services).

Netzwerkknoten Mit Blick auf die verwendete Hardware lässt sich ein (serviceorientiertes) IT-System in Einheiten unterteilen, die Services anbieten, über eine eigene *Central Processing Unit (CPU)* verfügen und untereinander über Netzwerkverbindungen kommunizieren. Diese Einheiten werden hier als *Netzwerkknoten* bezeichnet. Üblicherweise handelt es sich bei diesen um *Server* oder *Appliances*, allerdings können auch eingebettete Systeme wie Sensoren oder Aktuatoren Services innerhalb des Systems anbieten und somit als Netzwerkknoten im Sinne dieser Definition agieren.

Schnittstelle Schnittstellen beschreiben, in welcher Form mit einem System oder innerhalb desselben zwischen verschiedenen Diensten interagiert werden kann. Sie legen fest, welche Funktionalität an den Grenzen eines Dienstes exponiert ist.

Systemgrenze Ein serviceorientiertes IT-System besteht hardwareseitig aus einer Menge von Netzwerkknoten und softwareseitig aus einer Menge von Diensten. Demnach kann auch die Systemgrenze in diesen Dimensionen beschrieben werden. Bezüglich der Hardware wird die Systemgrenze von denjenigen Netzwerkknoten gebildet, welche einerseits zum betrachteten System gehören, andererseits aber von außerhalb dieses Systems über das Netzwerk erreichbar sind. Analog dazu besteht die Software-Systemgrenze aus denjenigen Schnittstellen, die nicht ausschließlich innerhalb der Hardware-Systemgrenzen verwendet werden können.

Geschäftsprozess Ein Geschäftsprozess ist die Formalisierung wiederkehrender Tätigkeiten aus dem Geschäftsalltag. Ein Beispiel für einen Geschäftsprozess ist der *Order-To-Cash*-Prozess, der die Bearbeitung einer Bestellung bis hin zur Lieferung und Bezahlung abbildet.

Prozessschritt Geschäftsprozesse werden in Prozessschritte aufgeteilt, von denen einige manuell durchgeführt werden können, andere vollständig durch IT-Systeme. Letztere werden üblicherweise durch Services repräsentiert (siehe *Prozessdienst*).

Service Ein Service ist eine durch ein IT-System bereitgestellte Funktionalität mit öffentlich zugänglicher Beschreibung der Schnittstelle und des Kommunikationswegs. Andere Definitionen von *Service* schließen auch andere Arten der Leistungserbringung ein, etwa eine manuelle wie z. B. *Haare Schneiden*. Das SOA-Qualitätsmodell betrachtet allerdings ausschließlich Software, weshalb eine entsprechende Eingrenzung des *Service*-Begriffs die Terminologie der folgenden Kapitel erheblich vereinfacht. Dienste lassen sich aufgrund ihres Abstraktionsgrades in eine der folgenden drei Kategorien einteilen.

Prozessdienst Ein Dienst, der Geschäftsprozess-Funktionalität bereitstellt, wird als *Prozessdienst* bezeichnet. Üblicherweise werden Prozessdienste dergestalt angeboten, dass die Funktionalität des Dienstes genau dem entspricht, was zur Bearbeitung eines

Prozessschrittes erforderlich ist. Aufgrund dieser groben Granularität sind Prozessdienste im Allgemeinen zusammengesetzte Dienste – außer im Falle der Kapselung eines Altsystems als Dienst.

Basisdienst Ein Dienst, der grundlegende, atomare Funktionalität bereitstellt, wird als *Basisdienst* bezeichnet. Insbesondere sind Basisdienste nicht spezifisch für eine bestimmte Anwendungsdomäne definiert, sondern zeichnen sich durch domänenübergreifende Wiederverwendbarkeit aus (siehe *Cross Domain Utility Layer*[3]).

Hilfsdienst Ein Dienst, dessen Funktionalität weder atomar ist, noch unmittelbar einem Geschäftsprozess zugeordnet werden kann, wird als *Hilfsdienst* bezeichnet. Typischerweise konsumiert ein solcher Dienst einen oder mehrere Basis- oder Hilfsdienste und stellt seine Funktionalität einem oder mehreren Hilfs- oder Prozessdiensten zur Verfügung. Hilfsdienste sind grundsätzlich zusammengesetzte Dienste.

Service-Dokumentation Über den Namen und die Schnittstelle von Services hinaus sind weitere Beschreibungen denkbar, etwa Hinweise zur Benutzung oder genauere textuelle Beschreibung der Funktionalität. Alle Arten das Verhalten oder die Verwendung von Diensten zu beschreiben werden unter dem Begriff *Service-Dokumentation* zusammengefasst.

Operation Einzelne Elemente einer Schnittstelle werden als Operationen bezeichnet. Eine Operation ist ein Interaktionspunkt einer Komponente, wobei die Gesamtheit der Operationen die Schnittstelle bilden. Wichtige Eigenschaften einer Operation sind dabei ihr Name und ihre Parameter, welche zusammen ihre Signatur bilden. Eine Operation benötigt zumindest einen Ein- oder Ausgabeparameter.

Parameter Parameter dienen der Übermittlung von Daten zwischen dem Anbieter und dem Nutzer einer Funktionalität. Das Format und die Struktur von Parametern wird durch Datentypen bestimmt. Man unterscheidet zwischen einfachen und komplexen Datentypen, wobei z. B. *Ganzzahl* ein einfacher Datentyp ist und *Person* ein komplexer.

[3] `http://www.soapatterns.org/cross_domain_utility_layer.php`

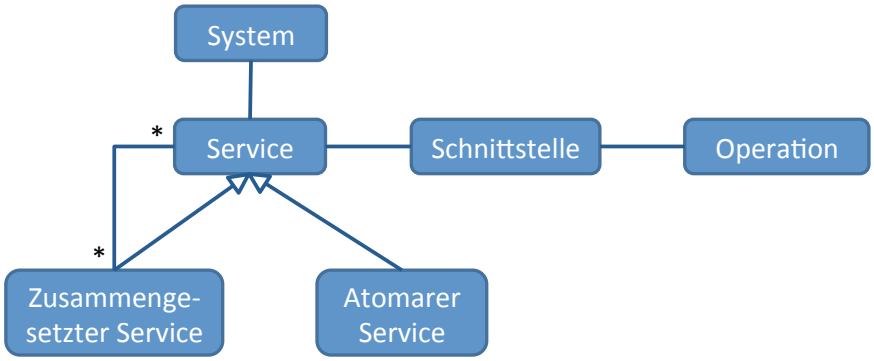

Abbildung 4.8: SOA-Referenzmodell – Dienste

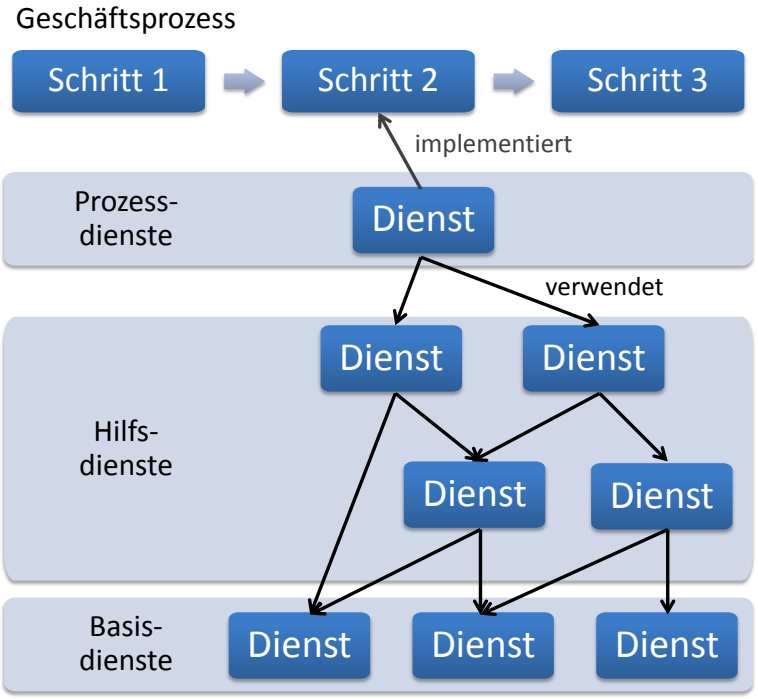

Abbildung 4.9: SOA-Referenzmodell – Überblick

4.4 Aktivitätenbasierte Qualitätsziele

Zur Beschreibung der Qualitätsattribute eines SOA-Systems werden im Folgenden Aktivitäten verwendet. Eine Begründung für diese Entscheidung und Diskussion möglicher Alternativen ist in Abschnitt 4.2.1 zu finden. Die Menge und Struktur der verwendeten Aktivitäten basiert auf den Vorarbeiten von Deißenböck, Wagner und Winter [49, 141, 144] und ergänzt Aktivitäten, die sich aus den in Abschnitt 2.4.3 betrachteten SOA-spezifischen Publikationen ergeben. Qualitätsattribute, deren Bedeutung und Bewertung im Kontext von SOA keine Besonderheiten aufweisen, sind nicht Gegenstand der Betrachtung und damit insbesondere nicht Teil des Qualitätsmodells.

Um Qualitätsziele mittels Aktivitäten zu beschreiben, ist es erforderlich, Eigenschaften zu formulieren, die von diesen Aktivitäten gefordert werden. Im Falle einzelner Qualitätsziele ist dies durch die implizite Forderung nach Kosteneffizienz oder Effektivität möglich (vgl. Abschnitt 2.2.3 auf Seite 8). Eine integrierte Betrachtung unterschiedlicher Qualitätsziele führt allerdings dazu, dass verschiedene Anforderungen in einem einzigen Modell zusammen kommen, so dass eine explizite Angabe sowohl der Aktivität als auch der geforderten Eigenschaft dieser Aktivität unerlässlich ist. Dies wird bereits bei der Betrachtung von *Security* und *Maintainability* sichtbar, wobei Angriffsaktivitäten (im Kontext von Security) möglichst ineffektiv und Wartungsaktivitäten möglichst kosteneffizient durchgeführt werden sollen.

Eine initiale Menge möglicher Aktivitäten kann anhand geeigneter Beschreibungen des Softwarelebenszyklus zusammengestellt und mit typischen Aktivitäten aus der SOA-Literatur ergänzt werden. Anschließend werden die Qualitätsattribute aus vorhandenen Standards und Modellen für Softwarequalität diesen Aktivitäten zugeordnet, indem pro Qualitätsattribut die implizit geforderte Eigenschaft an diese Aktivität abgeleitet wird. Tabelle 4.1 enthält eine beispielhafte Übersicht von Aktivitäten, Eigenschaften sowie den zugeordneten ISO/IEC-ähnlichen Charakteristiken. Sie erhebt keinen Anspruch auf Vollständigkeit, sondern dient zur Veranschaulichung des Zusammenhangs zwischen traditionellen Qualitätsattributen und ihrer aktivitätenbasierten Darstellung. Es wird deutlich, dass die Mehrzahl der beschriebenen Qualitätsattribute eine implizite Forderung nach Effizienz enthält. Insbesondere die Aktivität [Nutzung] vereint allerdings eine große Anzahl unterschiedlicher Qualitätsattribute auf sich, welche jeweils unterschiedliche Anforderungen an die Nutzung der Software formulieren. Analog zu Lochmann und Göb [5] wird daher die Beschreibung der Aktivitäten dergestalt erweitert, dass auch hier Attribute zum Einsatz kommen, z. B. [Nutzung | KOMFORT]. Für die Verwendung im Qualitätsmodell können weitere Aktivitäten ergänzt, bestehende Aktivitäten zu Teilaktivitäten verfeinert und weitere Attribute mit diesen verknüpft werden.

Aktivität	Effizienz	Kontinuität	Sicherheit[4]	Komfort	Effektivität
Analyse	Analyzability				
Test	Testability				
Wartung	Maintainability				
Support	Supportability				
Änderung	Modifyability				
Betrieb	Operability	Reliability			
Auffinden	Discoverability				
Nutzung	Performance	Availability	Safety	Satisfaction	Func. Appropriateness
	Usability				
Anpassung	Adaptability				
	Portability				
Komposition	Reusability				Interoperability
Installation				Installability	

Tabelle 4.1: Aktivitätenbasierte Darstellung traditioneller Qualitätsattribute

Konkret wurden im Rahmen der wissenschaftlichen Vorarbeiten eine Reihe von Aktivitäten identifiziert, denen eine besondere Bedeutung im Kontext von SOA zukommt. Quellen für diese Aktivitäten sind bestehende aktivitätenbasierte Qualitätsmodelle, insbesondere das Wartbarkeitsmodell von Deißenböck u. a. [49], der IEEE-Standard 1074 für den Projektlebenszyklus von Software [74], sowie die Definitionen der Qualitäts-Charakteristiken der ISO/IEC 25010 [77]. Die extrahierten Aktivitäten werden in den folgenden Abschnitten beschrieben und mit Konzepten aus verwandten Arbeiten in Bezug gesetzt, um die Einbettung in den Stand der Wissenschaft und Technik sicherzustellen. Die Gliederung erfolgt dabei entlang der Phasen im SOA-Lebenszyklus, wobei frühe Phasen wie die Anforderungsanalyse explizit nicht betrachtet werden, da hier noch keine Architektur-Artefakte existieren, deren Qualität mit Hilfe des Qualitätsmodells beschrieben werden soll. Das Qualitätsmodell kann während dieser Phasen verwendet werden, um gewisse Systemeigenschaften aus den Qualitätszielen abzuleiten. Die Auswirkungen dieser Eigenschaften treten jedoch zumeist beim Test, der Nutzung und der Weiterentwicklung des Systems auf, weshalb auch nur Aktivitäten aus diesen Phasen benötigt werden, um Einflüsse umfassend beschreiben zu können. Insbesondere wird dabei zwischen Angebot, Ausführung und Wartung unterschieden.

4.4.1 Angebot

Z1: [Auffinden | EFFIZIENZ]

Mit dem Kernkonzept der losen Kopplung von Diensten geht einher, dass diese anstelle einer direkten Verknüpfung in einem Verzeichnis (*Service Repository*) veröffentlicht und von möglichen Konsumenten dort gesucht, gefunden und schließlich verwendet werden. Die Auffindbarkeit von Diensten ist also ein wichtiges Qualitätsziel. In der englischsprachigen Literatur wird dieses üblicherweise als *Discoverability* bezeichnet (vgl. [21, 41, 55, 121]). Choi, Her und Kim definieren Auffindbarkeit wie folgt:

> The capability of the service to be easily, accurately, and suitably found at both design time and runtime for the required service specification. [41]

Für Service-Anbieter besteht der Vorteil einer hohen Auffindbarkeit darin, dass ein größerer Kreis potentieller Nutzer angesprochen und damit ein höherer Gewinn erzielt werden kann. Konsumenten wiederum profitieren von Auffindbarkeit, indem der Aufwand zur Suche passender Dienste reduziert und damit die Effizienz gesteigert werden kann.

Z2: [Externe Analyse | EFFIZIENZ]

Um zu entscheiden, ob ein Service die gewünschte Funktionalität bereitstellt und sich für das geplante Nutzungsszenario eignet, muss er verstanden werden. Dies kann z. B. aufgrund von Bezeichnern und Dokumentation erreicht werden. Je einfacher ein Service verstanden wird, desto größer ist die Wahrscheinlichkeit, dass sich ein potenzieller

Kunde für diesen Service entscheidet, was letztendlich steigende Einnahmen für den Anbieter des Dienstes bedeutet. Das beschriebene Phänomen ist in der ISO/IEC 25010 durch die Subcharakteristik *Appropriateness Recognizability* beschrieben:

> The degree to which users can recognise whether the product is appropriate for their needs. [...] Appropriateness recognisability will depend on the ability to recognise the appropriateness of the functions from initial impressions of the product and/or any associated documentation. [77]

Im Kontext von SOA ist dieses Konzept vor allem deshalb von Bedeutung, weil ein potenzieller Nutzer eines Dienstes aufgrund der Analyse desselben darüber entscheiden muss, ob der Dienst unter den gegebenen Anforderungen geeignet ist. Das Qualitätsziel hierfür ist die effiziente Durchführung der Analyse. Da sich diese Analyse vornehmlich auf Benutzer bezieht, also von außerhalb des beschriebenen Systems durchgeführt wird, wird diese Aktivität als *externe Analyse* bezeichnet: [Externe Analyse | EFFIZIENZ] (im Gegensatz zur Analyse durch Entwickler, welche als *interne Analyse* bezeichnet wird, siehe dazu auch Z8: [Interne Analyse | EFFIZIENZ]).

4.4.2 Ausführung

Bei der Ausführung eines Dienstes sind sowohl Aspekte zu beachten, die für den Nutzer desselben von Bedeutung sind, als auch solche, die vor allem aus Sicht des Anbieters relevant sind. Im Folgenden wird deshalb zwischen der Betreibersicht (*Betrieb*) und der Nutzersicht (*Nutzung*) unterschieden. In Übereinstimmung mit der allgemeinen SOA-Terminologie wird synonym zur Nutzung auch der Begriff *Konsum* gebraucht.

Z3: [Konsum | EFFEKTIVITÄT]

Wann immer ein Service verwendet wird, ist die möglichst präzise Abdeckung der vom Nutzer benötigten Funktionalität ein entscheidendes Qualitätskriterium. Diese Eigenschaft wird in der ISO/IEC 25010 als *Functional Appropriateness* bezeichnet:

> The degree to which the functions are suitable for specified tasks and user objectives. [77]

Ein Service, der die benötigte Funktionalität zur Verfügung stellt, ermöglicht eine effektive Nutzung des Dienstes in Bezug auf die Anforderungen des Nutzers, weshalb dieser Qualitätsaspekt wie folgt dargestellt wird: [Konsum | EFFEKTIVITÄT].

Z4: [Konsum | EFFIZIENZ]

Für den Konsum von Diensten ist Effizienz ein wichtiges Qualitätsziel. Der Begriff Effizienz bezieht sich dabei einerseits auf die möglichst schnelle Beantwortung von Anfragen, andererseits auf die zur Beantwortung dieser Anfragen benötigten Ressourcen wie z. B. Arbeitsspeicher oder Energie. Ersteres ist dabei vor allem für die Nutzer

eines Dienstes von Bedeutung, da die Antwortzeit eine von außen beobachtbare Eigenschaft ist. Letzteres liegt vorrangig im Interesse des Dienstanbieters, für den ein geringer Ressourcenbedarf auch geringere Betriebskosten bedeutet. In der ISO/IEC 25010 wird dies als *Performance Efficiency* bezeichnet:

> The performance relative to the amount of resources used under stated conditions. [77]

Diese Eigenschaft wird im SOA-Qualitätsmodell mit dem Qualitätsziel des effizienten Konsums dargestellt: [Konsum | EFFIZIENZ]. Dieses Qualitätsziel ist eng verwandt mit dem des effizienten Betriebs von Diensten, welches in Z6: [Betrieb | EFFIZIENZ] beschrieben ist.

Z5: [Konsum | KONTINUITÄT]

Zu den Erwartungen an einen Dienst zählt dessen Verfügbarkeit. Nutzer gehen im Allgemeinen davon aus, dass ein Dienst genau dann zur Verfügung steht, wenn er gebraucht wird. In der ISO/IEC 25010 wird dieses Qualitätsziel als *Availability* bezeichnet:

> The degree to which a system or component is operational and accessible when required for use. [77]

Diese Definition führt zwei Aspekte von Verfügbarkeit auf: Der Dienst muss einerseits aktiv sein, so dass er auf Anfragen reagieren kann. Andererseits muss auch der Zugriff durch den Nutzer auf den Dienst gewährleistet sein. Da es für einen Konsumenten im Falle mangelnder Verfügbarkeit oft nicht von Belang ist, welches dieser Kriterien verletzt wurde, fasst das SOA-Qualitätsmodell beide als ein Qualitätsziel zusammen: Z5: [Konsum | KONTINUITÄT].

Z6: [Betrieb | EFFIZIENZ]

Eine weitere wichtige Aktivität, welche stark mit der Nutzung von Diensten zusammenhängt, ist deren Betrieb (vgl. *Operate the System* in IEEE 1074 [74]). Insbesondere da Dienste häufig über das Internet zur Verfügung gestellt und nach verschiedenen Kriterien der Nutzung abgerechnet werden (z. B. Anzahl der Benutzer, Anzahl der Aufrufe etc.), ist ein effizienter Betrieb für den Dienstanbieter von entscheidender Bedeutung. Die entsprechende Charakteristik laut ISO/IEC 25010 lautet *Operability* und wird dort wie folgt definiert:

> The degree to which a product or system has attributes that make it easy to operate and control. [77]

Hier fällt auf, dass im Gegensatz zu anderen Definitionen explizit auf Eigenschaften der Software hingewiesen wird, die einen Einfluss auf das Qualitätsziel besitzen. Die einfache Durchführung dieser Aktivitäten *Operate* und *Control* werden in dieser Definition als gemeinsames Qualitätsziel verstanden. Für das aktivitätenbasierte Modell

erscheint die Formulierung als Forderung nach einem effizienten Betrieb der Software nicht zuletzt im Hinblick auf die Konsistenz der Beschreibung als geeigneter: [Betrieb | EFFIZIENZ].

Z7: [Betrieb | KONTINUITÄT]

Der kontinuierliche Betrieb eines Dienstes bedeutet, dass aus Sicht des Dienstanbieters keine Beeinträchtigung der Funktion des Dienstes vorliegt. Dies ist vor allem in Anbetracht von SLAs von Bedeutung, in denen den Nutzern üblicherweise eine gewisse Verfügbarkeit (z. B. 99,9 %) zugesichert wird. Ist der Dienst über den dort festgelegten Toleranzbereich hinaus nicht verfügbar, werden Strafzahlungen seitens des Betreibers fällig. Die ISO/IEC 25010 beschreibt dazu die Charakteristik *Reliability*:

> The degree to which a system or component performs specified functions under specified conditions for a specified period of time. [77]

Diese sehr unspezifische Beschreibung wird etwas konkreter, wenn man die verfeinernden Charakteristiken *Availability*, *Maturity*, *Fault Tolerance* und *Recoverability* betrachtet. Das letztlich adressierte Qualitätsziel ist jedoch, dass die angebotene Funktionalität möglichst zu allen Zeiten in der spezifizierten Art und Weise zur Verfügung steht: [Betrieb | KONTINUITÄT].

4.4.3 Wartung

Die Wartung von Softwaresystemen ist seit vielen Jahren Gegenstand der Forschung im Software Engineering. Es werden immer wieder Studien zitiert, nach denen der Großteil der über den Lebenszyklus eines Softwaresystems anfallenden Kosten auf die Wartung entfällt [30, 122]. Nicht zuletzt aus diesem Grund existiert eine Vielzahl von Qualitätsmodellen und sonstigen Forschungsarbeiten speziell zur Wartbarkeit von Software (z. B. [40, 92, 104, 127]). Auch die aktivitätenbasierte Qualitätsmodellierung wurde zuerst an einem Wartbarkeitsmodell angewendet [32]. Nach der dort verwendeten Taxonomie lässt sich Wartung in *Analyse*, *Implementierung*, *Test* und *Auslieferung* unterteilen, welche wiederum Teilaktivitäten besitzen. Aufgrund der Betrachtung von Softwarequalität im Allgemeinen werden im Rahmen der vorliegenden Arbeit nicht alle dort verwendeten Aktivitäten unverändert übernommen, sondern in einigen Punkten abgewandelt, wie die folgenden Abschnitte verdeutlichen. Insbesondere stellt die *Komposition* von Diensten eine zentrale Aktivität im Umgang mit SOA dar, weshalb diese explizit in das Modell aufgenommen wird.

Z8: [Interne Analyse | EFFIZIENZ]

Wartungsaktivitäten starten in den meisten Fällen mit einer Analyse – sei es, um aufgetretene Fehler zu finden, oder um geeignete Einstiegspunkte für funktionale Erweiterungen zu identifizieren. In allen Fällen liefert die Analyse des Ist-Zustands Informationen zum weiteren Vorgehen. Diese Einschätzung deckt sich sowohl mit dem

Ansatz des Maintainability-ABQM als auch mit der ISO/IEC 25010, die die Charakteristik *Analysability* folgendermaßen definiert:

> The ease with which the impact of an intended change on the rest of the product can be assessed, or the product can be diagnosed for deficiencies or causes of failures, or the parts to be modified can be identified. [77]

Hier werden implizit drei Arten der Analyse aufgeführt: Wenn das Ziel der Wartungsaktivität eine Fehlerbehebung ist, dient die Analyse zum Lokalisieren der Ursache. In anderen Fällen ist Analyse notwendig, um die Teile des Systems zu identifizieren, an denen eine Änderung vorzunehmen ist (z. B. zum Hinzufügen neuer Funktionalität). Auch die Analyse der Auswirkungen von Änderungen am System ist vorgesehen. Hierzu ist anzumerken, dass die Qualität eines Systems, Auswirkungen von Änderungen möglichst lokal zu halten, der Änderbarkeit zuzuschreiben sind, während die Erkenntnis darüber zur Analyse gehört. Das Qualitätsziel ist dabei stets die effiziente Durchführung der Analyse: [Interne Analyse | EFFIZIENZ]. Für eine Diskussion der Analyse von Diensten durch potentielle Benutzer sei auf Z2: [Externe Analyse | EFFIZIENZ] verwiesen.

Z9: [Anpassung | EFFIZIENZ]

Ein weiterer Qualitätsaspekt von Software ist die Möglichkeit, diese an veränderte Umgebungsbedingungen anzupassen. Dies kann durch Änderungen der Hardware erforderlich werden, aber auch beispielsweise durch ein verändertes Nutzungsverhalten. Die ISO/IEC 25010 beschreibt diesen Qualitätsaspekt unter der Bezeichnung *Adaptability*:

> The degree to which a product or system can effectively and efficiently be adapted for different or evolving hardware, software or other operational or usage environments. [77]

Diese Definition ist bereits unter Verweis auf eine Aktivität und die Forderung nach gewissen Eigenschaften dieser Aktivität formuliert, so dass eine Übertragung in das aktivitätenbasierte Modell unmittelbar gegeben ist: Das Qualitätsziel äußert sich also darin, dass Anpassungen möglichst effektiv und effizient durchgeführt werden können. Analog zu den oben beschriebenen Wartungsaktivitäten wird auch hier vor allem der Aspekt der Effizienz betrachtet und deshalb in das Qualitätsmodell aufgenommen: [Anpassung | EFFIZIENZ].

Z10: [Test | EFFIZIENZ]

Testen ist eine wichtige Aktivität in der Softwareentwicklung (vgl. Aktivitäten *Develop Test Procedures*, *Create Test Data* und *Execute Tests* in IEEE 1074 [74]). Die ISO/IEC 25010 beschreibt Testbarkeit (*Testability*) als eine Sub-Charakteristik von Wartbarkeit:

> The ease with which test criteria can be established for a system or component and tests can be performed to determine whether those criteria have been met. [77]

Insbesondere klingt in dieser Definition die Effizienz der Erstellung und Durchführung von Tests an. Eine gesteigerte Test-Effizienz ermöglicht zudem bei gleichen Ressourcen häufigeres Testen. Dies erhöht die Chance Probleme zu identifizieren und zu beseitigen, was sich wiederum positiv auf die Produktqualität auswirkt. Die Effizienz von Tests wird folglich als Qualitätsziel in das Modell aufgenommen: [Test | EFFIZIENZ].

Z11: [Komposition | EFFIZIENZ]

Komposition von Diensten bezeichnet das Verfahren, eine neue Funktionalität durch die kombinierte Nutzung mehrerer vorhandener Dienste zu erzielen und diese wiederum in Form eines Dienstes zur Verfügung zu stellen. Damit geht dieses Konzept sehr stark mit Wiederverwendung einher. Für die Eigenschaft von Software, sich leicht wiederverwenden zu lassen, existiert in der ISO/IEC 25010 die Charakteristik *Reusability*:

> The degree to which an asset can be used in more than one system, or in building other assets. [77]

Die Wiederverwendbarkeit ist in einem SOA-System dann besonders hoch, wenn sich Dienste effizient komponieren lassen. In der aktivitätenbasierten Darstellung ergibt sich folglich die Effizienz der Komposition als Qualitätsziel: [Komposition | EFFIZIENZ].

Z12: [Komposition | EFFEKTIVITÄT]

SOA ist ein Architekturstil für die Entwicklung verteilter Systeme. Kommunikation zwischen Diensten ist also ein essentieller Bestandteil serviceorientierter Systeme. Die Eigenschaft von Softwareprodukten untereinander Informationen auszutauschen und diese zu nutzen, wird in der ISO/IEC 25010 mit der Charakteristik *Interoperability* bezeichnet:

> The degree to which two or more systems or components can exchange information and use the information that has been exchanged. [77]

Da der Informationsaustausch zwischen Services durch Komposition implementiert wird, bildet das Qualitätsmodell diese Eigenschaft auf das Qualitätsziel der effektiven Komposition von Services ab: [Komposition | EFFEKTIVITÄT].

Erweiterung

Erweiterung bezeichnet im Allgemeinen ein Hinzufügen von Funktionalität. Im Kontext eines SOA-basierten Systems geschieht dies üblicherweise durch Bereitstellung neuer Services, welche in vielen Fällen aufgrund funktionaler Nähe zu bereits vorhandenen Services von diesen Gebrauch machen. In diesem Fall ergibt sich die Erweiterung

des Systems durch Komposition von Services; die Effizienz dieser Erweiterung kann also bereits durch die Effizienz der Komposition (siehe Z11: [Komposition | EFFIZIENZ]) beschrieben werden. Die Erweiterung eines einzelnen Dienstes wird entweder ebenfalls per Komposition erreicht oder direkt im Quellcode auf Ebene der Programmiersprache. Diese Art der Erweiterung ist allerdings nicht Gegenstand der Dissertation und wird deshalb nicht betrachtet.

4.5 Konformität zu den SOA-Prinzipien

Im Folgenden werden Faktoren und Maße beschrieben, mittels derer sich die Umsetzung der in Abschnitt 2.4.1 eingeführten SOA-Prinzipen überprüfen lässt. Jeder dieser Faktoren ist einem SOA-Prinzip zugeordnet, welches wiederum Auswirkungen auf ein oder mehrere aktivitätenbasierte Qualitätsziele (siehe Abschnitt 4.4) besitzt. Diese Einflüsse sind entweder positiv oder negativ und werden jeweils im Anschluss an die Beschreibung eines SOA-Prinzips aufgelistet, basierend auf den Ergebnissen der Literaturarbeit in Abschnitt 2.4.1. Die beschriebenen Faktoren besitzen eine fortlaufende Nummerierung und bestehen aus je einem Attribut und einer Entität, sowie einer Beschreibung. Zur Qualitätsbewertung besitzt jeder Faktor außerdem eine Bewertungsvorschrift, welche die Werte der ihm zugeordneten Maße miteinander in Bezug setzt und auf einen Erfüllungsgrad im Intervall $[0; 1]$ abbildet. Dabei ist zu beachten, dass es sich bei diesen Bewertungsvorschriften um manuell definierte Abbildungen handelt, welche basierend auf persönlicher Erfahrung als sinnvoll eingeschätzt wurden. Vorarbeiten im Projekt Quamoco haben gezeigt, dass es für aussagekräftige Ergebnisse einer Qualitätsbewertung erforderlich ist, das Qualitätsmodell zur Bewertung einer großen Anzahl von Softwaresystemen zu verwenden und auf Grundlage der so gewonnenen Werte die Abbildungsvorschriften im Modell anzupassen. Eine Diskussion dieses Vorgehens ist in Abschnitt 6.2 zu finden. Die genannten Maße bilden beobachtbare Eigenschaften von Entitäten innerhalb eines SOA-Systems auf Messwerte ab, welche innerhalb der Faktoren interpretiert und zu einem Erfüllungsgrad verrechnet werden. In der Beschreibung jedes Maßes wird deshalb der Faktor referenziert, für den das Maß von Bedeutung ist. Maße bezüglich struktureller Eigenschaften von Diensten und deren Beziehungen untereinander werden basierend auf dem in Abschnitt 4.3.2 beschriebenen SOA-Referenzmodell beschrieben. So nutzen etwa die Maße *Fan-In* und *Fan-Out* die dort definierte Verwendungsrelation zwischen Diensten im Sinne der Nutzung einer durch einen anderen Dienst erbrachten Funktionalität. Dieses Beispiel wird in Abbildung 4.10 anhand der für das Referenzmodell verwendeten Darstellung verdeutlicht.

Tabelle 4.2 veranschaulicht die Zusammenhänge der in den folgenden Abschnitten beschriebenen Elemente des SOA-Konformitätsmoduls, ausgehend von den vorgestellten SOA-Prinzipien.

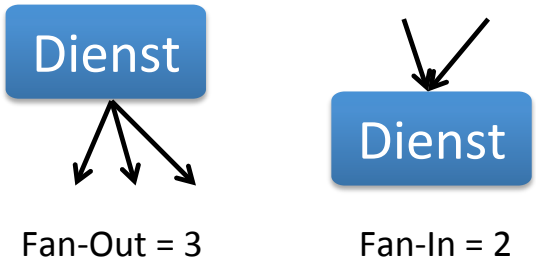

Abbildung 4.10: Maße und das SOA-Referenzmodell

SOA-Prinzip	Beeinflusste Qualitätsziele	Faktoren	Maße
Verschachtelte Wiederverwendung und Komposition	Z3: [Konsum | EFFEKTIVITÄT] Z11: [Komposition | EFFIZIENZ]	F1: [Service | AUFRUFBEZIEHUNGEN] F2: [Service | KONSUMVERHÄLTNIS]	M1: *Fan-Out* M2: *Konsument-Anbieter-Verhältnis*
Verteilung von Funktionalität	Z12: [Komposition | EFFEKTIVITÄT]	F3: [Service | REMOTE-NUTZUNG] F4: [Service | REMOTE-NUTZBARKEIT]	M3: *Anzahl systemexterner Abhängigkeiten* M4: *Anzahl exponierter Dienste*
Geschäftsprozessbezug und Fachlichkeit	Z2: [Externe Analyse | EFFIZIENZ] Z9: [Anpassung | EFFIZIENZ] Z11: [Komposition | EFFIZIENZ]	F5: [Service | ABSTRAKTIONSGRAD]	M5: *Abstraktionsgrad von Diensten*
Standardisierte Formate und Schnittstellen	Z10: [Test | EFFIZIENZ] Z11: [Komposition | EFFIZIENZ] Z12: [Komposition | EFFEKTIVITÄT]	F6: [Schnittstelle | BESCHREIBUNGSSPRACHEN] F7: [Schnittstelle | IMPLEMEN-TIERUNGSNEUTRALITÄT]	M6: *Anzahl konkurrierender Standards* M7: *Beschreibung von Implementierungsdetails*
Lose Kopplung und dynamische Dienstkomposition	Z5: [Konsum | KONTINUITÄT] Z7: [Betrieb | KONTINUITÄT] Z8: [Interne Analyse | EFFIZIENZ] Z9: [Anpassung | EFFIZIENZ]	F8: [Service | ENTKOPPLUNG]	M1: *Fan-Out* M9: *Fan-In* M8: *Direkt miteinander verbundene Services*

Tabelle 4.2: SOA-Konformität – Überblick

4.5.1 Verschachtelte Wiederverwendung und Komposition

Ein zentrales SOA-Prinzip ist die modulare Verwendung von Diensten durch Komposition. Funktionalität, welche in Form von Diensten zur Verfügung steht, wird durch Service-Komposition miteinander kombiniert und um zusätzliche Funktionen erweitert, wobei das Ergebnis wieder als Dienst bereitgestellt wird. Wenn dieses Prinzip umgesetzt wird, bedeutet dies unmittelbar, dass das Qualitätsziel der effizienten Komposition positiv beeinflusst wird. Da es darüber hinaus erleichtert wird, neue Funktionalität zu schaffen, wird auch ein effektiver Konsum (einer gewünschten Funktion) positiv beeinflusst.

$\xrightarrow{+}$ Z11: [Komposition | EFFIZIENZ]
$\xrightarrow{+}$ Z3: [Konsum | EFFEKTIVITÄT]

F1: [Service | AUFRUFBEZIEHUNGEN]

Um den Grad an Komposition innerhalb eines Systems zu bestimmen, müssen die Aufrufbeziehungen zwischen Diensten untersucht werden. Diese Aufrufbeziehungen sind im Rahmen des SOA-Referenzmodells in Abbildung 4.9 auf Seite 71 veranschaulicht. Dienste, welche andere Dienste verwenden, nutzen Komposition. Die direkte Verwendung anderer Dienste wird mit Hilfe von M1: *Fan-Out* bestimmt, was in Abbildung 4.10 dargestellt ist. Dieses Maß wird zur mittleren Anzahl ausgehender Abhängigkeiten pro Service transformiert. Die Abbildung auf den Wertebereich des Faktors geschieht durch Festlegung einer geeigneten oberen Schranke für den Messwert, ab dem der Faktor die maximale Bewertung erhält. Der Initialwert für diese obere Schranke wurde manuell auf Fünf festgelegt.

M1: *Fan-Out*

Der Fan-Out bezeichnet die Anzahl der ausgehenden Abhängigkeiten eines Dienstes, also die Anzahl der Dienste im System, welche von diesem Dienst direkt aufgerufen werden. Diese Definition ist identisch mit Ruds *Absoluter Abhängigkeit eines Service* [123]. Die Anzahl aufgerufener Dienste liefert einen Hinweis darauf, in welchen Maß Komposition im System verwendet wird; im Extremfall ist diese Zahl gleich Null, was auf ein System schließen lässt, in dem keine komponierten Dienste vorhanden sind. Steht die Kompositionslogik für eine statische Analyse zur Verfügung (etwa in Form von BPEL-Artefakten), so kann der Fan-Out automatisch erhoben werden, indem werkzeugbasiert die Information über aufgerufene Dienste aus diesen Beschreibungen extrahiert wird.

Darüber hinaus wird der Fan-Out gemeinsam mit M9: *Fan-In* zur Operationalisierung des Faktors F20: [Service | KRITIKALITÄT] verwendet, welcher nicht unmittelbar zu einem SOA-Prinzip beiträgt und deshalb später in Abschnitt 4.6 auf Seite 108 beschrieben wird.

F2: [Service | KONSUMVERHÄLTNIS]

Komposition wird durch solche Services erleichtert, welche in mehr als einer Art verwendet werden können. Einen Hinweis auf einen hohen Grad an Komposition liefert also das Zahlenverhältnis zwischen Diensten, welche andere Dienste verwenden, und solchen, die innerhalb des Systems von anderen Diensten verwendet werden. Ein hohes Maß an Komposition liegt dann vor, wenn konsumierende Dienste dabei die Mehrheit bilden.

Die Abbildung des Zahlenverhältnisses auf den Faktor-Wertebereich wird auch hier durch Definition eines Schwellwertes realisiert, ab dem der Faktor den Maximalwert annimmt. Dieser Wert wurde initial auf Zwei festgelegt.

M2: *Konsument-Anbieter-Verhältnis*

Um das Verhältnis von Konsumenten und Anbietern zu bestimmen, werden zunächst alle Aufrufbeziehungen zwischen Services innerhalb des Systems untersucht. Nun werden alle Services gezählt, von denen Aufrufe zu anderen Services ausgehen (Konsumenten), sowie solche, bei denen Aufrufe von anderen Services eingehen (Anbieter). Der Quotient dieser beiden Werte ergibt das Konsument-Anbieter-Verhältnis und damit den Wert dieses Maßes.

Unter der Annahme, dass im System mindestens ein Service existiert, welcher von einem anderen Service aufgerufen wird, ist der Wert dieses Maßes immer eine positive reelle Zahl. Sollte ein solcher Service nicht existieren, wird das Ergebnis auf Null festgelegt.

4.5.2 Verteilung von Funktionalität

Die Verteilung von Funktionalität über Systemgrenzen hinweg, oft über das Internet, wurde als weiteres Prinzip von SOA identifiziert. Daraus ergeben sich sowohl Chancen als auch Risiken im Hinblick auf die Qualität eines Systems. Dienstbasierte Anwendungen können über Unternehmensgrenzen hinweg Daten austauschen und Funktionalität einbinden, die von Dritten über das Internet bereitgestellt wird. Daraus ergibt sich grundsätzlich die Anforderung nach besonderer Beachtung der Datensicherheit [9]. Allerdings ist diese nach denselben Kriterien zu bewerten wie in anderen Anwendungen. Das Prinzip der Verteilung von Funktionalität und der damit verbundenen stärkeren Nutzung von Netzwerk-Infrastrukturen hat also nicht unmittelbar eine Auswirkung auf Sicherheitsaspekte, erhöht aber deren Wichtigkeit.

Als weitere Auswirkung dieses Prinzips ergibt sich eine erhöhte Zahl potentieller Nutzer, sofern das aktuelle System selbst nicht nur entfernte Dienste verwendet, sondern auch eigene Dienste für die entfernte Nutzung zur Verfügung stellt. Diese Eigenschaft wirkt sich positiv auf die Wiederverwendung aus, da entfernte Systeme sehr einfach die angebotene Funktionalität nutzen können, anstatt bestimmte Komponenten herunterzuladen, lokal einzurichten und zu betreiben.

$\xrightarrow{+}$ Z12: [Komposition | EFFEKTIVITÄT]

F3: [Service | REMOTE-NUTZUNG]

Die Nutzung entfernter Funktionalität wird in diesem Faktor über die Eigenschaft eines Dienstes beschrieben, Dienste außerhalb des eigenen IT-Systems aufzurufen. Diese Eigenschaft wird durch das Maß M3: *Anzahl systemexterner Abhängigkeiten* quantifiziert. Dabei handelt es sich um eine statisch zu erhebende Größe. Denkbar wäre außerdem eine Analyse der tatsächlich im laufenden System genutzten extern bereitgestellten Funktionalität, etwa in Form des Verhältnisses externer Service-Aufrufe zur Gesamtzahl von Service-Aufrufen. Dies würde allerdings ein bereits aktiv genutztes System voraussetzen und wäre deshalb erst sehr spät im Lebenszyklus eines Systems feststellbar, weshalb sich der Faktor auf statisch zu erhebende Daten fokussiert. Die manuell geschätzte Abbildung auf einen Erfüllungsgrad geschieht dabei wie folgt: Wenn weniger als einer von 100 Services externe Services konsumieren, wird mit Null bewertet, die maximale Erfüllung des Faktors wird ab einem Anteil von 10 % angenommen.

M3: *Anzahl systemexterner Abhängigkeiten*

Die Aufrufe externer Funktionalität (F3: [Service | REMOTE-NUTZUNG]) werden quantifiziert, indem pro Service die Abhängigkeiten zu Diensten außerhalb des lokalen IT-Systems gezählt werden. Diese Kennzahl ist offensichtlich ein Maß für den Grad der systemübergreifenden Nutzung von Services.

F4: [Service | REMOTE-NUTZBARKEIT]

Damit ein Dienst von außerhalb des lokalen Netzwerks genutzt werden kann, müssen sowohl seine Schnittstellenbeschreibung von außen zugänglich als auch eine entsprechende Kommunikation über die Unternehmensgrenze hinweg möglich sein. Diese Eigenschaft wird mittels des Maßes M4: *Anzahl exponierter Dienste* überprüft und pro Dienst quantifiziert. Für jeden Dienst bedeutet der Wert Null, dass kein solcher Zugriff möglich ist; ein Wert von Eins bedeutet, dass der Dienst an der Systemgrenze exponiert ist. Systemweit interpretiert ergibt sich eine durchschnittliche Exponiertheit pro Dienst als Erfüllungsgrad des Faktors.

M4: *Anzahl exponierter Dienste*

Ein Dienst wird als *exponiert* bezeichnet, wenn von außerhalb des Systems auf seine Schnittstellenbeschreibung und seine Funktionalität zugegriffen werden kann. Besitzt ein Dienst diese Eigenschaft, wird ihm der Wert Eins zugewiesen, sonst Null. Als Summe über alle Dienste im System ergibt sich die Anzahl derjenigen Dienste, auf welche von außerhalb der Systemgrenze zugegriffen werden kann. Diese Angabe wird zur Quantifizierung des Faktors F4: [Service | REMOTE-NUTZBARKEIT] herangezogen, da die Zugreifbarkeit von Schnittstelle und Funktionalität eine Voraussetzung für die Nutzung des Dienstes von außen darstellt.

4.5.3 Geschäftsprozessbezug und Fachlichkeit

Eine wichtige Eigenschaft von SOA ist die fachliche Natur von Diensten: Funktionalität wird oft in der Granularität beschrieben, die für die Verwendung in Geschäftsprozessen erforderlich ist; ein Dienst korrespondiert dabei oft mit einem Prozessschritt (siehe dazu auch Abschnitt 2.4.1 und Abschnitt 4.3). Programmierparadigmen wie OO beschreiben Dinge und deren Eigenschaften. Demgegenüber rückt bei Services die Verwendung in den Vordergrund, indem die nach außen sichtbare Schnittstelle das zentrale Element der Beschreibung von Diensten darstellt. Dies versetzt Entwickler, welche diese Dienste verwenden, in die Lage, sich auf die konkreten Anforderungen der Anwendungsdomäne zu konzentrieren.

Neue Anforderungen oder regulative Bestimmungen, die zu Änderungen in Geschäftsprozessen führen, können leichter umgesetzt werden, wenn Services auf gleicher Ebene wie Prozessschritte definiert sind: So können etwa Änderungen einzelner Prozessschritte direkt auf die entsprechenden Services abgebildet werden, und der Austausch eines Prozessschrittes korrespondiert mit dem Austausch des betreffenden Services. Darüber hinaus können Dienste in dieser Abstraktion leichter von Experten der Geschäftsdomäne hinsichtlich ihrer Funktion beurteilt werden. Insgesamt ergibt sich damit ein positiver Einfluss auf die gesamte Wartung eines Dienstes, wie er u. a. auch von Pahl, Zhu und Gacitua-Decar [115] beschrieben wird.

$\xrightarrow{+}$ Z2: [Externe Analyse | EFFIZIENZ]
$\xrightarrow{+}$ Z11: [Komposition | EFFIZIENZ]
$\xrightarrow{+}$ Z9: [Anpassung | EFFIZIENZ]

F5: [Service | ABSTRAKTIONSGRAD]

Wann immer Funktionalität verstanden und weiterentwickelt werden muss, spielt der Abstraktionsgrad der Beschreibung eine wichtige Rolle. Mit Blick auf SOA und die damit verbundene fachliche Sicht auf Dienste lassen sich diese unterteilen in Dienste, die geschäftsprozessbezogene Funktionalität abbilden und solche, die technische Aufgaben erfüllen, z. B. Zugriff auf eine Datenbank bieten, und damit im Allgemeinen unabhängig vom Geschäftsprozess definiert sind. Im SOA-Referenzmodell (siehe Abschnitt 4.3.2) wurden die Abstraktionsebenen *Prozessdienst*, *Hilfsdienst* und *Basisdienst* definiert, welche hier als Kriterium für den Abstraktionsgrad herangezogen werden können. Zur Quantifizierung dieses Faktors wird das Maß M5: *Abstraktionsgrad von Diensten* verwendet.

M5: *Abstraktionsgrad von Diensten*

Um den Faktor F5: [Service | ABSTRAKTIONSGRAD] zu quantifizieren und damit den durchschnittlichen Abstraktionsgrad von Diensten im System zu bestimmen, werden zunächst sämtliche Dienste in die Kategorien Prozessdienst, Hilfsdienst und Basisdienst gemäß dem SOA-Referenzmodell aus Abschnitt 4.3.2 aufgeteilt und nach Tabelle 4.3 mit Bewertungspunkten versehen, so dass ein abstrakterer Dienst eine höhere Bewertung erhält.

Kategorie	Punkte
Prozessdienste	1
Hilfsdienste	0.5
Basisdienste	0

Tabelle 4.3: Punkteschema für den Abstraktionsgrad von Diensten

Dabei sind diejenigen Dienste als Basisdienste zu bezeichnen, die atomare Operationen bereitstellen, z. B. Datenzugriff. Im Gegensatz dazu bilden Prozessdienste Funktionalität ab, welche mit Prozessschritten in einem Geschäftsprozess korrespondieren. Dienste mit einem Abstraktionsniveau dazwischen (also keine atomare, aber durchaus noch prozessunabhängige Funktionalität) werden als Hilfsdienste bezeichnet. Insgesamt ergibt sich für den durchschnittlichen Abstraktionsgrad von Services im System ein Wert zwischen Null und Eins.

4.5.4 Standardisierte Formate und Schnittstellen

Das Prinzip der Standardisierung von Schnittstellenbeschreibungen und Datenformaten besitzt einige offensichtliche Implikationen: Durch standardisierte Schnittstellen wird Interoperabilität zumindest syntaktisch sichergestellt. Darüber hinaus kann an standardisierten Schnittstellen effizienter automatisch getestet werden, da im Idealfall keine Anpassung der Testwerkzeuge nötig ist. Auch die Wiederverwendbarkeit wird positiv beeinflusst, da bereits vorhandene Zugriffsschichten für den entsprechenden Standard verwendet werden können, um eine Funktionalität wiederzuverwenden.

$\xrightarrow{+}$ Z10: [Test | EFFIZIENZ]
$\xrightarrow{+}$ Z12: [Komposition | EFFEKTIVITÄT]
$\xrightarrow{+}$ Z11: [Komposition | EFFIZIENZ]

F6: [Schnittstelle | BESCHREIBUNGSSPRACHEN]

Eine konsistente Nutzung von Standards zur Beschreibung von Schnittstellen liegt dann vor, wenn alle Schnittstellen im System in derselben Beschreibungssprache spezifiziert sind. Dies schließt nicht aus, dass Dienste existieren, die ihre Funktionalität darüber hinaus in anderen Beschreibungssprachen zur Verfügung stellen, sondern soll lediglich eine Fragmentierung des Systems in Bereiche vermeiden, die ausschließlich unterschiedliche Standards verwenden.

Aus diesem Grund wird das Maß M6: *Anzahl konkurrierender Standards* dahingehend interpretiert, dass nur dann die höchste Bewertung erzielt werden kann, wenn sämtliche Schnittstellen im System nach demselben Standard beschrieben werden: $F6(s) = 1 - M6(s)$.

M6: *Anzahl konkurrierender Standards*

Dieses Maß wird verwendet, um F6: [Schnittstelle | BESCHREIBUNGSSPRACHEN] zu quantifizieren. Hierzu muss zunächst bestimmt werden, welcher Schnittstellenstandard (z. B. WSDL) im untersuchten System vorherrscht. Dies sollte durch einen Experten unmittelbar zu beantworten sein. In einem zweiten Schritt werden alle Dienste im System daraufhin untersucht, ob sie eine Schnittstellenbeschreibung anbieten, die diesem Standard genügt. Offensichtlich ist hier eine automatisierte Suche (etwa nach WSDL-Dokumenten) nur bedingt hilfreich, da diese zwar die konformen Beschreibungen findet, nicht aber solche, die einem anderen Standard folgen oder im ungünstigsten Fall gar nicht existieren. Ausgangspunkt der Analyse muss folglich eine Beschreibung des Soll-Zustands sein, also etwa eine Spezifikation, in der alle Dienste aufgeführt sind. Nun kann basierend auf einem Soll-Ist-Vergleich der Anteil der Dienste bestimmt werden, für die keine Schnittstellenbeschreibung nach dem vorherrschenden Standard verfügbar ist. Daraus ergibt sich ein Wertebereich zwischen Null und Eins.

F7: [Schnittstelle | IMPLEMENTIERUNGSNEUTRALITÄT]

Eine standardisierte Beschreibung der Schnittstellen ist nicht nur aus Gründen der Einheitlichkeit ein zentrales SOA-Prinzip, sondern auch um die Schnittstelle wirksam von der Implementierung des Dienstes zu entkoppeln. Dies ist allerdings nur dann möglich, wenn der gewählte Beschreibungsstandard keine Konzepte verwendet, die eine bestimmte Programmiersprache oder technische Plattform voraussetzen. Nur so kann gewährleistet werden, dass z. B. ein bestehender Service durch einen anderen ausgetauscht werden oder mit einem solchen kommunizieren kann, der mit Hilfe einer anderen Technologie implementiert wurde. Offensichtlich trägt diese Eigenschaft damit sowohl zur Interoperabilität als auch zur Möglichkeit der schrittweisen Modernisierung eines Systems bei. Das entsprechende Maß M7: *Beschreibung von Implementierungsdetails* wird bei einem Wert von Null auf den maximalen Erfüllungsgrad abgebildet. Der obere Grenzwert, bei dem eine Bewertung des Faktors mit Null erfolgt, sollte niedrig sein. Für das Qualitätsmodell wurde festgelegt, dass der Faktor bereits bei einer solchen Abhängigkeit mit Null bewertet wird, also nur dann eine von Null verschiedene Bewertung erhält, wenn keine Abhängigkeiten zu Implementierungsdetails in den Schnittstellen vorhanden sind:

$$\mathrm{F7} = \begin{cases} 1 & \text{falls } \mathrm{M7} = 0 \\ 0 & \text{sonst} \end{cases}$$

M7: *Beschreibung von Implementierungsdetails*

Schnittstellen von Diensten sollten wie im vorangegangenen Abschnitt erwähnt von Details der technischen Implementierung abstrahieren. Diese Eigenschaft wurde im Faktor F7: [Schnittstelle | IMPLEMENTIERUNGSNEUTRALITÄT] beschrieben und wird wie folgt quantifiziert: Die in M6: *Anzahl konkurrierender Standards* bestimmte vorherrschende Beschreibungssprache für Service-Schnittstellen wird daraufhin untersucht, ob sie Refe-

renzen auf Spezifika bestimmter Implementierungsarten (z. B. einer bestimmten Programmiersprache) enthält. Dies kann bei angepassten Beschreibungssprachen der Fall sein, etwa durch Definition eines spezifischen Typsystems. Werden lediglich verbreitete Beschreibungssprachen wie WSDL und XSD-Datentypen verwendet, ist dieses Risiko gering – bei firmenspezifischen Anpassungen und Erweiterungen ist jedoch eine gründliche Analyse durch Technologie-Experten zu empfehlen. Der Messwert ergibt sich aus der Summe der identifizierten Abhängigkeiten von Implementierungsdetails in den untersuchten Schnittstellen.

4.5.5 Lose Kopplung und dynamische Dienstkomposition

Dynamisches Auffinden und Ausführen von Diensten sind zwei zentrale Konzepte von SOA. Sie sind die Grundlage für Dienstkomposition während der Ausführung, um bei Bedarf dynamisch neue Dienste zu erzeugen. Oft wird diese lose Kopplung jedoch technisch nicht oder nur unzureichend umgesetzt (vgl. Michlmayr u. a. [103]). Daraus ergibt sich die Notwendigkeit, die Umsetzung dieses Prinzips zu überprüfen.

Die Verwendung dynamischer Dienstkomposition führt dazu, dass bei Nichtverfügbarkeit eines aufzurufenden Dienstes stattdessen ein anderer verwendet werden kann. Im Idealfall kann dynamische Dienstkomposition auch dazu verwendet werden, die Qualität von Diensten zu optimieren, indem aus einer Menge in Frage kommender Dienste für eine bestimmte Funktionalität dynamisch immer derjenige ausgeführt wird, der aktuell bezüglich nicht-funktionaler Eigenschaften die beste Bewertung erreicht (z. B. das schnellste Antwortverhalten oder den geringsten Preis). Darüber hinaus erleichtert lose Kopplung die Anpassung von Diensten an neue Anforderungen oder Umgebungsbedingungen. Sind allerdings Abhängigkeiten zwischen Diensten nicht statisch definiert, fällt die Analyse schwer, an welchen Stellen im System ein Dienst verwendet wird und welche Auswirkungen eine Änderung an diesem mit sich bringt.

$\xrightarrow{+}$ Z5: [Konsum | KONTINUITÄT]
$\xrightarrow{+}$ Z7: [Betrieb | KONTINUITÄT]
$\xrightarrow{+}$ Z9: [Anpassung | EFFIZIENZ]
$\xrightarrow{-}$ Z8: [Interne Analyse | EFFIZIENZ]

F8: [Service | ENTKOPPLUNG]

Kopplung ist ein bekanntes Phänomen in der der Objektorientierung, wo sie die Abhängigkeit zwischen Klassen beschreibt. Shim u. a. [128] übertragen dieses Konzept auf SOA und definieren Kopplung als die Abhängigkeit zwischen Diensten in einem System, welche mittels M8: *Direkt miteinander verbundene Services* bestimmt wird. Es wird also der Grad an direkt voneinander abhängigen Diensten im System bestimmt und als Kenngröße für die Kopplung des Systems herangezogen. Dieses Vorgehen ist durchaus legitim, da etwa Michlmayr u. a. [103] gezeigt haben, dass direkte Abhängigkeiten zwischen Diensten häufig anzutreffen sind. Dabei wird ein Service als *entkoppelt* bezeichnet und mit dem Erfüllungsgrad Eins bewertet, wenn er keine statisch definierten Abhängigkeiten zu anderen Diensten im System besitzt ($M8 = 0$).

M8: *Direkt miteinander verbundene Services*

Wie stark Dienste im System miteinander gekoppelt sind, lässt sich aus der Analyse der systeminternen Kompositionslogik ermitteln. Dazu wird für jeden Service bestimmt, wie viele Services er selbst benutzt (vgl. M1: *Fan-Out*), und von wie vielen Services innerhalb des Systems er direkt verwendet wird (vgl. M9: *Fan-In*). Die Summe dieser beiden Angaben, jeweils normiert über die Gesamtanzahl an Diensten im System, ergibt den Messwert:

$$\mathrm{M8} = \mathrm{M1} + \mathrm{M9}$$

Damit steht eine Quantifizierung für F8: [Service | ENTKOPPLUNG] zur Verfügung, welche automatisch durch Analyse der Kompositionslogik gewonnen werden kann.

M9: *Fan-In*

Mit dem Fan-In eines Dienstes wird die Anzahl der übrigen Dienste im System bezeichnet, welche diesen Dienst aufrufen, wobei nur direkte Aufrufe berücksichtigt werden. Im Gegensatz zu Ruds *Absoluter Wichtigkeit eines Service* [123] werden dabei nicht nur diejenigen Dienste gezählt, die sich auf anderen Knoten im Netzwerk befinden. Dies ist der Tatsache geschuldet, dass das Maß nicht wie dort nur zur Bestimmung von Zuverlässigkeit und Netzwerklast verwendet wird, sondern in einem allgemeinen Kontext, so dass die dort getroffenen Annahmen hier nicht gelten. Darüber hinaus legt die terminologische Ähnlichkeit zu M1: *Fan-Out* eine symmetrische Definition nahe, welche in [123] zum entsprechenden Maß *Absolute Abhängigkeit eines Service* nicht gegeben ist. Neben seinem Beitrag zur Berechnung von M8: *Direkt miteinander verbundene Services* wird der Fan-In später zur Bestimmung von F20: [Service | KRITIKALITÄT] herangezogen.

4.5.6 Weitere Prinzipien aus der Literatur

Die in den vorangegangenen Abschnitten beschriebenen SOA-Prinzipien mögen selektiv oder unvollständig erscheinen. Wie bereits in Abschnitt 2.4.1 diskutiert, variieren Anzahl und Art der genannten SOA-Prinzipien je nach Autor und Publikation. Dabei werden gelegentlich verschiedenartige Konzepte vermischt und unter der gemeinsamen Bezeichnung *SOA-Prinzipen* zusammengefasst. Die vorhergehenden Abschnitte betrachten explizit nur jene dieser Prinzipien, die Merkmale der Architektur darstellen. Weitere Konzepte, welche in einigen Arbeiten ebenfalls als SOA-Prinzipien bezeichnet werden, sind im Folgenden beschrieben. Dabei wird jeweils begründet, weshalb diese Konzepte nicht Teil des SOA-Konformitätsmoduls sind. Sofern Konzepte aus der folgenden Aufzählung im SOA-Designmodul (Abschnitt 4.6) adressiert werden, wird hier auf die entsprechenden Faktoren verwiesen.

Automatisierung von Komposition und Test

Automatisierung ist einerseits eine positive Folge von Standardisierung und Komposition (vgl. Abschnitt 2.4), andererseits aber auch selbst ein wichtiges Prinzip serviceorientierter Architekturen. Der automatisierte Konsum von Diensten im Rahmen einer

Komposition ist tief im serviceorientierten Paradigma verankert. Zwar ließe sich argumentieren, dass sich aus technischer Sicht in einem gegebenen System der Grad an Automatisierung bestimmen ließe – etwa durch Ermittlung der Anzahl automatischer Testfälle pro Service, allerdings wäre ein solches Maß kein verlässlicherer Indikator als die bloße Existenz standardisierter Schnittstellen. Aus diesem Grund wurde darauf verzichtet, Automatisierung als explizites Modell-Element aufzunehmen. Bezüglich der Auswirkungen von Automatisierung auf Qualität werden in der Literatur vor allem Anforderungen an den Qualitätssicherungsprozess abgeleitet: So müssen bei automatischer Komposition von Diensten auch Qualitätsanforderungen automatisch überprüfbar gemacht werden. Einen Beitrag dazu leistet diese Arbeit, indem sie bestimmte Qualitätsaspekte von Services operationalisiert und z. T. automatisch bewertbar macht. Eine unmittelbare Auswirkung von Automatisierung auf die Produktqualität ist jedoch nicht gegeben.

Implementierungsneutralität

Implementierungsneutralität von Dienst-Schnittstellen bedeutet, dass diese unabhängig von der Implementierung der Dienste beschrieben werden. Im Sinne von *Information Hiding* (vgl. Vogel u. a. [138]) werden damit Details zur Implementierung eines Dienstes, die nicht für dessen Verwendung von Bedeutung sind, vor dem Aufrufer verborgen. Darüber hinaus soll die Schnittstellenbeschreibung möglichst keine Elemente enthalten, die eine Realisierung des Dienstes in einer speziellen Programmiersprache oder mit einer speziellen Technologie erfordern. Erl [54] bezeichnet die Eigenschaft der Entkopplung von Schnittstelle und Implementierung als *Service Loose Coupling* und positioniert sie als eigenständiges Prinzip von SOA. Oft wird diese Anforderung allerdings in direktem Zusammenhang mit dem Prinzip der Standardisierung der Schnittstellenbeschreibung erwähnt (vgl. Hündling [73]). Interpretiert man dieses Prinzip dergestalt, dass Schnittstellen nicht nur eine allgemein standardisierte Beschreibung besitzen, sondern einem Beschreibungsstandard folgen sollen, welcher die Unabhängigkeit von der konkreten Realisierung der zugrundeliegenden Dienste garantiert, so ist Implementierungsneutralität als Teil dieses Prinzips zu verstehen. Der Faktor F7: [Schnittstelle | IMPLEMENTIERUNGSNEUTRALITÄT] wurde dieser Argumentation folgend beschrieben.

Zustandslosigkeit

Unter *Zustandslosigkeit* von Diensten wird gemeinhin die Eigenschaft verstanden, dass ein Dienst nach einem Aufruf weiterhin Kontextinformationen im Arbeitsspeicher vorhalten muss, welche für folgende Aufrufe desselben Dienstes durch denselben Konsumenten von Bedeutung sind. Selbstverständlich benötigt ein Dienst im Allgemeinen persistente Informationen – etwa in Form von persönlichen Einstellungen, Dokumenten oder Daten eines Benutzers. Diese werden allerdings für die Diskussion, ob ein Dienst zustandsbehaftet oder zustandslos ist, nicht in Betracht gezogen. Erl beschreibt den Begriff *Zustand* wie folgt:

> Jeder Zustand kann von Daten dargestellt und beschrieben werden, deren Lebensdauer normalerweise der Zeitspanne entspricht, die das Programm für eine gegebene Aufgabe oder einen gegebenen Zweck aktiv bleibt. Daher sind alle Varianten der Zustandsinformation von Natur aus vergänglich. Somit kann Zustandsverwaltung auch als Verwaltung von temporären, aktivitätsspezifischen Daten betrachtet werden. [55]

Zustandslosigkeit beschreibt also die Eigenschaft eines Dienstes, zwischen Operationsaufrufen keine temporären Daten im Arbeitsspeicher vorzuhalten. Aufeinanderfolgende Operationsaufrufe eines zustandslosen Dienstes können also von unterschieldlichen Netzwerkknoten bearbeitet werden, sofern diese auf denselben Persistenzdaten operieren. Alle Daten, welche für die Ausführung Operation eines zustandslsen Dienstes erforderlich sind, werden also zwischen Operationsaufrufen persistiert oder vom Konsumenten als Parameter an die Operation übergeben.

Ungeachtet der Tatsache, dass man hier lediglich zwischen der Speicherung von Daten im Arbeitsspeicher und auf einem nicht-flüchtigen Datenträger unterscheidet, wird Zustandslosigkeit von Diensten häufig diskutiert und von manchem Autoren (u. a. Hündling [73]) als ein festes Prinzip von SOA betrachtet. Da in der Literatur allerdings in diesem Punkt bisher kein Konsens erkennbar ist, wird hier darauf verzichtet, dieser Eigenschaft eine solch zentrale Rolle zuzusprechen. Zweifellos bringt die Unabhängigkeit von der Verwaltung des Anwendungszustands gewisse Vorteile mit sich, weshalb der Faktor F21: [Service | ZUSTANDSLOSIGKEIT] als Teil des Design-Moduls im folgenden Abschnitt näher betrachtet wird; lediglich eine Formulierung als SOA-Prinzip bleibt zu diesem Zeitpunkt aus.

Flexible Konfigurierbarkeit

Dass SOA-Systeme flexibel konfigurierbar sind, ist nicht als Merkmal der Architektur ausgeprägt, sondern wird als Folge der Erfüllung anderer Prinzipien erkennbar (z. B. der losen Kopplung). Aus diesem Grund wird diese Eigenschaft dem Bereich der Qualitätsziele zugeordnet und spiegelt sich im Qualitätsmodell als Z9: [Anpassung | EFFIZIENZ] wider. Inwieweit SOA als Architekturstil tatsächlich flexible Konfigurierbarkeit und Anpassbarkeit gewährleistet, wird in Abschnitt 5.3.1 mit Hilfe des Qualitätsmodells diskutiert. Insgesamt wird Flexibilität also nicht als inhärentes Prinzip, sondern als zu erreichendes Qualitätsziel verstanden.

4.6 Qualität im SOA-Design

Wenn sichergestellt ist, dass ein serviceorientiertes IT-System die SOA-Prinzipien korrekt umsetzt (vgl. Abschnitt 4.5), können weitere Design-Faktoren berücksichtigt werden, welche die Produktqualität beeinflussen. Solche Faktoren werden in diesem Abschnitt betrachtet und analog zum vorigen Abschnitt beschrieben. Ein wesentlicher Unterschied dabei ist jedoch, dass Design-Faktoren selbst Einflüsse auf Qualitätsziele besitzen, wohingegen diese bei Konformitätsfaktoren mittelbar über die SOA-Prinzipien ausgedrückt wurden.

Die hier vorgestellten Designfaktoren setzen eine konforme SOA voraus, so dass das Konformitätsmodul zwar alleine verwendet werden kann, das Designmodul aber nur zusammen mit dem Konformitätsmodul (vgl. schematische Darstellung in Abbildung 4.2 auf Seite 61). Die hier vorgestellten Faktoren und Maße stammen einerseits aus den in Abschnitt 2.4.3 betrachteten Qualitätsmodellen für SOA, andererseits aus den Ergebnissen der Literaturanalyse, die in Kapitel 3 beschrieben wurde. Letztere wurden für die Verwendung im SOA-Qualitätsmodell angepasst, da sich die Literaturanalyse über unterschiedliche Programmierparadigmen erstreckte und dort identifizierte Faktoren für Softwarequalität im Allgemeinen nicht in identischer Form auf SOA anwendbar waren.

Analog zum vorigen Abschnitt ist zu beachten, dass die hier angegebenen Abbildungen von Messwerten auf Erfüllungsgrade von Faktoren basierend auf persönlicher Erfahrung manuell definiert wurden. Für aussagekräftige Ergebnisse einer Qualitätsbewertung ist es erforderlich, das Qualitätsmodell zur Bewertung einer großen Anzahl von Softwaresystemen zu verwenden und auf Grundlage der so gewonnenen Werte die Abbildungsvorschriften im Modell anzupassen. Eine Diskussion dieses Vorgehens ist in Abschnitt 6.2 zu finden. Tabelle 4.4 veranschaulicht die Zusammenhänge der in den folgenden Abschnitten beschriebenen Elemente des Qualitätsmodells.

Faktor	Einfluss	Maß
F9: [Schnittstelle | BEDEUTSAME BESCHREIBUNG]	Z2: [Externe Analyse | EFFIZIENZ] Z8: [Interne Analyse | EFFIZIENZ] Z11: [Komposition | EFFIZIENZ]	M10: *Bedeutsame Beschreibung*
F10: [Service | ANGEMESSENE BENENNUNG]	Z1: [Auffinden | EFFIZIENZ] Z2: [Externe Analyse | EFFIZIENZ] Z8: [Interne Analyse | EFFIZIENZ]	M11: *Angemessenheit des Dienstnamens*
F11: [Operation | ANGEMESSENE BEZEICHNER]	Z2: [Externe Analyse | EFFIZIENZ] Z8: [Interne Analyse | EFFIZIENZ]	M12: *Angemessenheit des Operationsnamens* M13: *Bedeutsame Parameter-Namen*
F12: [System | DESIGN-GRÖßE]	Z8: [Interne Analyse | EFFIZIENZ]	M14: *Anzahl der Dienste im System*
F13: [System | FRAGMENTIERUNG]	Z8: [Interne Analyse | EFFIZIENZ] Z6: [Betrieb | EFFIZIENZ]	M15: *Rechner-externe Abhängigkeiten*
F14: [System | SCHNITTSTELLENKOMPLEXITÄT]	Z8: [Interne Analyse | EFFIZIENZ] Z11: [Komposition | EFFIZIENZ]	M16: *Operationen an der Systemgrenze* M17: *Semantisch äquivalente Dienste*
F15: [Schnittstelle | KOMPLEXITÄT]	Z2: [Externe Analyse | EFFIZIENZ] Z8: [Interne Analyse | EFFIZIENZ] Z11: [Komposition | EFFIZIENZ]	M18: *Anzahl Operationen* M19: *Operationskomplexität*
F16: [Service | FUNKTIONALE GRANULARITÄT]	Z2: [Externe Analyse | EFFIZIENZ] Z11: [Komposition | EFFIZIENZ] Z9: [Anpassung | EFFIZIENZ]	M20: *Berührte und voll abgedeckte funktionale Domänen* M21: *Gegenseitige Dienstabhängigkeiten*
F17: [Operation | FUNKTIONALE GRANULARITÄT]	Z11: [Komposition | EFFIZIENZ]	M22: *Anzahl ausgeführter Funktionen*
F18: [Service | DATENGRANULARITÄT]	Z2: [Externe Analyse | EFFIZIENZ] Z3: [Konsum | EFFEKTIVITÄT] Z11: [Komposition | EFFIZIENZ] Z9: [Anpassung | EFFIZIENZ]	M23: *Datengranularität*

F19: [Schnittstelle | KOHÄSION]	Z2: [Externe Analyse | EFFIZIENZ] Z8: [Interne Analyse | EFFIZIENZ] Z11: [Komposition | EFFIZIENZ] Z3: [Konsum | EFFEKTIVITÄT]	M24: *Gemeinsame Datentypen für Parameter* M25: *Nutzung von Operationen durch Dienstkonsumenten* M26: *Abfolge-Zusammenhang zwischen Operationen*
F20: [Service | KRITIKALITÄT]	Z7: [Betrieb | KONTINUITÄT] Z5: [Konsum | KONTINUITÄT]	M28: *Minimale Dienst-Zuverlässigkeit* M27: *Abhängigkeitsprodukt* M9: *Fan-In* M1: *Fan-Out*
F21: [Service | ZUSTANDSLOSIGKEIT]	Z6: [Betrieb | EFFIZIENZ]	M29: *Anzahl kontextabhängiger Operationen*
F22: [Service | ABHÄNGIGKEITSPFADE]	Z4: [Konsum | EFFIZIENZ] Z5: [Konsum | KONTINUITÄT]	M31: *Abhängigkeitstiefe* M30: *Abhängigkeitssumme*

Tabelle 4.4: SOA-Design – Überblick

4.6.1 Dienstbeschreibung und Dokumentation

Im Folgenden werden Faktoren beschrieben, welche die Beschreibung und Dokumentation von Diensten betreffen. Dazu gehören u. a. Bezeichner für Dienste und deren Operationen. Darüber hinaus werden aber auch sonstige Formen menschenlesbarer Dokumentation von Schnittstellen betrachtet, welche dem Benutzer die Funktion eines Dienstes beschreiben, etwa in Form von Benutzerhandbüchern oder Web-Seiten. Die Auswirkungen dieser Faktoren stammen allesamt aus dem Umfeld der Verständlichkeit, beschreiben also insbesondere die Effizienz der Analyse sowie der Komposition von Diensten.

F9: [Schnittstelle | BEDEUTSAME BESCHREIBUNG]

Chen und Huang [40] analysieren häufig auftretende Problemfaktoren in der Softwareentwicklung hinsichtlich ihrer Auswirkungen auf die Wartbarkeit von Softwaresystemen. Ein Faktor, der in dieser Analyse als wichtig für die Wiederverwendbarkeit identifiziert wurde, ist die Klarheit und Zuverlässigkeit der mitgelieferten Dokumentation. Bei der Dokumentation spielt die Beschreibung der Schnittstellen eine wichtige Rolle, sowohl technischer Art (z. B. mittels WSDL) als auch in menschenlesbarer Form. Die Sinnhaftigkeit und Verständlichkeit einer Dokumentation kann nur durch Experten beurteilt werden, was im Modell durch das manuell zu erhebende Maß M10: *Bedeutsame Beschreibung* umgesetzt wird. Der Wertebereich dieses Maßes genügt bereits den Anforderungen einer Faktorbewertung, so dass der Wert von M10 direkt übernommen werden kann.

Ein weiterer Aspekt der Schnittstellenbeschreibung ist die Sinnhaftigkeit von Bezeichnern für Dienste sowie Operationen (inklusive deren Parameter). Da sich in diesen Fällen sowohl die Entitäten als auch die Einflüsse unterscheiden, werden diese Eigenschaften im Modell getrennt betrachtet und mittels eigener Faktoren abgebildet: F10: [Service | ANGEMESSENE BENENNUNG], F11: [Operation | ANGEMESSENE BEZEICHNER].

Da der Schnittstellenbeschreibung eine hohe Bedeutung in Bezug auf Verständlichkeit sowie Wiederverwendung nachgewiesen wurde, werden hier Einflüsse auf die Effizienz von Analyse und Komposition modelliert.

- $\xrightarrow{+}$ Z2: [Externe Analyse | EFFIZIENZ]
- $\xrightarrow{+}$ Z8: [Interne Analyse | EFFIZIENZ]
- $\xrightarrow{+}$ Z11: [Komposition | EFFIZIENZ]

M10: *Bedeutsame Beschreibung*

Dokumentation ist für jede Art von Software von Bedeutung, was in verschiedenen Arbeiten gezeigt wurde [18, 59, 135]. Im Fall von SOA sollten also insbesondere Schnittstellen angemessen und bedeutsam dokumentiert sein. Dies geschieht im Allgemeinen über entsprechende textuelle Beschreibungen als Metadaten im *Service Repository*, was mit dem Faktor F9: [Schnittstelle | BEDEUTSAME BESCHREIBUNG] ausgedrückt wird. Da die Sinnhaftigkeit von Dokumentation ausschließlich manuell bewertet werden kann, ist dieses Maß von Experten zu erheben. Dabei bewertet der Experte die Beschreibung

von Schnittstellen im System auf einer Skala von Null (*nicht vorhanden oder komplett unverständlich*) bis Eins (*vollständig und nachvollziehbar*).

F10: [Service | ANGEMESSENE BENENNUNG]

Der Name eines Dienstes ist ein wichtiger Bezeichner, anhand dessen ein Benutzer erste Rückschlüsse auf die angebotene Funktionalität zieht. Der Erfüllungsgrad dieses Faktors wird mittels des manuellen Maßes M11: *Angemessenheit des Dienstnamens* bestimmt. Dieses besitzt bereits den für die Faktorbewertung erforderlichen Wertebereich von $[0; 1]$ und kann deshalb direkt verwendet werden.

Wenn der Name eines Dienstes seine Funktion nicht oder nicht ausreichend klar erkennen lässt, ist es für den Benutzer schwierig, den Dienst zu finden. Außerdem hilft ein sinnvoll gewählter Name bei der Überlegung, ob der Service unter gegebenen Anforderungen von Nutzen ist. Daraus ergeben sich die folgenden Einflüsse:

- $\xrightarrow{+}$ Z1: [Auffinden | EFFIZIENZ]
- $\xrightarrow{+}$ Z2: [Externe Analyse | EFFIZIENZ]
- $\xrightarrow{+}$ Z8: [Interne Analyse | EFFIZIENZ]

M11: *Angemessenheit des Dienstnamens*

Es ist nicht möglich, automatisiert zu ermitteln, zu welchem Grad Dienste innerhalb eines Systems angemessen benannt sind, wie in F10: [Service | ANGEMESSENE BENENNUNG] beschrieben wurde. Aus diesem Grund wird eine manuelle Bewertung vorgeschlagen, in der ein Experte einschätzt, inwieweit die Funktionalität jedes Dienstes durch seinen Namen angemessen beschrieben ist. Dazu muss er sowohl über das technische Verständnis verfügen, die Funktionalität aufgrund der Implementierung (sofern zur Ansicht verfügbar) oder des an der Schnittstelle beobachtbaren Verhaltens zu erkennen, als auch über ausreichendes Domänenwissen, um einschätzen zu können, inwieweit der Bezeichner aus fachlicher Sicht diese Funktionalität beschreibt. Diese Notwendigkeit ergibt sich aus der Tatsache, dass auf der Ebene von Services häufig die Sprache der Anwendungsdomäne verwendet wird (vgl. Abschnitt 2.4.1).

F11: [Operation | ANGEMESSENE BEZEICHNER]

Ähnlich wie Dienste sollten auch Operationen angemessen benannt sein. Die Definition von Shim u. a. [128] unterscheidet auf der Faktor-Ebene nicht zwischen Bezeichnern von Operationen und Diensten, sondern fasst beide Konzepte zu einer Design-Eigenschaft zusammen (siehe dazu F10: [Service | ANGEMESSENE BENENNUNG]). Aufgrund der detaillierteren und deshalb leicht unterschiedlichen Beschreibung der Auswirkungen beider Faktoren, werden sie hier allerdings getrennt betrachtet. Inwieweit dieser Faktor erfüllt ist, wird mittels der manuellen Maße M12: *Angemessenheit des Operationsnamens* und M13: *Bedeutsame Parameter-Namen* bestimmt.

Wenn der Name einer Operation ihre Funktion nicht oder nicht ausreichend klar erkennen lässt, ist es für den Benutzer schwierig, die Operation zu verstehen und zu entscheiden, ob diese Operation für gegebene funktionale Anforderungen geeignet ist.

$\xrightarrow{+}$ Z2: [Externe Analyse | EFFIZIENZ]
$\xrightarrow{+}$ Z8: [Interne Analyse | EFFIZIENZ]

M12: *Angemessenheit des Operationsnamens*

Analog zu Diensten im Allgemeinen ist auch bei der Bewertung der Angemessenheit von Operationsnamen (vgl. F11: [Operation | ANGEMESSENE BEZEICHNER]) eine automatische Erhebung nicht möglich, so dass eine manuelle Erhebung erforderlich ist. Die Anforderungen an den Experten, der die Bewertung durchführt, sind identisch zu den in M11: *Angemessenheit des Dienstnamens* beschriebenen. Der Experte bewertet auf einer Skala zwischen Null und Eins, inwieweit die Namen von Operationen ihre Funktion korrekt beschreiben.

M13: *Bedeutsame Parameter-Namen*

Auch die Bewertung der Angemessenheit von Parameter-Bezeichnern ist nicht automatisch möglich, so dass eine manuelle Erhebung erforderlich ist. Die Anforderungen an den Experten, der die Bewertung durchführt, sind identisch zu den in M11: *Angemessenheit des Dienstnamens* und M12: *Angemessenheit des Operationsnamens* beschriebenen. Der Experte bewertet auf einer Skala zwischen Null und Eins, inwieweit die Namen von Parametern ihre Bedeutung korrekt beschreiben.

4.6.2 Größe und Komplexität

Die folgenden Faktoren beschreiben Eigenschaften von Systemen und Diensten, die deren Größe oder Komplexität betrachten. Dazu gehören sowohl die Anzahl von Diensten und Operationen, als auch komplexere Konstrukte, welche etwa neben der Anzahl von Parametern einer Operation auch deren Struktur einbeziehen. Ähnlich wie die Faktoren zur Dokumentation im vorigen Abschnitt sind vor allem Auswirkungen auf Qualitätsziele zu finden, die mit der Verständlichkeit in Zusammenhang stehen, insbesondere die Effizienz der Analyse und der Komposition. Zusätzlich bestehen aber auch Auswirkungen auf die Effizienz des Betriebs eines Systems.

F12: [System | DESIGN-GRÖßE]

Die Größe von Systemen wurde bereits vielfach als Indikator für ihre Verständlichkeit herangezogen (entsprechende Literaturanalysen sind beispielsweise in [22] oder [142] zu finden). Je größer ein System ist, desto umfangreicher ist eine Analyse dieses Systems, so dass diese mehr Aufwand erfordert. Insbesondere ist zu beachten, dass es sich um einen *negativen* Einfluss handelt, also ein System tendenziell ineffizienter zu analysieren wird, je größer es ist.

Die Größe eines serviceorientierten Systems wird analog zur Anzahl der Klassen in OO-Systemen mittels M14: *Anzahl der Dienste im System* quantifiziert.

$\xrightarrow{-}$ Z8: [Interne Analyse | EFFIZIENZ]

M14: *Anzahl der Dienste im System*

Ausgehend von der in der OO-Programmierung üblichen *Number of Classes*-Metrik zur Bestimmung der Größe eines Systems definieren Shim u. a. [128] die Anzahl der Dienste als Größenmaß für ein SOA-basiertes System. Es wird also der Faktor F12: [System | DESIGN-GRÖßE] gemessen.

F13: [System | FRAGMENTIERUNG]

Rud [123] versteht unter dem *Zusammenhalt* eines Systems die Anzahl der Aufrufbeziehungen zwischen verschiedenen Netzwerkknoten des Systems. Besonders fällt dabei die sprachliche Nähe des Begriffs Zusammenhalt mit den üblichen Definitionen von *Kohäsion* (vgl. F19: [Schnittstelle | KOHÄSION]) auf. Entgegen der üblichen positiven Betrachtung von Kohäsion in Bezug auf Qualität wird dort allerdings die Minimierung des Zusammenhalts angestrebt, so dass jeder Netzwerkknoten möglichst unabhängig von seiner Umgebung ist. Diese Entscheidung basiert auf dem Argument, dass Abhängigkeiten zwischen verschiedenen Knoten (und damit im Allgemeinen verschiedenen Zuständigkeitsbereichen) die Komplexität des Systems erhöhe. Aus diesem Grund ist der Begriff des Zusammenhaltes nicht unbedingt sinnvoll gewählt – *Fragmentierung* des Systems erscheint als treffendere Umschreibung. Komplett vermeiden lässt sich eine derartige Fragmentierung allerdings laut Rud nicht, da jeder Netzwerkknoten nur eine begrenzte Rechenkapazität besitze und deshalb nicht beliebig viele Dienste anbieten könne. Zur Quantifizierung dieses Faktors wird das Maß M15: *Rechner-externe Abhängigkeiten* herangezogen und durch die Anzahl aller Operationen im System normiert. Als initialer Grenzwert wird festgelegt, dass das System als sehr fragmentiert (F13 = 1) bewertet wird, wenn jeder fünfte Operationsaufruf zu einem anderen Netzwerkknoten erfolgt.

Legt man das Argument zugrunde, dass Fragmentierung die Komplexität erhöht, ergibt sich analog zu anderen Faktoren aus dem Bereich der Komplexität ein negativer Einfluss auf die Effizienz der Analyse. Darüber hinaus sind Auswirkungen auf die Effizienz der Kommunikation innerhalb des Systems möglich, was wiederum einen effizienten Betrieb desselben negativ beeinflusst.

⟶ Z8: [Interne Analyse | EFFIZIENZ]
⟶ Z6: [Betrieb | EFFIZIENZ]

M15: *Rechner-externe Abhängigkeiten*

Abhängigkeiten zu Diensten, die auf anderen Knoten im Netzwerk zur Verfügung stehen, werden bestimmt, indem pro Service-Operation die Existenz von Aufrufen solcher „externer“ Dienste festgestellt wird. Anschließend wird die Anzahl der Operationen bestimmt, die solche Aufrufe enthalten. Es ergibt sich eine nicht-negative ganze Zahl, die nach oben durch die Anzahl von Operationen im System beschränkt ist. Diese Zahl dient als Indikator für die Fragmentierung des Systems, welche in F13: [System | FRAGMENTIERUNG] beschrieben wird.

F14: [System | SCHNITTSTELLENKOMPLEXITÄT]

Die von außerhalb des Systems betrachtete Komplexität der Systemschnittstelle ist vor allem dann von Bedeutung, wenn ein SOA-basiertes IT-System in eine bestehende IT-Infrastruktur eingebunden werden soll (vgl. *Enterprise Architecture* in Abschnitt 2.3.1). In solchen Szenarien muss ein Experte das System daraufhin analysieren, auf welche Weise damit interagiert werden kann.

Einen Hinweis auf die Komplexität der Systemschnittstelle liefert die Anzahl der Operationen, die außerhalb des Systems sichtbar sind und aufgerufen werden können. Darüber hinaus tragen Dienste, welche bereits anderweitig angebotene Funktionalität zur Verfügung stellen, zur Erhöhung der Komplexität bei. Gibt es solche Dienste in einem System nicht, dann ist die Systemschnittstelle in dem Sinne minimal, dass keine Funktionalität mehrfach exponiert wird. Aus diesen Gründen werden die Maße M16: *Operationen an der Systemgrenze* und M17: *Semantisch äquivalente Dienste* zur Operationalisierung dieses Faktors verwendet, wobei letzteres doppelt so stark in die Bewertung des Faktors eingeht, da mit Blick auf eine sinnvolle Integration immer Operationen an der Systemgrenze exponiert sein müssen, ein redundantes Angebot derselben Funktionalität allerdings in jedem Fall zu vermeiden ist. Aus diesem Grund wird in der Bewertung der Schnittstellenkomplexität der Schwellenwert für M17 auf 0,1 festgelegt, so dass dieser Faktor bereits die schlechteste Bewertung erhält, wenn zu 10 % aller Dienste im System ein semantisches Äquivalent existiert.

⟞ Z8: [Interne Analyse | EFFIZIENZ]
⟞ Z11: [Komposition | EFFIZIENZ]

M16: *Operationen an der Systemgrenze*

Zur Bestimmung der in F14: [System | SCHNITTSTELLENKOMPLEXITÄT] beschriebenen Komplexität der Systemschnittstelle werden diejenigen Operationen gezählt, die von außerhalb des Systems aufgerufen werden können. Es werden bewusst Operationen statt Diensten gezählt, da etwa Mouchawrab, Briand und Labiche [106] die *number of system-level operations* als wichtiges Maß für die Komplexität einer Systemschnittstelle identifiziert haben. Um diese Anzahl zu ermitteln, müssen zunächst die Dienste identifiziert werden, die an der Systemgrenze exponiert sind. Die Summe ihrer Operationen ergibt den Messwert für M16.

M17: *Semantisch äquivalente Dienste*

Innerhalb eines SOA-Systems sollten keine Dienste verwendet werden, die semantisch äquivalent sind, da dies die Komplexität des Systems erhöht und die Komposition bzw. Integration erschwert [123]. Im Sinne von Konsistenz und Wiederverwendung sollte bei semantisch äquivalenten Diensten einer dieser Dienste ausgewählt und in der Folge exklusiv verwendet werden. Diese Analyse bedeutet einen Mehraufwand für potentielle Nutzer des Systems, und die Schnittstelle des Systems ist offensichtlich komplexer als nötig. Aus diesem Grund geht M17 in die Bewertung von F14: [System | SCHNITTSTELLENKOMPLEXITÄT] ein.

Anzahl	Einfach	Mittel	Komplex	Sehr Komplex
0	0.05	0.05	0.05	0.05
1-3	0.1	0.15	0.2	0.25
4-6	0.2	0.3	0.4	0.5
7-9	0.3	0.45	0.6	0.75
≥ 10	0.4	0.6	0.8	1.0

Tabelle 4.5: Ermittlung der Methodenkomplexität (Quelle: [126])

F15: [Schnittstelle | KOMPLEXITÄT]

Services mit besonders komplizierten Schnittstellen sind schwer verständlich [128]. Komplexität kann in diesem Zusammenhang in zwei Varianten vorliegen: Erstens kann die bloße Anzahl an Operationen und deren Parametern bereits ein Indikator für Komplexität sein. Die zweite Art der Komplexität ist die Größe der Datenstrukturen, welchen die Parameter genügen müssen. Da es prinzipiell immer möglich ist, eine Operation mit vielen Parametern durch eine funktional gleiche Operation mit einem einzigen, strukturell komplexeren Parameter zu ersetzen, müssen sowohl Anzahl als auch strukturelle Komplexität der Parameter betrachtet werden. Als entsprechende Maße hierfür werden M18: *Anzahl Operationen* und M19: *Operationskomplexität* verwendet.

- $\rightarrow$ Z2: [Externe Analyse | EFFIZIENZ]
- $\rightarrow$ Z8: [Interne Analyse | EFFIZIENZ]
- $\rightarrow$ Z11: [Komposition | EFFIZIENZ]

M18: *Anzahl Operationen*

Eine offensichtliche Eigenschaft eines Dienstes ist die Anzahl seiner Operationen. Diese liefert sowohl Hinweise auf die in F15: [Schnittstelle | KOMPLEXITÄT] beschriebene Komplexität des Dienstes, als auch auf die Granularität der Operationen und deren Parameter innerhalb eines Dienstes. Letzteres wird im folgenden Abschnitt als M23: *Datengranularität* detaillierter beschrieben. Darüber hinaus wird die Anzahl der Operationen als Normalisierungsgröße für andere Maße verwendet.

M19: *Operationskomplexität*

Die Komplexität von Operationen wird in Anlehnung an die von Sharma, Kumar und Grover [126] definierte Methodenkomplexität für objektorientierter Methoden basierend auf der Anzahl und Komplexität ihrer Parameter und Rückgabewerte bestimmt. Die Bewertung erfolgt dabei basierend auf den Werten aus Tabelle 4.5 und liefert Werte im Bereich zwischen 0.05 und 1.0. Da die Komplexität von Operationen maßgeblich zur Komplexität des Dienstes beiträgt, dem sie angehören, ist dieses Maß dem Faktor F15: [Schnittstelle | KOMPLEXITÄT] zugeordnet.

4.6.3 Granularität

Die Faktoren in diesem Abschnitt beschreiben Eigenschaften, die mit der Granularität von Diensten, Operationen und Parametern zusammenhängen. Granularität wird dabei sowohl im funktionalen Sinne verstanden als auch bezogen auf die ausgetauschten Daten. Auch die Kohäsion von Operationen wird hier beschrieben, da dieses Konzept stark mit der Granularität verwandt ist: Wenn Operationen keinen inhaltlichen Zusammenhang aufweisen, also eine geringe Kohäsion vorliegt, ist dies ein Hinweis auf eine nicht optimal gewählte Granularität des betreffenden Dienstes. Die hier beschriebenen Faktoren besitzen vor allem Einflüsse auf Qualitätsziele bezüglich der Wartung (Analyse, Anpassung) sowie der Nutzung (Komposition, Konsum) von Diensten.

F16: [Service | FUNKTIONALE GRANULARITÄT]

Die funktionale Granularität eines Dienstes beschreibt seine Eigenschaft, Funktionalität aus verschiedenen Domänen anzubieten. Bezüglich einer solchen funktionalen Domäne (z. B. Adressdatenverwaltung) sollte ein Dienst möglichst alle relevanten Operationen zur Verfügung stellen (etwa das Anlegen, Abfragen, Suchen, Ändern und Löschen von Adressdatensätzen). Allerdings sollten möglichst keine Teilfunktionalitäten anderer Domänen vorhanden sein (z. B. Bankverbindung abfragen). In dieser speziellen Hinsicht ist die funktionale Granularität mit der Kohäsion (vgl. F19: [Schnittstelle | KOHÄSION]) verwandt, die ebenfalls zum Ausdruck bringt, wie stark verschiedene Operationen eines Dienstes zusammenhängen. Eine direkte Betrachtung aus Sicht der abgebildeten Funktionalität ist unter dieser Bezeichnung allerdings nicht gebräuchlich.

Zur Quantifizierung der funktionalen Granularität eines Dienstes werden die Maße M20: *Berührte und voll abgedeckte funktionale Domänen* sowie M21: *Gegenseitige Dienstabhängigkeiten* herangezogen. Ersteres untersucht dabei den Zusammenhang zwischen funktionalen Domänen, aus welchen der Service Funktionalität anbietet und Domänen, die er vollständig abdeckt. Letzteres identifiziert Paare von Services im System, welche gegenseitig voneinander abhängen und daher vermutlich Teile derselben funktionalen Domäne betreffen.

Dienste, die eine optimale Granularität aufweisen, sind leicht in ihrer Funktionalität zu beurteilen, da sie eine Vermischung unterschiedlicher funktionaler Konzepte vermeiden. Darüber hinaus sind sie mit einer höheren Wahrscheinlichkeit für gegebene Anwendungen funktional geeignet. Granularität beeinflusst außerdem die Anpassbarkeit positiv, da Dienste mit optimaler Granularität leicht bei geänderten Bedingungen ausgetauscht werden können.

$\xrightarrow{+}$ Z2: [Externe Analyse | EFFIZIENZ]
$\xrightarrow{+}$ Z11: [Komposition | EFFIZIENZ]
$\xrightarrow{+}$ Z9: [Anpassung | EFFIZIENZ]

M20: *Berührte und voll abgedeckte funktionale Domänen*

Ein Dienst kann Funktionalität aus verschiedenen funktionalen Domänen, also semantisch nicht zusammenhängende Funktionalität, anbieten. Darüber hinaus ist es

möglich, dass nicht die gesamte Funktionalität von einem Dienst angeboten wird, die unter dem Gesichtspunkt der semantischen Verwandtschaft eine solche funktionale Domäne bildet. Diese beiden Phänomene werden analysiert, indem für jeden Dienst expertenbasiert festgestellt wird, wie viele semantisch verwandte Funktionseinheiten der Dienst enthält (nachfolgend: B) und wie viele davon vollständig abgebildet sind (nachfolgend: V).

Ein Dienst, welcher nur seine Kernfunktionalität bereitstellt, berührt nach dieser Definition eine einzige funktionale Domäne ($B = 1$). Im Gegensatz dazu wird ein trivialer (funktionsloser) Dienst keine Domäne berühren ($B = 0$), und ein Dienst, der Funktionalität verschiedener Art bereitstellt, ist mehreren Domänen zuzuordnen ($B > 1$). Die Bestimmung der vollständigen Abdeckung funktionaler Domänen geschieht analog, so dass einem Dienst, welcher die gesamte relevante Funktionalität einer Domäne anbietet, der Wert $V = 1$ zugewiesen wird. Interessant dabei ist auch der Vergleich von V und B, da beispielsweise ein Dienst, welcher zwei funktionale Domänen komplett abbildet ($B = V = 2$), zwar übergewichtig ist, aber zumindest keine unvollständigen Funktionseinheiten besitzt [123]. Aus diesen beiden Angaben wird nach dem in Tabelle 4.6 abgebildeten Schema der Messwert gebildet, welcher schließlich zur Bestimmung des Faktors F16: [Service | FUNKTIONALE GRANULARITÄT] herangezogen wird.

M21: *Gegenseitige Dienstabhängigkeiten*

Einen wichtigen Aspekt funktionaler Granularität (siehe F16: [Service | FUNKTIONALE GRANULARITÄT]) stellen gegenseitige Abhängigkeiten von Diensten dar. Wenn ein Dienst von einem anderen abhängt und umgekehrt, deutet dies darauf hin, dass beide Dienste auch inhaltlich eng verwandt sind, also dieselbe Domäne betreffen. Darüber hinaus kann davon ausgegangen werden, dass jeder der beiden Dienste nur eine unvollständige Menge von Funktionen dieser Domäne bereitstellt, da er anderenfalls nicht auf Funktionalität des jeweils anderen Dienstes angewiesen wäre. Zwei Dienste, die gegenseitig voneinander abhängen, zeugen also nicht von einer sinnvollen Geschäftslogik [123], sodass solche Dienst-Paare im Allgemeinen unerwünscht sind.

Paarweise voneinander abhängige Dienste können üblicherweise zu einem einzigen zusammengefasst werden, sofern sie vom selben Anbieter stammen. Ein Werkzeug zur Analyse der Kompositionslogik von Services kann dazu verwendet werden, solche

B	V	M20(s)
1	1	1.0
1	0	0.75
> 1	B	1 / B
> 1	< B	1 / (2 B)
0	0	0.0

Tabelle 4.6: Punkteschema für funktionale Domänen (nach Rud [123])

paarweisen Abhängigkeiten zu finden und die Fundstellen zu dokumentieren. Die Zielgröße ist dabei, keine solche Fundstelle im System vorzufinden. Die Dokumentation der Fundstellen kann dabei in einem beliebigen Format erfolgen – als Messwert für die weitere Aufbereitung der Qualitätsanalyse ist lediglich die Anzahl von Bedeutung.

F17: [Operation | FUNKTIONALE GRANULARITÄT]

In der Objektorientierung wurde die Atomarität von Methoden als ein wichtiger Faktor für die Wiederverwendbarkeit von Software identifiziert [23]. Eine Methode ist dann funktional atomar, wenn sie genau eine Funktion ausführt. Auch Service-Operationen sollten diese Eigenschaft besitzen, also genau eine Funktion ausführen, z. B. einen Datensatz abfragen. Inwieweit eine Operation funktional atomar ist, lässt sich durch das Maß M22: *Anzahl ausgeführter Funktionen* feststellen, welches die von der Operation ausgeführten Funktionen zählt. Die Atomarität ergibt sich als Kehrwert dieses Maßes.

Funktional atomare Service-Operationen sind einfacher komponierbar, weil atomare Funktionseinheiten in einer größeren Anzahl unterschiedlicher Situationen verwendet werden können. Eine atomare Operation kann also innerhalb mehrerer anderer Services von Nutzen sein, wohingegen Operationen, welche verschiedene Funktionen vereinen, sich seltener für eine Komposition eignen dürften.

$\xrightarrow{+}$ Z11: [Komposition | EFFIZIENZ]

M22: *Anzahl ausgeführter Funktionen*

Um die funktionale Granularität einer Operation zu beurteilen, ist eine manuelle Inspektion nötig. Basierend auf der Dokumentation des betreffenden Dienstes und ggf. des beobachteten Verhaltens während der Ausführung wird entschieden, wie viele Funktionen (im semantischen Sinne) von der Operation ausgeführt werden. Ein anschauliches Beispiel sind Operationen mit Seiteneffekten: Wenn eine Operation in erster Linie zur Abfrage eines bestimmten Datensatzes dient, gleichzeitig aber einen anderen Datensatz verändert, so führt sie zwei Funktionen aus. Operationen, welche genau eine Funktion ausführen, werden als funktional atomar bezeichnet. Die Anzahl ausgeführter Funktionen wird zur Bestimmung des Faktors F17: [Operation | FUNKTIONALE GRANULARITÄT] verwendet.

F18: [Service | DATENGRANULARITÄT]

Neben der in F16: [Service | FUNKTIONALE GRANULARITÄT] beschriebenen *funktionalen* Granularität von Operationen ist auch die Granularität der verarbeiteten *Daten* von Bedeutung, insbesondere die der exponierten Ein- und Ausgabeparameter. Rud [123] bezeichnet dieses Konzept als *Datenbezogene Granularität von Operationen eines Service*, beschreibt die Ermittlung von Messwerten allerdings über alle Operationen eines Dienstes hinweg, so dass eine Modellierung als Eigenschaft des Dienstes hier sinnvoller erscheint.

Shim u. a. [128] verwenden für die Beschreibung der Granularität von Diensten lediglich die Anzahl der Operationen und die Anzahl der insgesamt verwendeten Da-

tentypen, die sie miteinander ins Verhältnis setzen. Dabei werden eine höhere Anzahl an Operationen sowie eine geringere Anzahl an Datentypen pro Service als positiv betrachtet. Da optimale Granularität stark vom Anwendungsfall und damit von der Semantik der analysierten Operationen abhängt, erscheint diese Definition als nicht ausreichend und wird hier bewusst nicht verwendet. Stattdessen wird die Definition aus [123] aufgegriffen und in M23: *Datengranularität* beschrieben. Die Abbildung des Messwerts auf den Erfüllungsgrad des Faktors erfolgt dabei durch Definition einer Schranke für den Quotienten aus dem Messwert und der Anzahl an Operationen des Dienstes sowie einer linear fallenden Abbildungsfunktion (vgl. Abbildung 4.4 auf Seite 63).

Rud [123] argumentiert, redundante Datenübertragung sei oft effizienter als zusätzliche Aufrufe, da diese oft ressourcenintensiver seien [81]. Schnittstellen gröberer Granularität haben aufgrund der geringeren Anzahl von Operationen darüber hinaus den Vorteil einer besseren Lesbarkeit, sofern Parameter sinnvoll und selbsterklärend gewählt sind. Außerdem können Parameter gröberer Granularität über gemeinsame Typsysteme unternehmensweit genutzt werden, was die systeminterne Interoperabilität von Services erhöht.

Grobe Granularität erschwert allerdings das Puffern (Caching) von Daten, da redundante Anteile übertragener Daten nicht separat, sondern als Teil einer komplexeren Antwort übermittelt werden.

$\xrightarrow{+}$ Z2: [Externe Analyse | EFFIZIENZ]
$\xrightarrow{+}$ Z3: [Konsum | EFFEKTIVITÄT]
$\xrightarrow{+}$ Z11: [Komposition | EFFIZIENZ]
$\xrightarrow{+}$ Z9: [Anpassung | EFFIZIENZ]

M23: *Datengranularität*

Die in F18: [Service | DATENGRANULARITÄT] beschriebene Granularität der Ein- und Ausgabedaten von Diensten erfordert die Analyse der Operationen anhand ihrer Parameter. Dies ist im Allgemeinen nur unter Berücksichtigung von Domänenwissen effizient möglich, weshalb eine manuelle Analyse vorgeschlagen wird. Allerdings kann diese Analyse automatisiert unterstützt werden, indem Operationen mit gemeinsamen Parametern gruppiert werden.

Analog zu Rud [123] wird hier die Anzahl der Operationen zur Quantifizierung der Datengranularität verwendet, welche durch Zusammenfassen feingranularer Operationen entfallen könnten. Dazu werden zunächst Gruppen von Operationen gebildet, die sich zusammenfassen lassen (z. B. das Abfragen von Name, Adresse und Telefonnummer einer Person). Anschließend werden die entstandenen Gruppen (z. B. Abfrage der Kontaktdaten) gezählt. Die Differenz dieser Werte ergibt die Anzahl der eingesparten Operationen (in diesem Fall $3 - 1 = 2$) und damit den Messwert.

F19: [Schnittstelle | KOHÄSION]

Als Kohäsion eines Dienstes wird der inhaltliche Zusammenhang seiner Operationen bezeichnet. Der Begriff des *Zusammenhangs* ist hier bewusst allgemein und wird in der Literatur auch in Bezug auf Kohäsion sehr breit interpretiert. Stevens, Myers und Constantine [131] etwa beschreiben Kohäsion als Ausprägung mehrerer Arten von Zusammenhang der Elemente eines Softwaremoduls:

> Binding is the measure of the cohesiveness of a module. [...] The scale of cohesiveness, from lowest to highest, follows: 1. Coincidental. 2. Logical. 3. Temporal. 4. Communicational. 5. Sequential. 6. Functional. [131]

Die Allgemeinheit dieses Begriffs wird durch die Definition verschiedener Maße adressiert, welche jeweils einen Aspekt des Zusammenhangs quantifizieren: M24: *Gemeinsame Datentypen für Parameter*, M25: *Nutzung von Operationen durch Dienstkonsumenten* sowie M26: *Abfolge-Zusammenhang zwischen Operationen*. Damit ist der hier verfolgte Ansatz deutlich differenzierter als der von Shim u. a. [128], die als einzigen Indikator für die Kohäsion von Diensten den Kehrwert der mittleren Anzahl an verwendeten Datentypen verwenden.

Der Kohäsion einer Service-Schnittstelle werden verschiedene positive Eigenschaften in Bezug auf ihre Qualität zugeschrieben: Zum einen sind kohäsive Dienste einfacher zu verstehen und deshalb insbesondere einfacher zu komponieren. Darüber hinaus erfüllen sie mit größerer Wahrscheinlichkeit funktionale Anforderungen, da ein Dienst mit schwach zusammenhängender Funktionalität für ein gegebenes Nutzungsszenario im Allgemeinen nicht ohne Einschränkungen einsetzbar ist [118, 128].

- $\xrightarrow{+}$ Z2: [Externe Analyse | EFFIZIENZ]
- $\xrightarrow{+}$ Z8: [Interne Analyse | EFFIZIENZ]
- $\xrightarrow{+}$ Z11: [Komposition | EFFIZIENZ]
- $\xrightarrow{+}$ Z3: [Konsum | EFFEKTIVITÄT]

M24: *Gemeinsame Datentypen für Parameter*

Dieses Maß ist eines von drei Maßen, welche den Faktor F19: [Schnittstelle | KOHÄSION] quantifizieren. Perepletchikov, Ryan und Tari [118] schlagen vor, die Anzahl aller Parameter und Rückgabewerte zu bestimmen, welche gemeinsame Datentypen verwenden, um einen Aspekt von Kohäsion zu bestimmen, den sie als *Kommunikationskohäsion* bezeichnen. Maximale Kommunikationskohäsion wird dabei erreicht, wenn alle Operationen eines Dienstes gemeinsame Datenstrukturen verwenden, um ihre Parameter und Rückgabewerte zu definieren. Shim u. a. [128] verfolgen den gleichen Ansatz in ihrer Definition der *Number of Messages Used*-Metrik und der Beschreibung von Datenkohäsion, welche identisch mit der o. g. Kommunikationskohäsion ist. Allerdings werden in diesem Fall nicht die gemeinsamen, sondern die unterschiedlichen Datentypen gezählt. Als Resultat ist die Metrik einfacher, besitzt allerdings eine invertierte Logik, da wenige unterschiedliche Datentypen (ähnlich wie viele gemeinsam genutzte) eine hohe Kohäsion bedeuten.

Da die Bestimmung eines Messwertes lediglich einfaches Zählen von Datentypen erfordert, kann sie mit geringem Aufwand automatisch durchgeführt werden. Analog zur Definition der Kommunikationskohäsion nach Perepletchikov, Ryan und Tari werden hierzu zunächst alle kombinatorisch möglichen Paare von Operationen gebildet. Der Messwert ergibt sich aus dem Prozentsatz dieser Paare, welche mindestens einen gemeinsamen Datentyp in den Ein- oder Ausgabeparametern besitzen.

M25: *Nutzung von Operationen durch Dienstkonsumenten*

Ein weiterer Aspekt des Faktors F19: [Schnittstelle | KOHÄSION] wird von Perepletchikov, Ryan und Tari [118] als *externe Kohäsion* bezeichnet, weil dabei die Nutzung im Vordergrund steht, also eine Außensicht eingenommen wird. Zur Bestimmung des Messwertes wird pro Operation ermittelt, wie viele der Dienstkonsumenten diese Operation verwenden. Dieses Maß gründet auf der Idee, dass eine Operation, welche nur einen geringen Zusammenhang zu den übrigen Operationen des Dienstes aufweist, auch nur von einer geringen Anzahl der Konsumenten des Dienstes verwendet wird. Im Gegensatz dazu ist anzunehmen, dass ein kohäsiver Service oft derart konsumiert wird, dass viele oder alle seiner Operationen von den Konsumenten verwendet werden. Maximale externe Kohäsion wird erreicht, wenn alle Operationen von allen Konsumenten verwendet werden.

Dieses Maß kann nur dann erhoben werden, wenn der Dienst tatsächlich genutzt wird, wozu auch Tests gezählt werden können. In diesem Fall bietet sich die Auswertung von Log-Dateien an – in jedem Fall ist eine automatische Messung angebracht.

M26: *Abfolge-Zusammenhang zwischen Operationen*

Ein Abfolge-Zusammenhang zwischen Operationen äußert sich darin, dass die Ausgabe oder Nachbedingung einer Operation gleichzeitig Eingabe oder Vorbedingung einer anderen Operation desselben Dienstes ist (vgl. *Sequential Cohesion* in [118]). Offensichtlich ist auch dies ein Kriterium zur Bestimmung des Faktors F19: [Schnittstelle | KOHÄSION]. Informationen über den Abfolge-Zusammenhang zweier Operationen kann zum einen mittels der Analyse gemeinsam genutzter Datentypen ermittelt werden (siehe M24: *Gemeinsame Datentypen für Parameter*); darüber hinaus kann sie aus Geschäftsprozessmodellen oder Service-Kompositionen abgeleitet werden. Da diese Artefakte üblicherweise in maschinenlesbarer Form vorliegen, kann auch dieses Maß automatisch erhoben werden. Für den Fall, dass die benötigten Artefakte allerdings zum Zeitpunkt der Analyse nicht verfügbar sind, ist auch eine manuelle Einschätzung durch einen Domänenexperten denkbar. Der Messwert wird dabei wie von Perepletchikov, Ryan und Tari [118] vorgeschlagen bestimmt: Basierend auf Vor- und Nachbedingungen sowie Ein- und Ausgabeparametern der Operationen eines Dienstes wird bestimmt, zwischen wie vielen Paaren von Operationen ein Abfolge-Zusammenhang besteht. Der Messwert ergibt sich aus dem Verhältnis dieser Anzahl zur Anzahl aller möglichen Paare von Operationen, für einen Service mit n Operationen $0,5 \cdot n \cdot (n-1)$. Daraus ergibt sich unmittelbar ein Wertebereich zwischen Null und Eins.

4.6.4 Abhängigkeiten von Diensten und Operationen

Trotz des SOA-Prinzips der losen Kopplung sind in der Praxis häufig direkte Abhängigkeiten zwischen Diensten im Allgemeinen und deren Operationen im Besonderen zu beobachten [103]. Die folgenden Faktoren beschreiben daher verschiedenen Eigenschaften von Diensten, die aus diesen Abhängigkeiten resultieren. Dabei wird neben der direkten Abhängigkeit in Form von Aufrufbeziehungen zwischen Diensten auch die indirekte Abhängigkeit zwischen Operationen betrachtet. Diese wird vor allem von der zugrundeliegenden Geschäftslogik verursacht und äußert sich in Form von zustandsbehafteten Diensten, in denen der Aufruf bestimmter Operationen nur unter Vorbedingungen möglich ist, zu deren Erfüllung zuvor eine andere Operation aufgerufen werden muss. Die hier beschriebenen Faktoren besitzen in erster Linie Einflüsse auf Qualitätsziele aus dem Bereich des Betriebs und der Nutzung serviceorientierter Systeme.

F20: [Service | KRITIKALITÄT]

Rud [123] betrachtet Dienste als *kritisch*, die eine große Bedeutung für das Gesamtsystem besitzen. Dies ist insbesondere bei Diensten der Fall, von denen viele weitere Dienste abhängen. Eine Fehlfunktion eines solchen Dienstes hat potentiell stärkere Auswirkungen auf das Gesamtsystem, da mehrere Dienste davon betroffen sind. Zusätzlich birgt ein Dienst, der seinerseits viele Abhängigkeiten besitzt, ein erhöhtes Risiko bezüglich Fehlfunktion oder Ausfall, da Fehlfunktionen in jedem einzelnen der verwendeten Dienste sich auf die Funktion des Dienstes auswirken können, sofern nicht für jeden der verwendeten Dienste ausreichende Maßnahmen zur Ausnahmebehandlung getroffen werden. Die Gesamtheit dieser Eigenschaften wird über das Maß M28: *Minimale Dienst-Zuverlässigkeit* erfasst, das wiederum auf die Maße M27: *Abhängigkeitsprodukt*, M9: *Fan-In* und M1: *Fan-Out* zurückgreift.

Die Auswirkungen werden sowohl aus Betreiber- als auch aus Nutzersicht beschrieben, weshalb sich negative Einflüsse sowohl auf die Kontinuität des Betriebs als auch der Nutzung von Diensten ergeben.

$\longrightarrow$ Z7: [Betrieb | KONTINUITÄT]
$\longrightarrow$ Z5: [Konsum | KONTINUITÄT]

M27: *Abhängigkeitsprodukt*

Um das Ausfallrisiko eines Dienstes zu beschreiben verwendet Rud [123] das Produkt aus M9: *Fan-In* und M1: *Fan-Out*, ausgehend von der Annahme, dass häufig verwendete Dienste ebenso besondere Aufmerksamkeit hinsichtlich ihrer Zuverlässigkeit erfordern wie solche, die von vielen anderen Diensten abhängen. Diese Eigenschaft eines Dienstes wird im Maß M28: *Minimale Dienst-Zuverlässigkeit* für alle Dienste eines Systems zusammengefasst. Der Wert dieses Maßes kann direkt aus den beiden genannten Maßen berechnet werden:

$$\mathrm{M27}(s) = \mathrm{M9}(s) \cdot \mathrm{M1}(s)$$

Die multiplikative Verknüpfung der eingehenden und ausgehenden Abhängigkeiten ist darin begründet, dass ein Dienst eine umso tragendere Rolle für das gesamte System spielt, je größer die Werte beider beitragenden Maße sind. Der *Fan-In* repräsentiert dabei die Auswirkungen einer möglichen Fehlfunktion, während der *Fan-Out* das Risiko einer solchen beschreibt. Um die Schwere eines Ausfalls zu beschreiben, liegt eine multiplikative Verknüpfung nahe, so dass etwa ein doppelter *Fan-In* bei gleichem *Fan-Out* eine Verdopplung des Messwertes zur Folge hat.

M28: *Minimale Dienst-Zuverlässigkeit*

Das zuvor beschriebene Produkt der eingehenden und ausgehenden Abhängigkeiten einzelner Dienste muss für die Bestimmung des Faktors F20: [Service | KRITIKALITÄT] zu einer Aussage über das Gesamtsystem zusammengefasst werden. Aufgrund der Tatsache, dass es sich hierbei um eine Risikobetrachtung handelt und der Ausfall oder die Fehlfunktion eines für das System wichtigen Dienstes erhebliche Auswirkungen auf das Gesamtsystem haben kann, erscheint es sinnvoll, eine Aggregation zu definieren, welche das größte identifizierte Risiko widerspiegelt. In Anlehnung an die Metapher, dass eine Kette nur so stark ist wie ihr schwächstes Glied, ergibt sich für die Zuverlässigkeit von Diensten:

$$\mathrm{M28} = \frac{1}{\max_{s \in S} \mathrm{M27}(s)}$$

wobei S die Menge aller Dienste im System ist, die verwendet werden. Die Kritikalität wird dabei für jeden Dienst mittels M27: *Abhängigkeitsprodukt* bestimmt [123].

F21: [Service | ZUSTANDSLOSIGKEIT]

Ein Dienst ist zustandslos, wenn er ausschließlich Operationen besitzt, die unabhängig von vorherigen Aufrufen verwendet werden können. Im Gegensatz dazu sind Mechanismen zum *Session Handling* ein Mittel, um Zustände zu verwalten. Zustandslose Dienste vermeiden Verwaltungsaufwand innerhalb des Dienstes und begünstigen etwa Methoden zum Lastausgleich (*Load Balancing*), da jeder Aufruf einer Funktion von einer beliebigen freien Ressource verarbeitet werden kann, unabhängig davon, welche Ressource vorherige Anfragen verarbeitet hat. Eine detailliertere Betrachtung des Konzeptes *Zustand* ist in Abschnitt 4.5.6 auf Seite 91 zu finden. Das Maß M29: *Anzahl kontextabhängiger Operationen* zur Bestimmung der Zustandslosigkeit analysiert einzelne Operationen und bestimmt die Anzahl derer, die vom aktuellen Anwendungskontext abhängen. Liegt diese Anzahl über einem Zehntel der Anzahl aller Operationen im System, so wird dieser Faktor mit Null bewertet.

Die positiven Auswirkungen der Zustandslosigkeit hinsichtlich Lastverteilung tragen dazu bei, die entsprechenden Dienste effizient betreiben zu können.

$\xrightarrow{+}$ Z6: [Betrieb | EFFIZIENZ]

M29: *Anzahl kontextabhängiger Operationen*

Kontextabhängige Operationen sind solche Operationen, die sich nicht in jeder Situation aufrufen lassen, z. B. weil sie Eingabedaten erwarten, die durch vorherigen Aufruf anderer Operationen erzeugt werden müssen (siehe auch M26: *Abfolge-Zusammenhang zwischen Operationen*) oder sonstige, spezifische Vorbedingungen erfüllt sein müssen. Operationen, die vom aktuellen Anwendungszustand abhängen, besitzen also die in F21: [Service | ZUSTANDSLOSIGKEIT] beschriebene Eigenschaft nicht und können damit (in Relation zu jenen Operationen, welche vom aktuellen Anwendungszustand unabhängig sind) zur Quantifizierung dieses Faktors herangezogen werden. Um zu bestimmen, ob Operationen nach dieser Definition kontextabhängig sind, ist eine manuelle Analyse erforderlich. Einen Hinweis auf Kontextabhängigkeit bieten Parameter, welche als Identifikator für einen vom Service verwalteten Zustand dienen (z. B. *Session IDs*).

F22: [Service | ABHÄNGIGKEITSPFADE]

Bereits bei der Diskussion des SOA-Prinzips der Entkopplung in F8: [Service | ENTKOPPLUNG] wurde argumentiert, dass statisch definierte Abhängigkeiten zwischen Services in der Praxis häufig vorkommen. Aus diesem Grund ist die Betrachtung der Transitivität dieser Abhängigkeiten, die durch Komposition entsteht, von besonderer Bedeutung. In diesem Zusammenhang werden hier sowohl die Tiefe der Abhängigkeiten (M31: *Abhängigkeitstiefe*) als auch deren Summe (M30: *Abhängigkeitssumme*) für einen komponierten Dienst betrachtet. Die Gesamtanzahl aller Abhängigkeiten ist dabei vor allem ein Hinweis auf mögliche Zuverlässigkeitsprobleme, da der Service bei Ausfall oder Fehlfunktion jedes Dienstes, von dem er direkt oder indirekt abhängt, möglicherweise ebenfalls unzuverlässig arbeitet. Die Tiefe der geschachtelten Abhängigkeiten weist eher auf mögliche Effizienzprobleme hin, da Aufrufe kaskadiert werden und je nach Leistungsfähigkeit und Auslastung des verwendeten Netzwerks zusätzliche Verzögerungen entstehen können.

Beide Maße gehen zu gleichen Teilen in die Bewertung ein, wobei für die Summe der Abhängigkeiten ein Grenzwert von Fünf für die volle Erfüllung des Faktors festgelegt wird; im Fall der Tiefe wird ab einem Wert von Drei die volle Erfüllung angenommen.

⟶ Z4: [Konsum | EFFIZIENZ]
⟶ Z5: [Konsum | KONTINUITÄT]

M30: *Abhängigkeitssumme*

In diesem Maß zur Quantifizierung des Faktors F22: [Service | ABHÄNGIGKEITSPFADE] werden analog zu Rud [123] alle Dienste gezählt, welche direkt oder indirekt zur Ausführung eines zusammengesetzten Dienstes benötigt werden. Der Wert dieses Maßes für einen Dienst ist immer größer oder gleich der Anzahl seiner direkten Abhängigkeiten, welche in M1: *Fan-Out* gezählt werden.

$$\mathrm{M30}(s) \geq \mathrm{M1}(s)$$

Elementtyp	Basis	Konformität	Design	Gesamt
Ziel	12	-	-	12
Faktor	-	8	14	22
Einfluss	-	13	33	46
Maß	-	9	22	31
Summe	12	30	69	111

Tabelle 4.7: Aufteilung der Elemente auf die Module des Qualitätsmodells

Dieses Maß ermöglicht die Analyse über mehrere Kompositionsebenen hinweg und kann damit präziser auf mögliche Zuverlässigkeits- oder Effizienzprobleme hindeuten.

M31: *Abhängigkeitstiefe*

Neben der zuvor beschriebenen Summe der Abhängigkeiten eines Dienstes geht auch die Tiefe der Abhängigkeiten in die Bewertung von F22: [Service | ABHÄNGIGKEITSPFADE] ein. Dabei werden alle Abhängigkeiten des Dienstes transitiv verfolgt und die maximale Anzahl kaskadierter Abhängigkeiten ermittelt.

4.7 Zusammenfassung

In diesem Kapitel wurde ein Qualitätsmodell für SOA vorgestellt. Die Qualitätsziele für eine SOA werden ausgedrückt, indem Eigenschaften von Aktivitäten gefordert werden, welche mit einem Dienst oder dem System durchgeführt werden, z. B. [Komposition | EFFIZIENZ]. Faktoren, welche einen Einfluss auf diese Qualitätsziele besitzen, werden in ähnlicher Art und Weise als Eigenschaften von Teilen des Dienstes oder des Systems dargestellt: [Schnittstelle | KOMPLEXITÄT], wobei der Einfluss selbst positiv oder negativ modelliert werden kann: [Schnittstelle | KOMPLEXITÄT] $\dashrightarrow$ [Komposition | EFFIZIENZ].

Inhaltlich wurde das Qualitätsmodell mit Hilfe einer Menge von aktivitätenbasierten Qualitätszielen $\mathbb{A}$, einer Menge von Faktoren $\mathbb{F}$ und einer Menge von Maßen $\mathbb{M}$ definiert. Dabei dienen Maße der Quantifizierung von Faktoren, welche wiederum Einflüsse auf Qualitätsziele besitzen. Das Qualitätsmodell ist auf zwei Ebenen anwendbar: Die korrekte Umsetzung der SOA-Prinzipien und davon beeinflusste Qualitätsziele sowie entsprechende Maße wurden in Abschnitt 4.5 beschrieben. Dieses Modul des Qualitätsmodells bildet die Grundlage für die Definition weiterer für die Produktqualität relevanter (Design-) Faktoren und Maße, welche in Abschnitt 4.6 beschrieben wurden.

Insgesamt enthält das Modell zwölf aktivitätenbasierte Qualitätsziele, welche über 43 Einflüsse mit 22 Faktoren in Beziehung stehen, die wiederum über 31 Maße quantifiziert werden können. Eine Aufteilung dieser Kennzahlen auf die Module SOA-Basis, SOA-Konformität und SOA-Design ist in Tabelle 4.7 zu finden.

Abbildung 4.11 zeigt eine Übersicht des Qualitätsmodells in Form eines *Sunburst*-Diagramms [16]. Auf der linken Seite ist die Hierarchie der Faktoren visualisiert, welche sich in SOA-Designfaktoren (oben) und SOA-Prinzipien (unten) unterteilen. Die rechte Seite des Diagramms zeigt die Hierarchie der Qualitätsziele. Diese sind auf der ersten Ebene anhand der Aktivitäten (in alphabetischer Reihenfolge: Analyse, Anpassung, Betrieb, Komposition, Konsum, Test) unterteilt, welche auf der inneren Ebene durch Attribute charakterisiert werden. Exemplarisch ist der Bereich der SOA-Designfaktoren hervorgehoben und die Einflüsse für alle untergeordneten Faktoren dargestellt.

Darüber hinaus sind weitere Informationen aus diesem Diagramm abzulesen. So kann über die Farbgebung der einzelnen Felder das Ergebnis der Qualitätsbewertung für ein SOA-basiertes System dargestellt werden. Nähere Informationen zu den hier dargestellten Ergebnissen werden in Abschnitt 5.3.2 beschrieben. Die relative Wichtigkeit benachbarter Felder kann zudem durch unterschiedlichen Flächeninhalt dargestellt werden. Da die Visualisierung einer solch großen Informationsmenge in statischer Form nur sehr schwer zu bewerkstelligen ist, sind solche Darstellungen als Teil des Quamoco-Editors für Qualitätsmodelle [45] implementiert und interaktiv nutzbar.

Die Anpassbarkeit der Qualitätsbewertung an spezifische Anforderungen für eine Anwendungsdomäne oder ein konkretes Softwareprojekt ist durch die Möglichkeit gegeben, in einem zusätzlichen Modul des Qualitätsmodells weitere Faktoren sowie Maße zu definieren oder für bestehende Maße eine Erhebung der Messwerte durch spezifische Werkzeuge festzulegen. Außerdem können bestehende Bewertungsregeln innerhalb dieses neuen Moduls überschrieben werden, wodurch eine Neugewichtung von Maßen innerhalb eines Faktors sowie von Faktoren innerhalb eines Qualitätsziels ermöglicht wird. Damit lassen sich neben der projektspezifischen Anpassung auch unterschiedliche Perspektiven auf die Produktqualität realisieren, etwa die des Service-Anbieters und die des Service-Nutzers. Details zu dieser Modularisierungsstrategie sind in Wagner u. a. [10] veröffentlicht.

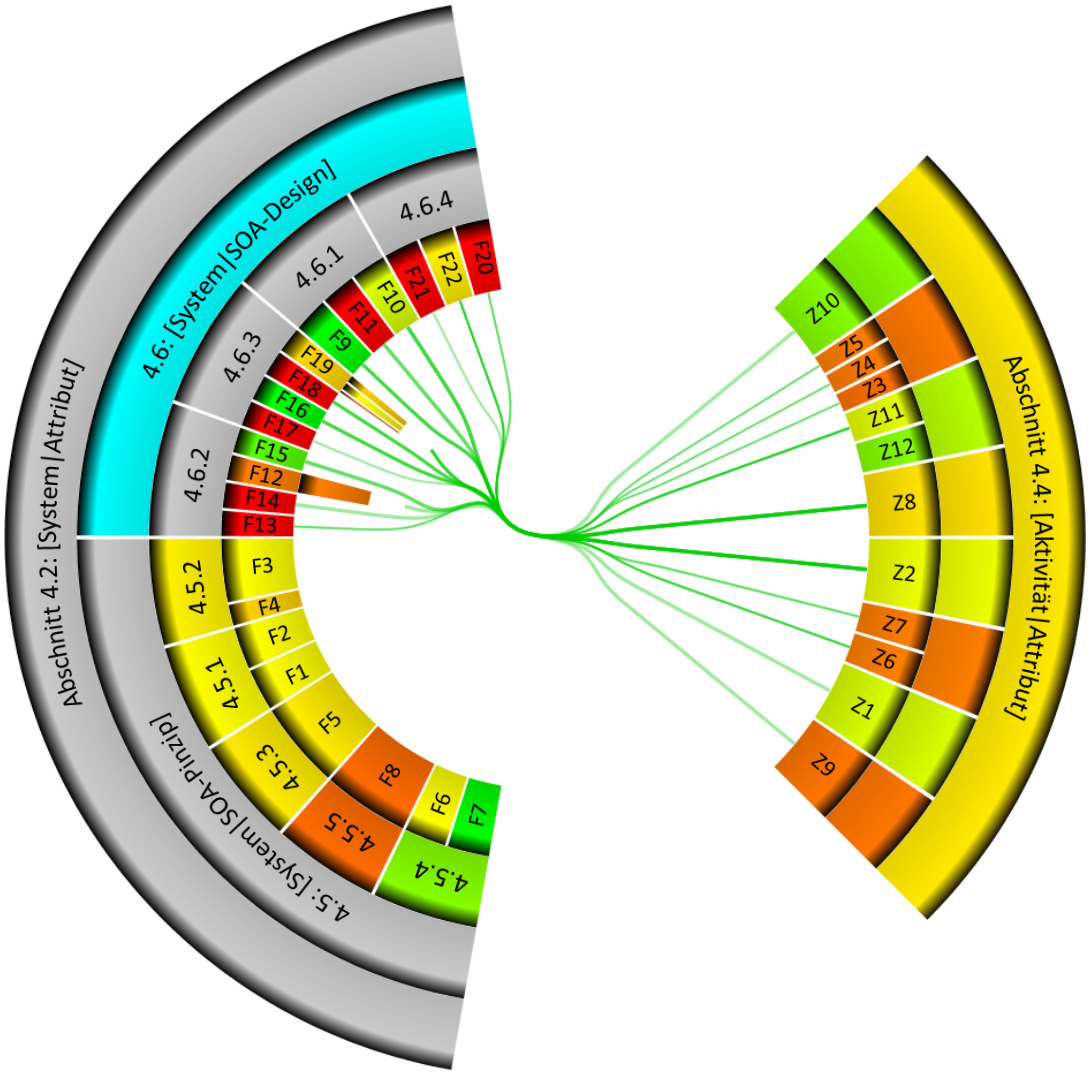

Abbildung 4.11: SOA-Qualitätsmodell – Übersicht

5 Expertenstudie und Anwendung

5.1 Einleitung

Das im vorigen Kapitel beschriebene Qualitätsmodell für SOA wird im Folgenden einer kritischen Betrachtung unterzogen. In Abschnitt 5.2 wird mittels einer Befragung von Experten auf den Gebieten SOA und Softwarequalität ermittelt, inwieweit das Qualitätsmodell die Anforderungen von Softwarearchitekten erfüllt. Besonderes Augenmerk liegt dabei auf der Angemessenheit der modellierten Konstrukte sowie der vollständigen Abdeckung der relevanten Faktoren, welche innerhalb SOA-basierter Softwaresysteme die Produktqualität beeinflussen. Abschnitt 5.3 zeigt verschiedene Szenarien auf, in denen das Qualitätsmodell verwendet werden kann. Zunächst werden die im Qualitätsmodell erfassten Informationen verwendet, um einige der Forschungsfragen zu adressieren, die diese Dissertation aufgeworfen hat. Darüber hinaus wird das Qualitätsmodell zur exemplarischen Bewertung der Qualität einer SOA verwendet. Dieses Kapitel zeigt also Stärken und Schwächen des Qualitätsmodells aus verschiedenen Blickwinkeln auf: Struktur und Inhalt des Modells werden direkt von Experten bewertet und diskutiert, sie werden aber auch indirekt über die Anwendung des Modells in typischen Nutzungsszenarien auf ihre Tragfähigkeit und Aussagekraft hin untersucht.

5.2 Statische Inspektion des Qualitätsmodells

In Anlehnung an die Entwicklung von Software, in der die statische Analyse als ein probates Mittel gilt, Fehler frühzeitig (und insbesondere vor der Ausführung) zu finden und zu beheben, wird auch das Qualitätsmodell für SOA vor der praktischen Anwendung statisch überprüft. Strukturelle Konsistenz konnte dabei vorab bereits durch entsprechende Regeln im Qualitätsmodell-Editor des Quamoco-Projekts[1] überprüft werden. Diese Regeln reichen von der einfachen Überprüfung unreferenzierter (isolierter) Modellelemente bis hin zu Regeln, welche die modulare Struktur des Modells überprüfen. Wenn etwa ein Faktor im Konformitätsmodul einen Einfluss auf ein Qualitätsziel im Modul SOA-Basis beschreibt, muss das SOA-Konformitätsmodul gemäß der Quamoco-Syntax angeben, dass es vom Basismodul abhängt. Verletzungen solcher Regeln werden dem Benutzer im Editor direkt angezeigt, so dass davon ausgegangen werden kann, dass das im vorigen Kapitel beschriebene Qualitätsmodell keine schwerwiegenden syntaktischen oder strukturellen Fehler aufweist. Die inhaltliche Bewertung des Modells erfordert jedoch eine manuelle Analyse und wird daher durch eine Expertenbefragung operationalisiert, welche im Folgenden beschrieben wird.

[1] `https://quamoco.in.tum.de/tools`

Die Inspektion des Modells durch Experten soll vor allem Erkenntnisse bezüglich der Angemessenheit und Vollständigkeit des Modells hervorbringen. Angemessenheit wird dabei als die Eigenschaft des Qualitätsmodells betrachtet, SOA-spezifisch zu sein und keine Konzepte abzubilden, welche im Kontext von SOA keine Relevanz besitzen. Vollständigkeit ist in diesem Zusammenhang die Eigenschaft des Qualitätsmodells, alle für die Qualität einer SOA relevanten Faktoren zu beschreiben.

5.2.1 Methodik

Die Vorgehensweise zur Analyse der Angemessenheit und Vollständigkeit des SOA-Qualitätsmodells orientiert sich am *Capture-Recapture*-Verfahren, das ursprünglich für Anwendungen in der Biologie entwickelt wurde [34, 114]. Das Prinzip hinter dem Verfahren ist die Abschätzung einer Eigenschaft einer Gesamtheit (dort: Gesamtpopulation einer Tierart) auf Basis der Beobachtung dieser Eigenschaft für einen Teil dieser Gesamtheit (dort: eine bestimmte Menge an Tieren dieser Art). Briand u. a. [31] sowie Petersson u. a. [120] haben das Verfahren auf das Software Engineering übertragen und zur Abschätzung der Fehlerrate in Software verwendet. Dabei werden Stichproben des Programmcodes durch Experten auf Fehler inspiziert und durch Anwendung von Schätzverfahren Aussagen über die Anzahl der Fehler im Gesamtsystem abgeleitet.

Die verwendeten Schätzverfahren werden dabei in Abhängigkeit vom Grad der Homogenität der Experten sowie der Inspektionsergebnisse ausgewählt. Dieses Vorgehen basiert auf der Annahme, dass Experten mit sehr unterschiedlichem Erfahrungshorizont tendenziell eher unterschiedliche Fehler finden, wohingegen bei sehr ähnlich erfahrenen Experten davon ausgegangen werden kann, dass auch ähnliche Fehler identifiziert werden. Gleichermaßen kann man bei hoher Übereinstimmung der gefundenen Fehler darauf schließen, dass diese bereits einen Großteil der tatsächlich vorhandenen Fehler darstellen, wohingegen stärker differierende Mengen identifizierter Fehler den Schluss nahelegen, dass noch weitaus mehr Fehler im System vorhanden sind, als von den Experten gefunden wurden.

Für die Inspektion von Qualitätsmodellen auf Vollständigkeit wurde das Verfahren dahingehend angepasst, dass statt nach Fehlern in Programmcode nach fehlenden Modellelementen gesucht wird. Basierend auf den Eigenschaften der Experten sowie der identifizierten fehlenden Modellelemente kommen nach Petersson u. a. [120] unterschiedliche *Capture-Recapture*-Modelle und damit unterschiedliche Schätzverfahren für die Generalisierung der Ergebnisse in Frage. Dabei bezeichnet *Heterogenität* Unterschiede in der Art der identifizierten Elemente sowie *Zeit* die vom Experten benötigte Zeit, diese zu identifizieren. Letzterem liegt die Annahme zu Grunde, dass einige Experten aufgrund ihrer persönlichen Erfahrung Fehler schneller finden als andere. Tabelle 5.1 fasst diese Modelle zusammen.

Für das SOA-Qualitätsmodell bedeutet dies, dass die Schätzer in Abhängigkeit der Homogenität der Experten sowie der identifizierten Lücken im Modell ausgewählt werden. Ersteres wird durch die Frage nach der Erfahrung der Experten mit serviceorientierten Architekturen operationalisiert (siehe Frage Q3.4 in Tabelle 5.3), letzteres durch die subjektive Einschätzung der Experten, wie offensichtlich eine Lücke war (sie-

Modell	Vorbedingungen	Schätzer
M0	Homogene Elemente Homogene Reviewer	M0-ML – maximum likelihood [114]
Mt	Homogene Elemente Heterogene Reviewer	Mt-ML – maximum likelihood [114] Mt-Ch – Chao [38]
Mh	Heterogene Elemente Homogene Reviewer	Mh-JK – Jackknife [34] Mh-Ch – Chao [38]
Mth	Heterogene Elemente Heterogene Reviewer	Mth-Ch – Chao [38]

Tabelle 5.1: Schätzmodelle für Capture-Recapture (Quelle: [31])

he Frage Q2.3). Dabei kommt eine Fünf-Punkt-Likert-Skala zum Einsatz, wobei bei einer Standardabweichung von $\sigma < 1$ von Homogenität ausgegangen wird, anderenfalls von Heterogenität.

Die Inhalte des SOA-Qualitätsmodells wurden vorrangig auf Basis wissenschaftlicher Publikationen entwickelt. Aus diesem Grund soll die Befragung vor allem Experten aus der Praxis adressieren, um die Relevanz dieser Inhalte auch aus dem Blickwinkel der Industrie zu beleuchten. Daraus ergibt sich die Auswahl der für die Befragung eingeladenen Experten, welche zu einem großen Teil aus der industriellen Softwareentwicklung stammen, und zu einem geringeren Teil aus dem akademischen Umfeld. Die befragten Experten wurden basierend auf ihrer Erfahrung mit SOA und Qualitätssicherung in SOA-Projekten ausgewählt.

Zur Anzahl der mindestens benötigten Experten werden im Wesentlichen zwei Arbeiten herangezogen: Für *Capture-Recapture*-Experimente schlagen Briand u. a. [31] mindestens vier bis fünf Experten vor. Diese Anzahl wurde als untere Schranke verwendet. Vegas und Basili [137] argumentieren in ihrer Anwendung einer Expertenbefragung dahingehend, dass eine Befragung weiterer Experten keine neuen Informationen mehr zur Gesamtheit der aus den vorherigen Befragungen hervorgegangenen Erkenntnisse hervorbringen würde. Eine ausreichende Anzahl von Experten sei also genau dann befragt worden, wenn sich eine Sättigung bezüglich neuer Verbesserungsvorschläge einstellt. In Anlehnung daran wurde hier nach jedem Interview untersucht, inwieweit sich ggf. vorgeschlagene, zusätzliche Elemente von den Vorschlägen vorhergehender Interviews unterschieden. So konnte im Verlauf der Befragungen eine Menge an Vorschlägen verwaltet und sukzessive erweitert werden, unter der Annahme, dass diese Menge im Laufe der Zeit langsamer wächst und schließlich annähernd konstant bleibt. Ist dies über mehrere Interviews hinweg der Fall, liegt der Schluss nahe, dass das Modell inklusive der von den Experten eingebrachten Vorschläge eine ausreichende Vollständigkeit erreicht hat und die Befragungen abgeschlossen werden können.

Insgesamt wurden zwölf Experten nach dieser Vorgehensweise befragt und die Er-

Dauer [min]	Aktivität
5	Begrüßung und Vorstellung des Vorgehens
10	Beschreibung der Modellstruktur
10	Erster Fragebogen: Wichtige Eigenschaften
30	Selektive Modellinspektion und zweiter Fragebogen
5	Dritter Fragebogen: Allgemeine Daten und Abschluss

Tabelle 5.2: Zeitlicher Ablauf einer Befragung

gebnisse entsprechend dokumentiert. Von den Teilnehmern sind sieben in der Softwareentwicklung bzw. Architektur tätig und fünf arbeiten als wissenschaftliche Mitarbeiter an den Technischen Universitäten in Darmstadt und München. Insbesondere die Teilnehmer aus dem industriellen Kontext können dabei eine langjährige Erfahrung im Bereich der Modellierung serviceorientierter Architekturen vorweisen, welche bei durchschnittlich sechs Jahren liegt. Die Teilnehmer aus dem akademischen Umfeld verfügen allesamt über einen Abschluss in Informatik. Drei von ihnen besitzen einen Doktorgrad, die beiden anderen arbeiten zurzeit an ihrer Dissertation. Alle Teilnehmer verfügen sowohl über theoretische als auch über praktische Erfahrung mit SOA und Qualitätssicherung, so dass grundsätzlich von einem homogenen Teilnehmerfeld ausgegangen werden kann. Die zuvor erwähnte Auswertung der eigenen Einschätzung der Erfahrung der Experten ergibt eine Standardabweichung von $\sigma \approx 1,084$, was gemäß der o. g. Definition allerdings knapp für Heterogenität spricht.

Während eines Interviews wurden folgende Hilfsmittel verwendet: Zur Einführung der Konzepte des Qualitätsmodells wurde eine schematische Darstellung verwendet (siehe Abbildung 4.3 auf Seite 62). Das SOA-Qualitätsmodell wurde in digitalisierter Form mit Hilfe des Quamoco-Qualitätsmodell-Editors vorgestellt. Als drittes wesentliches Element wurde ein Fragebogen verwendet. Dieser bestand aus drei Teilen: Der erste Teil enthielt Instrumente zur Angemessenheit des Qualitätsmodells, der zweite bezog sich auf die Vollständigkeit, und der dritte enthielt Fragen zum Hintergrund des Teilnehmers. Bedingt durch die oft knapp bemessene Zeit von erfahrenen Softwarearchitekten wurden die Befragungen auf den Umfang einer Stunde begrenzt, der sich wie in Tabelle 5.2 dargestellt auf einzelne Aktivitäten aufteilte.

Im ersten Teil der Befragung wurde offen nach Eigenschaften einer SOA gefragt, welche nach Meinung des Experten für die Qualität von Software von entscheidender Bedeutung sind. Zu jeder dieser Eigenschaften wurden die Experten nach einer kurzen Beschreibung gefragt, um synonym verwendete Begriffe im Rahmen der Analyse identifizieren und gruppieren zu können.

Den zweiten Teil der Befragung bildete die gezielte Inspektion des Qualitätsmodells bezüglich der zuvor genannten Eigenschaften. Jeder im ersten Teil erwähnte Punkt wurde aufgrund der zusätzlichen Beschreibung durch den Moderator auf einen Faktor des Qualitätsmodells abgebildet. Dieser wurde im Folgenden hinsichtlich der ihm zugeordneten Maße mit dem Experten analysiert und diskutiert. Im Falle von abs-

trakteren Konzepten wurden gegebenenfalls mehrere Faktoren betrachtet. Zu jedem Faktor wurden die Experten nach ihrer Einschätzung gefragt, inwieweit das Qualitätsmodell diesen angemessen abbildet. Außerdem konnten Ergänzungsvorschläge für weitere Maße gemacht werden, um den entsprechenden Faktor vollständiger quantifizieren zu können. Jedes der vorgeschlagenen Maße wurde durch die Experten bezüglich zweier Dimensionen eingeordnet: zum Einen bezüglich der *Offensichtlichkeit* des Zusammenhangs zwischen dem betrachteten Faktor und dem vorgeschlagenen Maß, zum Anderen bezüglich der *Wichtigkeit* des Maßes für den betrachteten Faktor. Zum Abschluss der Inspektion eines Faktors sollten die Experten einschätzen, inwiefern sie den betreffenden Faktor unter Einbeziehung der von ihnen vorgeschlagenen Ergänzungen als vollständig betrachten.

Da das Modell für eine umfassende Inspektion im Rahmen eines Interviews zu umfangreich ist, wurde in jedem Interview ein Ausschnitt des Modells untersucht. Generell unterscheidet man zwischen horizontalen und vertikalen Schnitten eines Qualitätsmodells. Ein horizontaler Schnitt liegt vor, wenn sämtliche Elemente einer Ebene untersucht werden, z. B. alle Maße des Modells oder alle Qualitätsziele. Im Gegensatz dazu beschreibt ein vertikaler Schnitt die Teilmenge eines Modells, die entlang eines Elements (z. B. eines Faktors) sämtliche Verbindungen zu anderen Elementen auf unterschiedlichen Ebenen betrachtet (also Qualitätsziele und Maße, welche mit diesem Faktor in Verbindung stehen).

Aufgrund der Tatsache, dass die befragten Experten jeweils besondere Schwerpunkte hinsichtlich ihrer Interessen und Kompetenzen besitzen und Vorstudien gezeigt haben, dass bei der Inspektion von Elementen aus dem jeweiligen Schwerpunkt detailliertere und belastbarere Antworten zu erwarten sind, wurde der vertikale Schnitt gewählt – aufgrund der unterschiedlichen Schwerpunkte allerdings nicht in allen Fällen entlang derselben Elemente, sondern pro Interview in Abhängigkeit der Angaben des jeweiligen Experten im ersten Teil der Befragung (siehe oben).

Nach Abschluss der Inspektion aller in diesem Schnitt enthaltenen Faktoren waren einige generelle Einschätzungen zum SOA-Qualitätsmodell gefragt. So sollten die Experten angeben, inwiefern sie im inspizierten Teil des Modells auf irrelevante Elemente gestoßen sind. Außerdem wurde nach Elementen gefragt, welche keinen spezifischen SOA-Bezug aufweisen. Zu jedem genannten Element wurde darüber hinaus eine Einordnung in folgende Kategorien erfragt:

Definition 5.1 (wünschenswert) *Ein Modellelement wird als* wünschenswert *bezeichnet, wenn es ein Konzept beschreibt, welches im Kontext von SOA eine entscheidende Bedeutung besitzt und nach Meinung des Experten auf jeden Fall in einem SOA-Qualitätsmodell enthalten sein sollte.*

Definition 5.2 (optional) *Ein Modellelement wird als* optional *bezeichnet, wenn es ein Konzept beschreibt, welches nicht ausschließlich für die Qualität von SOA (sondern etwa auch für Software allgemein) eine Rolle spielt, aber nach Meinung des Experten dennoch für die Beschreibung der Qualität von SOA relevant ist und deshalb durchaus in einem SOA-Qualitätsmodell enthalten sein sollte.*

Definition 5.3 (unangemessen) *Ein Element des Qualitätsmodells wird als* unangemessen *bezeichnet, wenn es ein Konzept beschreibt, welches für die Qualität von SOA nach Meinung des Experten keine Relevanz besitzt.*

Durch diese Aufteilung kann zwischen unspezifischen, aber im Kontext von SOA dennoch bedeutsamen Elementen und gänzlich irrelevanten unterschieden werden. Basierend auf diesen Kategorien kann Angemessenheit dadurch beschrieben werden, dass das Qualitätsmodell keine Elemente der Kategorie *Unangemessen* enthalten soll.

Der letzte Teil der Befragung zielte auf die Erfassung des Erfahrungshintergrunds der Experten im Bereich der SOA-Entwicklung, sowohl in zeitlicher Hinsicht als auch bezüglich ihrer Hauptaufgaben und ihrer Erfahrungen im Umgang mit Qualitätsmodellen. Eine detaillierte Aufstellung aller im Fragebogen enthaltenen Fragen ist in Tabelle 5.3 dargestellt.

5.2.2 Ergebnisse

Die Ergebnisse der Befragung sind im Folgenden ähnlich gegliedert wie der verwendete Fragebogen: Zunächst werden die Angaben zu Angemessenheit und Minimalität des Qualitätsmodells zusammengefasst, anschließend werden die Ergebnisse zur Vollständigkeit ausgewertet. Im dritten Unterabschnitt werden schließlich weitere Äußerungen der befragten Experten zusammenfassend präsentiert.

Angemessenheit und Minimalität

Fragen Q1.1 und Q1.2 wurden gestellt, um die in den Augen der Experten wichtigsten Eigenschaften einer SOA für deren Qualität sowie kurze Beschreibungen zu deren Definition in Erfahrung zu bringen. Diese Fragen waren als offene Fragen formuliert, so dass den Experten maximaler Spielraum bei der Beantwortung zur Verfügung stand. Um mögliche Überlappungen mit Antworten anderer Experten identifizieren zu können, wurde im Gespräch geklärt, was die genannten Eigenschaften für den jeweiligen Experten bedeuten. So konnten die am häufigsten genannten Eigenschaften wie folgt bestimmt werden: *Abstraktion* wurde als zentrale Eigenschaft einer SOA von neun der zwölf Befragten genannt, *Granularität* insgesamt fünfmal und *Komposition* viermal.

Zu jeder genannten Eigenschaft wurde mit Q2.1 gefragt, wie angemessen diese nach Meinung des Experten im Qualitätsmodell repräsentiert ist. Dazu wurde eine Likert-Skala von Eins (*sehr gut*) bis Fünf (*sehr schlecht*) verwendet. Die häufig genannten Eigenschaften wurden im Mittel allesamt mit Zwei (*gut*) oder besser bewertet. Die Ergebnisse sind in Tabelle 5.4 zusammengefasst.

Bezüglich der Minimalität des Modells wurden drei Fragen gestellt: Frage Q1.3 bot eine Likert-Skala, auf der die Experten bewerten sollten, inwiefern sie das Modell als frei von unnötigen Elementen ansehen. Hierbei stimmten alle Befragten (voll) zu. Darüber hinaus wurde in Q1.4 und Q1.5 gefragt, welche Elemente des Qualitätsmodells die Experten als nicht SOA-spezifisch ansehen bzw. für wie wichtig sie diese Elemente für das Modell erachten. Hier wurden die vier Bereiche *Namenskonventionen*, *Granularität*, *Kopplung* und *Kohäsion* genannt. Trotz ihrer SOA-unspezifischen Natur be-

ID	Frage	Typ
Q1.1	In your opinion, which are quality-related aspects that are important for SOA?	Offen
Q1.2	Please provide a short description for each aspect mentioned in Q1.1	Offen
Q1.3	Do you think the SOA quality model is free of unnecessary elements?	Likert
Q1.4	Which quality model elements are not SOA-specific in your opinion?	Offen
Q1.5	How necessary do you consider these elements for an SOA quality model?	Likert
Q2.1	Do you consider the SOA quality model appropriate regarding factor X?	Likert
Q2.2	Which elements should be added to the factor in order to increase completeness?	Offen
Q2.3	How obvious do you consider the mentioned elements?	Likert
Q2.4	How important do you consider the mentioned elements?	Likert
Q2.5	How complete do you consider the factor including your suggestions?	Likert
Q3.1	For how long have you been working in the SOA domain?	Offen
Q3.2	What is your main role/responsibility regarding SOA development?	Offen
Q3.3	What is the main SOA technology you are working with?	Offen
Q3.4	What is your experience with quality requirements in the SOA domain?	Likert
Q3.5	What is your experience with other quality models for SOA?	Likert
Q3.6	Were you able to understand the SOA quality model (with a reasonable amount of effort)?	Likert

Tabelle 5.3: Instrument zur Expertenbefragung

Eigenschaft	Anzahl	Mittelwert
Abstraktion	9	1,3
Granularität	5	1,7
Komposition	4	2,0

Tabelle 5.4: Angemessenheitsauswertung

werteten die entsprechenden Experten diese Elemente ausnahmslos als wichtig für die Qualität einer SOA. Zusammenfassend lässt sich schließen, dass das Qualitätsmodell nach Meinung der befragten Experten keine unnötigen Elemente enthält und damit das Minimalitätskriterium erfüllt.

Neben der Minimalität wurde auch die Verständlichkeit des Modells durch die Experten bewertet. Auf Frage Q3.6 stimmten sämtliche Befragten (voll) zu, dass sie in der Lage waren, das Qualitätsmodell mit angemessenem Aufwand zu verstehen. Für eine Diskussion dieses und weiterer Ergebnisse sei auf Abschnitt 5.2.3 verwiesen.

Vollständigkeit

Die Vollständigkeit des Qualitätsmodells wurde wie in Abschnitt 5.2.1 beschrieben mit einer angepassten Form des *Capture-Recapture*-Verfahrens bestimmt. Im Folgenden werden diejenigen Ausschnitte des Qualitätsmodells betrachtet, welche von den Experten als wichtig wahrgenommen wurden. Dies begründet sich darin, dass in der Befragung diejenigen Faktoren detaillierter betrachtet wurden, die vom jeweiligen Experten selbst genannt wurden. Die Anzahl der Experten, welche einen bestimmten Faktor inspiziert haben, ist also variabel. Briand u. a. [31] stellen fest, dass eine sinnvolle Anwendung des Schätzverfahrens ab einer Teilnehmerzahl von vier Experten möglich ist. Dies korrespondiert mit der Nennung eines Faktors durch ein Drittel der befragten Experten, so dass davon auszugehen ist, dass die hier betrachteten Faktoren die zentralen Elemente des Qualitätsmodells darstellen. Wie die Übersicht in Tabelle 5.6 auf Seite 124 zeigt, wurden die folgenden Faktoren von mindestens vier Experten inspiziert (Anzahl jeweils in Klammern): Fachliche Abstraktion (10), Standardisierung (6), Komposition (6), Funktionale Granularität (4).

Die Auswertung nach Homogenität oder Heterogenität der Experten wurde durch Frage Q3.4 operationalisiert, in der die Experten ihre Erfahrung mit Qualitätsanforderungen im Bereich SOA auf einer Skala von Eins (*keine Erfahrung*) bis Fünf (*sehr viel Erfahrung*) einschätzen sollten. Bei einer Standardabweichung von $\sigma < 1$ sollte von Homogenität ausgegangen werden, anderenfalls von Heterogenität. Die Befragung ergab einen Wert von $\sigma \approx 1,084$, was knapp für Heterogenität der Experten spricht. Um die fehlenden Elemente entsprechend zu klassifizieren, wurde die Offensichtlichkeit des Fehlens verwendet, was mit Frage Q2.3 operationalisiert wurde, in der die Experten selbst einschätzen sollten, wie offensichtlich der von Ihnen gemachte Vorschlag ist. Auch hier wurde als Grenze eine Standardabweichung von $\sigma < 1$ für Homogenität angenommen. Nach Auswertung der Ergebnisse ergibt sich ein Wert von $\sigma \approx 0,815$, so dass Homogenität angenommen werden sollte. Gemäß Tabelle 5.1 kommen also der *Maximum-Likelihood-* [114] und der *Chao*-Schätzer [38] in Frage, um eine ungefähre Anzahl der fehlenden Elemente im Modell zu ermitteln. Da die Standardabweichungen allerdings nicht besonders weit vom festgelegten Schwellenwert entfernt sind, wird zunächst der *Jackknife*-Schätzer angewendet, da dieser laut Petersson u. a. [120] für die Anwendung bei Inspektionen von Software besonders geeignet ist und auch von Mayr u. a. [98] für die Evaluation eines Qualitätsmodells zur Anwendung kam. Dieses Verfahren bezieht pro Vorschlag die Anzahl der Nennungen ein, so dass häufig genann-

Faktor	Vorschlag	Häufigkeit	Wichtigkeit
Fachliche Abstraktion	Datenanpassung	3	2,3
Fachliche Abstraktion	Datensemantik	2	4,0
Fachliche Abstraktion	Einhaltung der Ebenen	2	2,0
Fachliche Abstraktion	Konsistenz	2	2,0
Fachliche Abstraktion	Datenmodell	1	2,0
Fachliche Abstraktion	Dienste pro Prozess	1	3,0
Standardisierung	Datenmodell	2	3,0
Standardisierung	Datensemantik	2	1,0
Standardisierung	Datenanpassung	1	1,5
Standardisierung	Standardisiertes Verhalten	1	1,0
Komposition	Einhaltung der Ebenen	2	1,0
Komposition	Konsistenz	2	1,0
Funktionale Granularität	Bezug zur Spezifikation	1	3,0

Tabelle 5.5: Vollständigkeitsauswertung – Vorschläge

te Elemente einen anderen Einfluss besitzen als selten genannte. Je nach maximaler Anzahl der Nennungen berechnen sich die Schätzwerte $\hat{N}$ für die im Modell fehlenden Elemente folgendermaßen:

$$
\begin{aligned}
\hat{N}_{\mathrm{J1}} &= D + \frac{t-1}{t} \cdot f_1 \\
\hat{N}_{\mathrm{J2}} &= D + \frac{2t-3}{t} \cdot f_1 - \frac{(t-2)^2}{t(t-1)} \cdot f_2 \\
\hat{N}_{\mathrm{J3}} &= D + \frac{3t-6}{t} \cdot f_1 - \frac{3t^2-15t+19}{t(t-1)} \cdot f_2 + \frac{(t-3)^3}{t(t-1)(t-2)} \cdot f_3
\end{aligned}
$$

Dabei bezeichnet f_i die Anzahl der Vorschläge, die genau i-mal gemacht wurden. D entspricht der Anzahl der unterschiedlichen Vorschläge zu einem Faktor und t der Anzahl der Experten, die einen Faktor inspiziert haben. Die nach diesen Formeln ermittelten Werte sind Tabelle 5.6 zu finden.

Die Werte für f_i lassen sich dabei folgendermaßen aus Tabelle 5.5 herleiten: Für den Faktor *Fachliche Abstraktion* wurde der Punkt *Datenanpassung* als einziger dreimal vorgeschlagen, so dass f_3 für diesen Faktor gleich Eins ist. Außerdem wurden drei Vorschläge von jeweils zwei Experten gemacht und zwei weitere von jeweils einem Experten, woraus $f_2 = 3$ und $f_1 = 2$ folgen. D entspricht der Summe der Vorschläge, also $1 + 3 + 2 = 6$, was mit der Anzahl der Zeilen zu diesem Faktor in Tabelle 5.5 übereinstimmt. Den Faktor *Fachliche Abstraktion* wurde also von zehn Experten inspiziert, die insgesamt sechs unterschiedliche Vorschläge für zusätzliche Maße eingebracht

Faktor	f_1	f_2	f_3	D	t	m	J1	J2	J3
Fachl. Abstraktion	2	3	1	6	10	3	7,800	7,267	5,643
Standardisierung	2	2	0	4	6	2	5,667	5,933	5,533
Komposition	0	2	0	2	6	3	2,000	0,933	-0,467
Funkt. Granularität	1	0	0	1	4	2	1,750	2,250	2,500

Tabelle 5.6: Vollständigkeitsauswertung – Schätzung

haben. Die Spalte m enthält die Angabe, wie viele Maße einem Faktor im Qualitätsmodell zugeordnet sind – am Beispiel der fachlichen Abstraktion sind dies drei Maße. Basierend auf diesen Angaben ermittelt das *Jackknife*-Verfahren einen Schätzwert für die Anzahl der Maße, die dem Faktor hinzugefügt werden müssten, bis er (basierend auf dem Verständnis der befragten Experten) als vollständig zu betrachten ist. Je nach Parametrisierung des Verfahrens bedeutet dies für die fachliche Abstraktion die Hinzunahme weiterer sechs bis acht Maße. Für die Einordnung und Diskussion dieser Ergebnisse sei auf den folgenden Abschnitt 5.2.3 verwiesen.

Frei formulierte Einschätzungen

Im Rahmen der Befragungen wurden unterschiedliche Perspektiven der Experten zur Bedeutung von Qualität in SOA-Systemen deutlich. Ein klarer Schwerpunkt lag dabei auf der Kundensicht. Ein Experte sagte: „Qualität in SOA bedeutet, einen Geschäftsprozess zufriedenstellend abzudecken.“ Auch konkrete Faktoren (z. B. die Datengranularität einer Schnittstelle) seien immer im fachlichen Zusammenhang aus Kundensicht zu bewerten: „Die richtige Datengranularität hängt häufig von der Semantik der Daten ab.“ Generell sei der Kunde das Maß aller Dinge – insbesondere müsse die Software aber dessen Anforderungen auch nicht übererfüllen: „Qualität hängt vom Anspruch des Kunden ab. Die Software muss gut *genug* sein.“ Ein Experte ging sogar so weit festzustellen, die *interne* Qualität einer Software sei „im Prinzip egal“, da sie vom Kunden nicht unmittelbar wahrgenommen werde. Auch eine aktivitätenbasierte Sicht auf Qualität war vertreten, was in der Aussage „Qualität drückt sich in Kosten für Einarbeitung, Implementierung, Verteilung, Wartung und Anpassung aus“ deutlich wird.

Bei der Beschreibung der für Qualität wichtigen Eigenschaften einer SOA wurden einige Probleme in Bezug auf die Erstellung und Qualitätssicherung von SOA-Systemen deutlich. So wurde unter anderem der Bereich SOA-Governance als aufwändig (und damit teuer) beschrieben. Automatisierung sei zwar in Teilbereichen möglich, etwa bei der Überprüfung von Namenskonventionen, aber häufig nicht implementiert. Darüber hinaus sei „kein Bewertungsmodell für SOA vorhanden“. Wichtige Bereiche der Qualitätsbewertung ließen sich darüber hinaus nicht automatisiert durchführen: „Das größte Qualitätsproblem ist die Semantik – und Semantikprobleme sind teure Probleme: Es muss eine manuelle Bewertung durch Kunden oder Berater erfolgen, die die

Anwendung kennen.“ Als weiteres Problem sahen einige Experten, dass die vollständige Erfüllung sämtlicher theoretischer Architekturprinzipien im Einzelfall nicht die beste Lösung sein müsse. „Man startet oft mit den puren Prinzipien,“ führte einer der Experten aus. Ein anderer stellt fest: „Neue Systeme kann man ordentlich in SOA bauen“, und ergänzt im Verlauf der Befragung: „Es gibt nur sehr wenige *richtige* SOAs in der Praxis – und das sind häufig Beispielanwendungen.“ Als Gründe hierfür wurden verschiedene Umstände genannt: Einerseits wollen Kunden oft ihre bestehenden Systeme behalten, welche nicht serviceorientiert seien. Häufig müsse aber auch zwischen konkurrierenden Anforderungen abgewogen werden: „Man wägt immer Kosten gegen Nutzen ab, und *sauber* ist nicht immer der beste Weg.“

Generell wurden allerdings sowohl die grundsätzliche Idee eines SOA-Qualitätsmodells als auch dessen Inhalt äußerst positiv bewertet. Die befragten Experten äußerten etwa: „Das Modell ist gut strukturiert und verständlich.“ Bei der Verwendung abstrakter Qualitätskategorien wie *Maintainability* sei die Bedeutung dieser Begriffe eine essentielle Voraussetzung für effektives Qualitätsmanagement. Das Qualitätsmodell leiste hier durch seine klare Struktur einen entscheidenden Beitrag: „Transparenz und Objektivierbarkeit dieser Definitionen profitieren vom Qualitätsmodell.“ Vor allem die formalisierte Dekomposition sei ein wichtiger Faktor, so ein Experte: „Die Bewertung der Konformität [zu den SOA-Prinzipien] ist schwierig, weil deren Dekomposition oft willkürlich geschieht.“ Insbesondere die Möglichkeit, mit Hilfe des Qualitätsmodells Architekturbewertungen durchzuführen, stieß in den Befragungen auf großes Interesse: „Der Ansatz der automatischen Architekturanalyse ist gut.“ – „Der Ansatz könnte zur Analyse unserer Service-Schnittstellen spannend sein.“ Es kam allerdings auch zu einigen kritischen Anmerkungen, insbesondere bezüglich der praktischen Anwendung des Modells. Manche Probleme seien allerdings allgemein für Standards zu beobachten und nicht auf das SOA-Qualitätsmodell beschränkt. So zeige sich der „Nutzen eines Standards in seiner Verbindlichkeit, seinem Aufwand zur Anwendung und dem Verhältnis von darin abgebildeten Kundenanforderungen gegenüber internen Dimensionen.“

Konkret wurden von den Experten die folgenden Grenzen des vorgestellten Ansatzes identifiziert: Unter anderem seien manche Qualitätsdimensionen allein auf Grundlage der im Qualitätsmodell abgebildeten Faktoren nicht analysierbar: „Performance und Scalability lassen sich nicht vom Interface-Modell ableiten. Man braucht zusätzlich Kenntnis über das Laufzeit-Framework.“ Außerdem sei es nicht in jeder Umgebung möglich, die modellierten Faktoren vollständig zu erfüllen, denn „manche Eigenschaften der Architektur sind vom Framework vorgegeben“. Hier könnte eine Konkretisierung des Qualitätsmodells bezüglich dieser Vorgaben unterschiedlicher Frameworks sowohl eine Einschätzung über die Eignung von Frameworks ermöglichen als auch die Qualitätsanalyse einer Architektur dahingehend vereinfachen, dass aufgrund dieser Vorgaben nicht zu beeinflussende Maße entsprechend vorbelegt werden. Bezüglich der verwendeten Gewichtung von Einflüssen zur Bewertung von Faktoren wurde der Einwand erhoben, es sei sehr schwierig zu quantifizieren, wie relevant einzelne Faktoren für die Bewertung eines abstrakteren Faktors seien. Auch die Wertebereiche der verwendeten Metriken, insbesondere die für die Faktorbewertung verwendeten Grenzwerte, seien „sehr schwer global zu definieren und müssen von Fall zu Fall entschieden

werden“. Darüber hinaus seien viele der modellierten Faktoren nicht unabhängig, was die Bewertung zusätzlich erschwere.

Was die Anwendung des Qualitätsmodells zur Bewertung von Architekturen betrifft, wurde angemerkt, eine wichtige Voraussetzung für jegliche Qualitätsbewertung sei, dass „der Anspruch – also die Vorgaben – bekannt und formuliert sind.“ Die Transparenz der bewerteten Eigenschaften einer Architektur könne mit dem Qualitätsmodell erreicht werden, so ein Experte. Generell sei die unmittelbare Anwendung eines wissenschaftlich konstruierten Qualitätsmodells für SOA im industriellen Umfeld ein ambitioniertes Ziel, denn „die akademische Betrachtung [von SOA] unterscheidet sich grundlegend von der Praxis“. Dies sei vor allem darin begründet, dass akademische Arbeiten vorrangig dem wissenschaftlichen Erkenntnisgewinn dienten, während ein praktischer Einsatz des Qualitätsmodells (z. B. im Beratungswesen) darüber hinaus die Skalierbarkeit und Nutzerfreundlichkeit der verwendeten Werkzeuge sowie die Ableitung von Handlungsempfehlungen und zu erwartende Kosteneinsparungen erfordere. Ein weiterer Experte schätzte die im Qualitätsmodell verwendete Terminologie als zu generisch ein: „Für einen konkreten Nutzen im Projekt muss man spezifische Entitäten einführen, z. B. Business-Objekte“. Der Ansatz, die Architekturanalyse mit Werkzeugen durchzuführen, welche auch für die Analyse von Sourcecode verwendet werden können, wurde von den Experten als besonders positiv empfunden: „Eine Analyse über alle Ebenen hinweg ist am effektivsten.“ Neben diesen grundsätzlichen Anmerkungen zur praktischen Anwendung des Modells machten die Experten konkrete Vorschläge zur weitergehenden Nutzung. So könne man anhand der Architekturmerkmale neben der Qualitätsanalyse auch Abschätzungen des zu erwartenden Daten- und Transfervolumens, des Entwicklungs- und Supportaufwandes sowie des benötigten Umfangs von Regressionstests durchführen. Zudem wurde zur Einführung der modellbasierten SOA-Qualitätsanalyse vorgeschlagen, das Qualitätsmodell exemplarisch für ein Projekt in den Entwicklungsprozess zu integrieren und die dadurch verursachten Extrakosten gegen den generierten Mehrwert für den Kunden abzuwägen.

Die Befragungen führten auch zu Kommentaren bezüglich einzelner Inhalte des Qualitätsmodells: So sei etwa die Anzahl indirekter Abhängigkeiten eines Dienstes oft nicht nur ein Hinweis auf mögliche Probleme bezüglich der Antwortzeit und Zuverlässigkeit, sondern auch „ein Indikator für sehr spezielle Services“. Vor allem die Granularität von Diensten war häufig Anlass für Kommentare: Hier zeigte sich vor allem, dass die Experten Wert darauf legen, dass Dienste nicht zu generisch sind: „Wiederverwendung muss Use-Case-spezifisch verstanden werden: General-Purpose-Services sind nicht zielführend,“ denn „sehr breite und generische Interfaces bedeuten Aufwand bei der Anpassung.“ Das Konzept, Wiederverwendung auf verschiedenen Abstraktionsebenen zu nutzen, veranlasste einen Experten zu der Aussage: „Wiederverwendbarkeit im feingranularen Sinn ist einfacher zu messen, aber die Wiederverwendung abstrakterer Dienste ist effektiver und sollte stärker gewichtet werden.“ Bei allen Anpassungen müssten bestehende „Interfaces nach außen stabil“ bleiben, um Funktionalität für die Nutzer zu erhalten. Dabei sollten allerdings auch keine Kopien bestehender Dienste erzeugt und anschließend erweitert werden: „Redundanzen in der Architektur sind un-

bedingt zu vermeiden, sowohl bezüglich Daten als auch bezüglich Modellinstanzen und Code.“

Generell sei SOA eine Möglichkeit, durch Standardisierung und Abstraktion Software leichter wartbar und erweiterbar zu gestalten. Dies gelte allerdings nicht uneingeschränkt: „Es kommt immer auf den Use Case an, ob SOA die geeignete Wahl ist“.

5.2.3 Diskussion

Im Folgenden wird die Expertenbefragung diskutiert und kritisch gewürdigt. Dies gilt einerseits für die von den Experten getroffenen Aussagen und Einschätzungen, andererseits für das gewählte Vorgehen und daraus resultierende Stärken und Schwächen des Ansatzes.

Ergebnisse

Insgesamt profitierte das Qualitätsmodell von den Einschätzungen der Experten. Viele Antworten auf die freien Fragen ähnelten den in den Vorstudien getroffenen Aussagen (siehe Voelz und Göb [9]), obwohl teils unterschiedliche Experten befragt wurden. Mit dem Qualitätsmodell als Diskussionsgrundlage konnten diese Einschätzungen jedoch weiter konkretisiert und die Interpretation der damaligen Aussagen validiert werden. Darüber hinaus haben die Befragungsergebnisse gezeigt, dass das Modell in den zentralen Punkten eine Vollständigkeit aufweist, welche eine sinnvolle, grundlegende Bewertung SOA-basierter Systeme erlauben sollte.

Es ist zu beobachten, dass die geschätzten Werte für fehlende Elemente in vielen Fällen deutlich über der Anzahl der im Modell enthaltenen Maße liegt, also der Schluss nahe liegt, dass das Modell hochgradig unvollständig sei. Dem stehen allerdings die eigenen Einschätzungen der Experten gegenüber, die für die entsprechenden Faktoren angegeben haben, diese seien sehr gut bis gut im Qualitätsmodell abgedeckt (siehe z. B. die Angaben zur Angemessenheit in Tabelle 5.4 auf Seite 121). Darüber hinaus wurde zu jedem vorgeschlagenen Element die Wichtigkeit abgefragt, welche in einigen Fällen ebenfalls nahelegt, dass das entsprechende Element zwar einen Mehrwert bezüglich der Vollständigkeit bieten würde, allerdings nicht als besonders wichtig eingeschätzt wurde. So wurde zwar eine global festgelegte Datensemantik von einem Experten als sinnvolle Ergänzung für den Faktor *Fachliche Abstraktion* angegeben, aber auf einer Skala von Eins (*sehr wichtig*) bis Fünf (*unwichtig*) lediglich mit Vier bewertet.

Neben diesen inhaltlichen Vorschlägen konnten die Experten diverse kleinere Schwächen und Inkonsistenzen des Modells identifizieren, welche unmittelbar behoben werden konnten, was die Qualität des Modells positiv beeinflusste. Zu solchen direkt durchgeführten Korrekturen gehörten etwa das Hinzufügen zusätzlich identifizierter Einflüsse von bestehenden Faktoren auf bestehende Qualitätsziele oder die Umkehrung irrtümlich falsch gesetzter Effekte von Einflüssen (positiv statt negativ). Insofern wurde die Inspektion des Qualitätsmodells neben dem adaptierten *Capture-Recapture-*

Verfahren zur Vollständigkeitsschätzung auch erfolgreich im klassischen Sinne zum Auffinden von Fehlern genutzt.

Die Hinweise einiger Experten, dass es grundsätzlich schwierig sei, verschiedene Einflüsse auf ein Qualitätskriterium gegeneinander abzuwägen, erschienen gerechtfertigt. Bereits die Erstellung des Quamoco-Basismodells hatte gezeigt, dass die manuelle Vorgabe von Gewichten allein nicht zu einem praktisch anwendbaren Qualitätsbewertungsmodell führt. Vielmehr muss das Qualitätsmodell auf eine hinreichend große Anzahl geeigneter Softwaresysteme angewendet werden, um ein Bild von den tatsächlich auftretenden Messwerten zu erhalten und das Qualitätsmodell anschließend kalibrieren zu können. Dieses Thema wird in Abschnitt 6.2 aufgegriffen und näher erläutert.

Eine Aussage aus der Expertenbefragung ist besonders hervorzuheben, obwohl sie nicht unmittelbar das Qualitätsmodell betrifft: „Es kommt immer auf den Use Case an, ob SOA die geeignete Wahl ist.“ Hier zeigt sich die Abhängigkeit der Architekturentscheidung vom konkreten Anwendungsfall. Das Qualitätsmodell kann an dieser Stelle helfen, die Auswirkungen der Entscheidung für SOA als Architekturstil auf die Qualität der Software explizit zu machen und somit Softwarearchitekten eine Hilfestellung zu dieser Entscheidung zu geben.

Vorgehen

Das Vorgehen persönliche Interviews zu führen ist im Nachhinein positiv zu bewerten. Insbesondere aufgrund der zeitlich begrenzten Verfügbarkeit der befragten Experten war es nur so möglich, das Qualitätsmodell individuell auf den jeweiligen Experten angepasst zu erläutern und mögliche Rückfragen schnell zu beantworten. Dies war vor allem deshalb notwendig, da die Experten zwar allesamt langjährige Erfahrung in der Entwicklung serviceorientierter Architekturen und der Erfüllung von Qualitätsanforderungen an diese Architekturen vorweisen konnten, allerdings nicht über Vorwissen zum gewählten Modellierungsansatz und der Navigation innerhalb des Modells mit Hilfe des Qualitätsmodell-Editors verfügten. Ein weiterer Vorteil dieser persönlichen Befragung im Gegensatz zu einem rein Fragebogen-basierten Ansatz ist die Möglichkeit, bei unklaren Antworten gezielt nachzufragen, um so die Eindeutigkeit der Antworten und die Vergleichbarkeit der Angaben unterschiedlicher Experten sicherzustellen, etwa indem synonym verwendete Begriffe unmittelbar identifiziert werden konnten.

Durch den Umfang des Modells von über 100 Elementen war es nicht möglich, innerhalb einer Befragung das gesamte Qualitätsmodell inspizieren zu lassen; vielmehr wurden jeweils Ausschnitte des Modells basierend auf den Präferenzen der Experten gewählt. Die Befragungsergebnisse für diese Ausschnitte wurden gemäß dem gewählten *Capture-Recapture*-Ansatz zu einer Aussage über das gesamte Qualitätsmodell generalisiert. Wie jede Verallgemeinerung birgt dieses Vorgehen ein gewisses Fehlerrisiko, vor allem für den Fall, dass die gewählten Ausschnitte nicht repräsentativ für das gesamte Modell sind. Dieses Risiko steht der zuvor erwähnten Qualität der Antworten durch selektive Auswahl geeigneter Modellausschnitte entgegen und hätte nur durch

eine extrem große Anzahl befragter Experten minimiert werden können, was an der praktischen Durchführbarkeit scheiterte.

Insgesamt hilft der gewählte *Capture-Recapture*-Ansatz dabei, Einschätzungen zum Qualitätsmodell zu bekommen, bevor dieses im Rahmen einer groß angelegten, automatisierten Bewertung einer Vielzahl von Systemen kalibriert (siehe Abschnitt 6.2 auf Seite 146) und aufgrund der Bewertungsergebnisse evaluiert werden kann. Trotzdem ist zu beachten, dass es sich bei den ermittelten Werten für Angemessenheit und Vollständigkeit des Modells lediglich um Schätzwerte handelt, welche nicht in vollem Umfang den im praktischen Einsatz des Qualitätsmodells wahrgenommenen Werten entsprechen müssen.

Bezüglich der Anzahl befragter Experten sei insbesondere auf die Arbeiten von Vegas und Basili [137] sowie Briand u. a. [31] verwiesen. Vegas und Basili führen das Argument an, dass für die Vervollständigung eines Modells dann eine ausreichende Anzahl von Experten erreicht sei, wenn mit großer Wahrscheinlichkeit davon auszugehen ist, dass bei Befragung weiterer Experten keine neuen Vorschläge zum Hinzufügen weiterer Elemente in das Modell zu erwarten sind. Das dort evaluierte Modell wurde zwischen den Befragungen jeweils um die vorgeschlagenen Elemente erweitert, so dass nach jeder Befragung die absolute Anzahl neuer Ergänzungsvorschläge als Hinweis für das Erreichen einer ausreichenden Anzahl an Experten herangezogen werden konnte. Für das SOA-Qualitätsmodell wurde ein leicht unterschiedlicher Ansatz verfolgt, gemäß dem das Modell über die Dauer der Befragungen bis auf Fehlerkorrekturen unverändert blieb. So konnte bei einem zur Aufnahme vorgeschlagenen Element zusätzlich untersucht werden, wie viele der Experten den gleichen Vorschlag machen, was wiederum auf die Wichtigkeit oder Allgemeingültigkeit dieses Elements schließen ließ. Diese Information wäre nicht verfügbar gewesen, wenn man das Qualitätsmodell nach jeder Befragung um die vorgeschlagenen Elemente ergänzt hätte. Unter Einbeziehung dieser zusätzlichen Bedingung erscheint es sinnvoll, das Abbruchkriterium dahingehend anzupassen, dass von einem weiteren Experten keine Vorschläge zu erwarten wären, die nicht schon in vorherigen Befragungen genannt wurden. Für das verwendete *Capture-Recapture*-Verfahren führen Briand u. a. [31] eine erforderliche Anzahl von vier bis fünf Experten an. Da in diesem Fall der betrachtete Ausschnitt als konstant vorausgesetzt wurde, muss man in der hier durchgeführten Befragung die Anzahl der Experten dahingehend erhöhen, dass mindestens vier bis fünf Experten *denselben* Ausschnitt des Modells, d. h. dieselben Faktoren inspiziert haben. Für die im Detail betrachteten Faktoren *Fachliche Abstraktion*, *Standardisierung*, *Komposition* und *Funktionale Granularität* ist dies der Fall, wie Tabelle 5.6 (Spalte t) zeigt.

Weiterhin ist anzumerken, dass die Einführung in die Struktur des Qualitätsmodells mit zehn Minuten knapp bemessen war (siehe Tabelle 5.2 auf Seite 118). Innerhalb dieser Zeit ist nicht davon auszugehen, dass ein SOA-Experte sämtliche Modellierungskonzepte inklusive deren Repräsentation im Editor komplett erfasst und zur eigenständigen, effizienten Inspektion des Qualitätsmodells in der Lage ist. Die Durchführung der Befragung im Rahmen persönlicher Treffen und die Hilfestellung bei der Lokalisierung von Modellelementen konnten dieser Tatsache entgegenwirken. Einerseits birgt dieses Vorgehen zwar das Risiko, dass das Qualitätsmodell als intuitiver verständlich

wahrgenommen und bewertet wird, als es bei einer alleinigen Inspektion des Modells der Fall gewesen wäre. Dies ist vor allem bei der Bewertung der Tatsache zu beachten, dass sämtliche Experten der Aussage zustimmen, das Modell verstanden zu haben. Andererseits wäre eine eigenständige Inspektion implizit durch die Verständlichkeit und Benutzerfreundlichkeit des Editors beeinflusst worden und hätte daher ebenfalls keine verlässliche Aussage über die Verständlichkeit des Modells an sich liefern können. Insofern stellt das gewählte Vorgehen einen Kompromiss zwischen der zur Verfügung stehenden (Vorbereitungs-) Zeit pro Befragung und der Verlässlichkeit der gewonnenen Ergebnisse dar, wobei erstere wie zuvor erwähnt von außen limitiert war.

5.3 Anwendung des Qualitätsmodells

Im Folgenden wird das Qualitätsmodell für SOA in unterschiedlicher Art und Weise angewendet, um die eingangs formulierten Forschungsfragen zu beantworten. Die ausgewählten Szenarien orientieren sich an der Arbeit von Deißenböck u. a. [47], in der zwischen Qualitätsmodellen zur Definition, Bewertung und Vorhersage von Qualität unterschieden wird, welche in dieser Reihenfolge aufeinander aufbauen sollten. Gemäß dieser Klassifikation kann das SOA-Qualitätsmodell als Definitions- und Bewertungsmodell angesehen werden, so dass auch Anwendungsmöglichkeiten aus diesen beiden Bereichen in Frage kommen. Durch die Beschreibung der Faktoren und Maße im Modell wird ein gemeinsames Verständnis bezüglich Qualitätsanforderungen an SOA-Systeme gefördert und Wissen über die Eigenschaften von SOA-Systemen, welche deren Qualität beeinflussen, vermittelbar gemacht. Um dieses Anwendungsgebiet des Qualitätsmodells zu adressieren, wird in Abschnitt 5.3.1 basierend auf dem Modell untersucht, inwieweit Qualitätsaspekte, die SOA zugeschrieben werden, tatsächlich unmittelbar aus der Erfüllung der SOA-Prinzipien resultieren oder darüber hinaus besondere Anforderungen an das Softwaredesign stellen.

Das SOA-Qualitätsmodell wurde allerdings nicht allein als Definitionsmodell konzipiert, sondern ist auch zur Qualitätsbewertung von SOA-basierten Systemen geeignet. Deißenböck u. a. [47] erwähnen dabei explizit die zentrale Rolle des Qualitätsmodells als Basis für sämtliche Messungen sowie die automatische Generierung von Richtlinien für manuelle Inspektionen. In Abschnitt 5.3.2 wird das Qualitätsmodell exemplarisch zur Bewertung einer SOA verwendet, die im Rahmen eines Forschungsprojekts entstanden ist.

5.3.1 Qualitätseigenschaften der SOA-Prinzipien

Im Folgenden wird erörtert, inwieweit SOA die Erwartungen erfüllt, die oft damit verbunden werden. Dazu bedient sich dieser Abschnitt einerseits der Struktur von Abschnitt 2.4.2, der diese Argumente und Erwartungen einführt und erläutert. Mit Hilfe des Qualitätsmodells, insbesondere des Moduls zur Konformität, wird für jeden angeführten Qualitätsaspekt überprüft, inwieweit er allein durch Konformität zu den SOA-Prinzipien sichergestellt werden kann. Ist dies der Fall, folgt daraus, dass eine Software, die in Einklang mit den SOA-Prinzipien implementiert wurde, bereits den

Qualitätsziel	Konformität	Design
Z1: [Auffinden | EFFIZIENZ]		F10
Z2: [Externe Analyse | EFFIZIENZ]	F5	F9–F11, F15, F16, F18, F19
Z3: [Konsum | EFFEKTIVITÄT]	F1, F2	F18, F19
Z4: [Konsum | EFFIZIENZ]		F22
Z5: [Konsum | KONTINUITÄT]	F8	F20, F22
Z6: [Betrieb | EFFIZIENZ]		F13, F21
Z7: [Betrieb | KONTINUITÄT]	F8	F20
Z8: [Interne Analyse | EFFIZIENZ]	F8	F9–F15, F19
Z9: [Anpassung | EFFIZIENZ]	F5, F8	F16, F18
Z10: [Test | EFFIZIENZ]	F6, F7	
Z11: [Komposition | EFFIZIENZ]	F1, F2, F5–F7	F9, F14–F19
Z12: [Komposition | EFFEKTIVITÄT]	F3, F4, F6, F7	

Tabelle 5.7: Einflüsse gruppiert nach Qualitätszielen

entsprechenden Qualitätsaspekt erfüllt. Andererseits werden alle im Modell enthaltenen Qualitätsziele im Kontext der auf sie entfallenden Einflüsse betrachtet. Tabelle 5.7 listet in der ersten Spalte die Qualitätsziele auf, gefolgt von den Einflüssen aus dem Konformitätsmodul in der zweiten und den Einflüssen aus dem Designmodul in der dritten Spalte.

Besonders zu beachten sind hier neben den erwähnten, oft angeführten Qualitätsversprechen diejenigen Qualitätsziele, die jeweils nur von Faktoren aus einem der beiden Module beeinflusst werden – dort liegt nämlich entweder eine alleinige Beeinflussung durch SOA vor oder es konnte kein direkter Einfluss von SOA nachgewiesen werden. Die nachfolgenden Abschnitte unterziehen diese Qualitätsziele nacheinander einer Betrachtung hinsichtlich der auf sie entfallenden Einflüsse und bewerten den Beitrag von SOA zur Erreichung dieser Ziele.

Interoperabilität

Eines der Ziele, das mit SOA verfolgt wird, ist die verbesserte Interoperabilität zwischen Systemen. Wie bereits in Abschnitt 4.4.3 beschrieben wurde, wird der Informationsaustausch zwischen Diensten durch Komposition abgebildet, so dass Interoperabilität durch die Effektivität von Komposition, also Z12: [Komposition | EFFEKTIVITÄT], ausgedrückt werden kann. Abbildung 5.1 zeigt dieses Qualitätsziel zusammen mit sämtlichen im Qualitätsmodell enthaltenen Faktoren, die einen Einfluss auf dieses Ziel beschreiben.

Die Möglichkeit Dienste verteilt zu nutzen ist gewissermaßen eine notwendige Bedingung für deren Interoperabilität. Eine weitere technische Grundlage ist die Kompatibilität, also die Standardisierung von Schnittstellenbeschreibungen, welche möglichst vollständig von Implementierungsdetails abstrahieren, so dass die Verwendung unter-

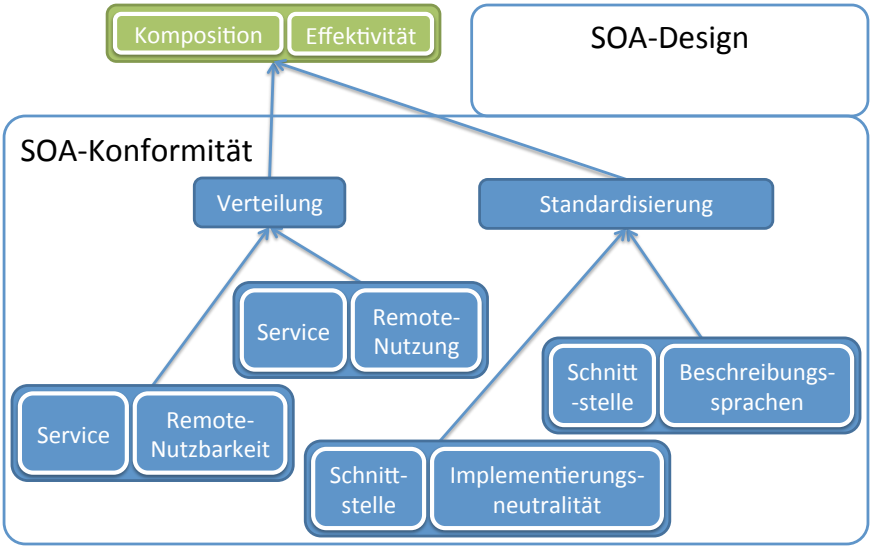

Abbildung 5.1: Einflüsse auf Interoperabilität

schiedlicher Technologien und insbesondere Programmiersprachen möglich wird. Diese technische Voraussetzung spiegelt sich im Qualitätsmodell als Operationalisierung des SOA-Prinzips *Standardisierung* wider. Das Fehlen von Einflüssen aus dem Bereich der SOA-Designfaktoren auf die Interoperabilität legt den Schluss nahe, Interoperabilität könne allein durch Festlegung syntaktischer Schnittstellenstandards gewährleistet werden. Dies ist jedoch im Allgemeinen nicht der Fall. Das zentrale Problem der Interoperabilität findet sich nicht auf der syntaktischen, sondern vielmehr auf der semantischen Ebene: Sowohl in der Literatur (z. B. [43, 113]) als auch in der industriellen Praxis (siehe Abschnitt 5.2) ist die Ansicht vertreten, dass erst durch Weiterentwicklung und Umsetzung semantischer Technologien das Interoperabilitätsproblem heutiger Softwaresysteme gelöst werden kann.

Der Grund für diese Diskrepanz zwischen Modell und Realität liegt in der spezifischen Zielsetzung des Modells, ausschließlich technische und strukturelle Eigenschaften der Architektur als Einflussfaktoren zu enthalten. Modelle bilden grundsätzlich nur einen Teil der Realität ab, so dass sich immer Bereiche identifizieren lassen, in denen ein Modell nicht zur Erklärung bestimmter Phänomene der Realität geeignet ist. Eine abschließende Antwort auf die Frage, welche Eigenschaften eine SOA außer implementierungsunabhängigen Schnittstellenstandards besitzen muss um ein größtmögliches Maß an Interoperabilität zu gewährleisten, wird im Rahmen dieser Dissertation nicht gegeben, da vor allem die semantische Interoperabilität zur Zeit noch ein ungelöstes, wenn auch sehr aktiv erforschtes Gebiet darstellt. Hinweise darauf, welche konkreten Einflüsse komponentenbasierte Architekturen auf Interoperabilität besitzen, finden sich unter anderem in den Arbeiten von Davis, Gamble und Payton [44] oder McArthur, Saiedian und Zand [99].

Ziel	Prinzip/Faktor	Modul
Z9	Geschäftsprozessbezug und Fachlichkeit	Konformität
	Lose Kopplung und dynamische Dienstkomposition	Konformität
	[Service | FUNKTIONALE GRANULARITÄT]	Design
	[Service | DATENGRANULARITÄT]	Design
Z11	Verschachtelte Wiederverwendung und Komposition	Konformität
	Geschäftsprozessbezug und Fachlichkeit	Konformität
	Standardisierte Formate und Schnittstellen	Konformität
	[Schnittstelle | BEDEUTSAME BESCHREIBUNG]	Design
	[System | SCHNITTSTELLENKOMPLEXITÄT]	Design
	[Schnittstelle | KOMPLEXITÄT]	Design
	[Service | FUNKTIONALE GRANULARITÄT]	Design
	[Operation | FUNKTIONALE GRANULARITÄT]	Design
	[Service | DATENGRANULARITÄT]	Design
	[Schnittstelle | KOHÄSION]	Design
Z8	Lose Kopplung und dynamische Dienstkomposition	Konformität
	[Schnittstelle | BEDEUTSAME BESCHREIBUNG]	Design
	[Service | ANGEMESSENE BENENNUNG]	Design
	[Operation | ANGEMESSENE BEZEICHNER]	Design
	[System | DESIGN-GRÖßE]	Design
	[System | FRAGMENTIERUNG]	Design
	[System | SCHNITTSTELLENKOMPLEXITÄT]	Design
	[Schnittstelle | KOMPLEXITÄT]	Design
	[Schnittstelle | KOHÄSION]	Design

Tabelle 5.8: Einflüsse auf die Flexibilität

Flexibilität

SOA wird häufig die Möglichkeit zugesprochen, vergleichsweise einfach an Änderungen in Geschäftsprozessen anpassbar zu sein, und damit flexibel auf geänderte Rahmenbedingungen reagieren zu können. In der aktivitätenbasierten Darstellung lässt sich deshalb diese Flexibilität durch die Effizienz der Anpassung des Systems ausdrücken: Z9: [Anpassung | EFFIZIENZ]. Der Begriff Flexibilität schließt auch Szenarien ein, in denen das System nicht angepasst, sondern erweitert wird. Die Erweiterbarkeit eines SOA-basierten Systems wird wie in Abschnitt 4.4.3 erläutert durch das Qualitätsziel Z11: [Komposition | EFFIZIENZ] beschrieben. Die Anwendung des Qualitätsmodells besteht nun in der Analyse der Einflüsse auf diese Qualitätsziele. Tabelle 5.8 zeigt eine entsprechende Projektion des Qualitätsmodells unter Berücksichtigung der Trennung zwischen Faktoren aus dem Konformitäts- und solchen aus dem Design-Modul.

Für die Effizienz der Anpassung zeigt sich, dass vor allem die SOA-Prinzipien der lo-

sen Kopplung und des hohen Abstraktionsgrades von Bedeutung sind. Darüber hinaus ist aber auch Granularität eine wichtige Dimension, sowohl in funktionaler Hinsicht als auch bezogen auf die verwendeten Datentypen. Bezüglich effizienter Komposition zur Erweiterung eines Systems spielen außerdem Faktoren zur sinnvollen Beschreibung, Komplexität und Kohäsion von Diensten eine wichtige Rolle.

Darüber hinaus ist auch das Qualitätsziel Z8: [Interne Analyse | EFFIZIENZ], also die Analyse von Diensten zum Zwecke der Anpassung oder Erweiterung, von Bedeutung. Zwar sind die von außen sichtbaren Eigenschaften eines Dienstes für diese Aktivität im Allgemeinen irrelevant, weshalb zunächst keine Einflüsse von Faktoren auf der Ebene allgemeiner SOA-Prinzipien zu erwarten wären. Da dieses Qualitätsziel jedoch auch die Analyse der möglichen Auswirkungen einer Änderung umfasst, ist es wichtig zu untersuchen, von welchen anderen Diensten im System ein Dienst verwendet wird. Der Aufwand hierfür ist im Fall von dynamischer Komposition verglichen mit statisch definierten Abhängigkeiten größer, so dass sich SOA negativ auf dieses Qualitätsziel auswirkt.

Es lässt sich also feststellen, dass ein Teil der beeinflussenden Faktoren für Flexibilität tatsächlich von den SOA-Prinzipien herrühren und somit das Argument, dass SOA eine positive Auswirkung auf die Flexibilität eines Softwaresystems hat, bedingt gerechtfertigt ist. Allerdings deutet die Vielzahl der Faktoren aus dem Designmodul darauf hin, dass SOA allein noch keine Flexibilität garantieren kann. Auch ein mit allen SOA-Prinzipien konformes System kann durch Nichtbeachtung der oben aufgeführten Faktoren schlecht anpassbar oder erweiterbar sein.

Uniformität der Beschreibung

Die Uniformität der Architekturbeschreibung stellt einen Sonderfall innerhalb dieser Betrachtung dar, da sie sich zwar als Qualitätskriterium auffassen lässt, gleichzeitig aber auch direkt als SOA-Prinzip formuliert ist. Im Faktor F6: [Schnittstelle | BESCHREIBUNGSSPRACHEN] auf Seite 87 beschreibt das SOA-Konformitätsmodell die Eigenschaft der einheitlichen Beschreibung von Schnittstellen als Ausdruck des Prinzips der Standardisierung. Insofern sorgt die vollständige Erfüllung dieses Faktors unmittelbar dafür, dass das Versprechen der Uniformität eingehalten wird. Die Anwendung des Modells zur Bewertung einer Software kann über diese Erkenntnis hinaus konkrete Hinweise darauf liefern, inwieweit tatsächlich einheitliche Beschreibungen vorhanden sind.

In beiden Fällen wird allerdings ausschließlich die *syntaktische* Uniformität beschrieben respektive untersucht. Über Dienstschnittstellen ausgetauschte Daten bleiben dabei etwa hinsichtlich verwendeter Einheiten unberücksichtigt. Die auch auf semantischer Ebene uniforme Beschreibung eines Systems bedarf weitaus umfassenderer Überprüfungen, beispielsweise in den Bereichen Datenformate, Einheiten oder synonym bzw. homonym verwendete Bezeichner.

Auffindbarkeit

Das Qualitätsziel Z1: [Auffinden | EFFIZIENZ] wird lediglich von F10: [Service | ANGEMESSENE BENENNUNG] beeinflusst, so dass auf den ersten Blick SOA für die Effizienz des Auffindens von Diensten keine Relevanz besitzt. Ein entscheidender Bestandteil des klassischen SOA-Dreiecks (vgl. [69, S. 29], [71]) ist die *Service Registry*, bei der sich Dienste registrieren, um später aufgefunden werden zu können. Die praktische Erfahrung zeigt jedoch, dass im Kontext von betrieblichen Informationssystemen Dienste zumeist statisch miteinander verknüpft werden und damit praktisch keine Dienstsuche stattfindet [103]. Die Dienstsuche ist daher vor allem für solche Dienste relevant, die im Internet angeboten werden. Hierfür mangelt es jedoch aktuell an offen zugänglichen Verzeichnissen oder Dienste-Plattformen. Das Forschungsprojekt InDiNet[2] entwickelt derzeit eine solche offene Dienste-Plattform, stellt für die Übergangszeit aber auch Konzepte bereit, bestehende Dienstbeschreibungen durch semantische Annotationen für aktuelle Internet-Suchmaschinen aufzubereiten. Neben klassischen Bezeichnern und Beschreibungen sind dabei vor allem Informationen zum Preismodell der Dienste vorgesehen, die allerdings nicht dem Auffinden dienen, sondern der Unterstützung einer möglichen Kaufentscheidung. Somit ist auch in diesem Ansatz die Benennung des Dienstes das zentrale Kriterium für dessen Auffindbarkeit.

Zuverlässigkeit und Effizienz in Nutzung und Betrieb

Die Zuverlässigkeit (englisch *Reliability*) von Diensten wird durch die Qualitätsziele Z5: [Konsum | KONTINUITÄT] und Z7: [Betrieb | KONTINUITÄT] beschrieben. Diese werden im Qualitätsmodell von Faktoren aus beiden Modulen beeinflusst. Da Dienste in betrieblichen Informationssystemen häufig statisch komponiert werden und damit die theoretischen Vorteile von SOA hinsichtlich Ausfallsicherheit durch ein Angebot redundanter, dynamisch komponierbarer Dienste nicht zum Tragen kommt, ist der Einfluss aus dem Bereich der SOA-Konformität allerdings gering.

Die Qualitätsziele Z4: [Konsum | EFFIZIENZ] und Z6: [Betrieb | EFFIZIENZ] entsprechen den ISO/IEC-Charakteristiken *Performance Efficiency* bzw. *Operability*. Für sie sind keine nachgewiesenen Einflüsse aus dem SOA-Konformitätsmodul vorhanden. Ein einfacher Lastausgleich zur Steigerung der Effizienz und der Skalierbarkeit wird vor allem durch die Zustandslosigkeit von Diensten ermöglicht, was in F21: [Service | ZUSTANDSLOSIGKEIT] beschrieben wurde, allerdings aufgrund der in Abschnitt 4.5.6 diskutierten Gründe nicht als SOA-Prinzip repräsentiert ist. Darüber hinaus tragen die Anzahl der zur Ausführung einer Operation durchgeführten Aufrufe weiterer Operationen sowie die Verteilung der entsprechenden Dienste über unterschiedliche Knoten im Netzwerk entscheidend zur Effizienz bei. Auch diese Faktoren sind allerdings nicht von SOA-Prinzipien abgeleitet, sondern beschreiben darüber hinausgehende Design-Eigenschaften.

[2] `http://www.software-cluster.com/de/swc/projekte/verbundprojekte/indinet`

Zusammenfassung

Zusammenfassend wird klar, dass die Erfüllung der SOA-Prinzipien durchaus zu Qualitätsverbesserungen in den einzelnen Bereichen beiträgt. Bezüglich der Interoperabilität bestehen etwa positive Einflüsse ausgehend vom Prinzip der Standardisierung von Schnittstellen. Allerdings bestehet die große Herausforderung bezüglich Interoperabilität in der Semantik, die den Diensten zugrunde liegt. Auch Flexibilität wird durch die SOA-Prinzipien positiv beeinflusst, insbesondere durch den engeren Bezug von Diensten zu Geschäftsprozessen, Standardisierung von Schnittstellenbeschreibungen und der Komponierbarkeit von Diensten. Allerdings sind eine Vielzahl weiterer Faktoren erforderlich, um Flexibilität sicherzustellen, so dass SOA nicht als alleiniger Garant angesehen werden kann. Selbst die uniforme Beschreibung eines Systems wird durch SOA nicht vollständig sichergestellt. Hier spielen außer der syntaktischen Standardisierung von Schnittstellen vor allem Datenformate sowie Einheiten eine Rolle, welche zusätzlich zu den SOA-Prinzipien vereinheitlicht werden müssen. Auffindbarkeit von Diensten kann allenfalls durch die Existenz einer Service-Registry positiv beeinflusst werden. Allerdings zeigt die Praxis, dass im Unternehmensumfeld Dienste häufig statisch komponiert werden und dass nach öffentlich zugänglichen Dienste im Internet im Allgemeinen über klassische Suchmaschinen gesucht wird. Auch bezüglich Zuverlässigkeit und Performanz existieren keine Verbesserungen, die allein aus den SOA-Prinzipien abgeleitet werden können.

Generell sind also die SOA-Prinzipien nicht die einzigen Einflussfaktoren für die Qualität der Software und werden zudem erfahrungsgemäß nicht in allen als SOA bezeichneten Architekturen auch konsequent umgesetzt. Allerdings sind die Einflüsse der SOA-Prinzipien auf die Softwarequalität fast ausschließlich positiv, so dass die konsequente Umsetzung von SOA durchaus einen Qualitätszuwachs bringen kann – vorausgesetzt, die übrigen qualitätsrelevanten Faktoren werden dabei nicht vernachlässigt.

5.3.2 Architektur-Qualitätsbewertung

Um die Anwendbarkeit des Qualitätsmodells zur Bewertung der Qualität serviceorientierter Architekturen und das Zusammenspiel des SOA-Qualitätsmodells mit den in Quamoco erstellten Werkzeugen zur Modellierung und Qualitätsanalyse zu demonstrieren, wurde eine Fallstudie durchgeführt. Dazu wurde wie bereits bei der Expertenbefragung die maschinenlesbare Variante des SOA-Qualitätsmodells verwendet, welche mit dem Quamoco-Qualitätsmodell-Editor erstellt wurde.

Als Versuchsobjekt diente die in der Entwicklung befindliche SOA des EU-Forschungsprojekts *SmartProducts*[3]. Aufgrund der unterschiedlichen Entwicklungsstände der einzelnen Services und der ausschließlich prototypischen und einmaligen Durchführung der Qualitätsanalyse wurden sämtliche Maße manuell erhoben. Dazu wurde das Qualitätsmodell entsprechend angepasst, so dass ausschließlich sogenannte *Manual Instruments* zum Einsatz kamen. Aufgrund einer Vorgabe des Qualitätsanalysewerk-

[3] `http://www.smartproducts-project.eu`

zeugs können manuell erhobene Maße ausschließlich numerisch sein. Dies ermöglicht die automatische Generierung einer Messvorlage im Excel-Format, welche neben der eindeutigen Kennung des Maßes auch dessen Beschreibung enthält, sowie eine Spalte, in die der Experte im Anschluss an die Messung die Messwerte eintragen kann. Für die manuelle Erhebung der Messwerte konnten fünf an der Entwicklung der Architektur beteiligte Experten gewonnen werden, welche alle über ein abgeschlossenes Studium der Informatik sowie langjährige praktische Erfahrung in der Entwicklung von Software verfügen. Zwei der Experten besitzen darüber hinaus einen Doktorgrad. Tabelle 5.9 zeigt die Kennungen der Maße sowie deren Werte. Bei den Werten handelt es sich um Mittelwerte aus den fünf von Experten durchgeführten Analysen. Die dritte Spalte referenziert die entsprechenden Maße, welche in Abschnitt 4.5 und Abschnitt 4.6 beschrieben sind.

Die von den Experten ausgefüllte Tabelle wird im Anschluss durch das Analysewerkzeug eingelesen und zur Berechnung der im Qualitätsmodell enthaltenen Faktoren verwendet. Das Analysewerkzeug stellt die Ergebnisse der Qualitätsbewertung neben den grafischen Darstellungen (siehe Abbildung 4.11 auf Seite 113) als hierarchische Tabelle im HTML-Format zur Verfügung. Ein Ausschnitt aus der so generierten Übersicht ist in Abbildung 5.2 dargestellt. Dabei enthält die erste Spalte die hierarchische Dekomposition der Qualitätsziele, inklusive der sie beeinflussenden Faktoren und der zugeordneten Maße. Die rechte Spalte zeigt entsprechende Ergebnisse, was bei Qualitätszielen und Faktoren Werten im Intervall $[0; 1]$ entspricht, welche den Erfüllungsgrad des betreffenden Elements repräsentieren.

Bei der Interpretation der Zahlenwerte ist anzumerken, dass diese nicht als absolute Aussagen zu verstehen sind. Die Erfahrung aus der Arbeit mit Modellen zur Qualitätsbewertung hat gezeigt, dass die Kalibrierung von Grenzwerten unter Zuhilfenahme einer großen Basis an Referenzprodukten ein essentieller Schritt auf dem Weg zu einem objektiven Qualitätsbewertungsmodell ist. Die hier durchgeführte exemplarische Anwendung des Qualitätsmodells dient einzig dem Ziel, die praktische Anwendbarkeit des Modells sowie die Kompatibilität mit den im Quamoco-Projekt entwickelten Werkzeugen zu belegen. Eine intuitive Interpretation (insbesondere mittels Abbildung auf intuitiv verständliche Skalen wie etwa Schulnoten) erfordert eine vorherige Kalibrierung der zu erwartenden Wertebereiche für Maße sowie die Gewichtung der beeinflussenden Faktoren für die Qualitätsziele auf Grundlage einer großen Anzahl von bewerteten Systemen. Für das Quamoco-Basismodell für Sourcecode-Qualität wurden zu diesem Zweck etwa 120 Softwareprodukte analysiert und miteinander verglichen [10].

Trotz dieser Beschränkungen können die Aussagen zu einzelnen Faktoren bereits Hinweise auf die Qualität der Architektur liefern. Ein auffälliges Ergebnis der Analyse war beispielsweise der vergleichsweise niedrige Wert für das SOA-Prinzip der Abstraktion (siehe Zeile *High Abstraction Level* in Abbildung 5.2). Dieser wurde in der Diskussion der Ergebnisse mit den beteiligten Experten als gerechtfertigt bestätigt. Es sei in der betrachteten Architektur durchaus der Fall, dass feingranulare Programmierschnittstellen als Services exponiert werden, und vergleichsweise wenig Geschäftsprozess-relevante Funktionalität. Die Qualitätsanalyse mit Hilfe des SOA-Qualitätsmodells ermöglicht es also den Entwicklern einer Architektur, mit geringem

Maß	Messwert	Entsprechung
IDA	0,00	M7: *Beschreibung von Implementierungsdetails*
ISC	0,50	M6: *Anzahl konkurrierender Standards*
NBS	5,00	M5: *Abstraktionsgrad von Diensten*
NDCCS	2,00	M9: *Fan-In*
NDCPS	2,00	M1: *Fan-Out*
NIS	3,00	M5: *Abstraktionsgrad von Diensten*
NPS	4,00	M5: *Abstraktionsgrad von Diensten*
SSNS	12,00	M14: *Anzahl der Dienste im System*
CAUD	50,00	M10: *Bedeutsame Beschreibung*
CSD	0,00	M21: *Gegenseitige Dienstabhängigkeiten*
DDT	2,00	M31: *Abhängigkeitstiefe*
EGP	6,00	M23: *Datengranularität*
MON	28,00	M12: *Angemessenheit des Operationsnamens*
MPN	35,00	M13: *Bedeutsame Parameter-Namen*
MSN	5,00	M11: *Angemessenheit des Dienstnamens*
NCDO	5,00	M29: *Anzahl kontextabhängiger Operationen*
NEO	0,00	M16: *Operationen an der Systemgrenze*
NO	55,00	M18: *Anzahl Operationen*
NP	100,00	M19: *Operationskomplexität*
ONXD	10,00	M15: *Rechner-externe Abhängigkeiten*
OPMF	10,00	M22: *Anzahl ausgeführter Funktionen*
SDT	2,00	M30: *Abhängigkeitssumme*
SES	0,00	M17: *Semantisch äquivalente Dienste*
SIDC	0,40	M24: *Gemeinsame Datentypen für Parameter*
SISC	0,15	M26: *Abfolge-Zusammenhang zwischen Operationen*
SIUC	0,50	M25: *Nutzung von Operationen durch Dienstkonsumenten*
TCFD	11,00	M20: *Berührte und voll abgedeckte funktionale Domänen*

Tabelle 5.9: Messwerte der Architektur

Qualitaetsbewertung (Main)
Qualitaetsbewertung

Element	Evaluation Result
Property @Product []	
Quality @UseCase []	0,613
Quality @Anpassung [refined]	0,652
Effizienz @Anpassung [refined]	0,652
Quality @Betrieb [refined]	0,512
Effizienz @Betrieb [refined]	0,546
Kontinuität @Betrieb [refined]	0,479
Quality @Ext. Analyse [refined]	0,687
Effizienz @Ext. Analyse [refined]	0,687
Cohesion @ServiceInterface [impacted]	0,417
Complexity @ServiceInterface [impacted]	0,841
Data Granularity @Service Operation [impacted]	0,564
Functional Granularity @Service [impacted]	0,958
High Abstraction Level [impacted]	0,458
Meaningful Names @ServiceInterface [impacted]	0,573
Proper Documentation @ServiceInterface [impacted]	1,000
Quality @Int. Analyse [refined]	0,499
Effizienz @Int. Analyse [refined]	0,499
Quality @Komposition [refined]	0,705
Effektivität @Komposition [refined]	0,750
Effizienz @Komposition [refined]	0,661
Quality @Konsum [refined]	0,498
Effektivität @Konsum [refined]	0,490
Effizienz @Konsum [refined]	0,523
Low Fragmentation @SOA System [impacted]	0,091
Short Dependency Paths @Service [impacted]	0,956
DDT	rank=1 threshold=[0,0 ; 3,0] ratio=0,167 value=0,944
SDT	rank=1 threshold=[0,0 ; 5,0] ratio=0,167 value=0,967
Kontinuität @Konsum [refined]	0,479
Quality @Test [refined]	0,736
Effizienz @Test [refined]	0,736

Abbildung 5.2: Bewertungsergebnis (Ausschnitt)

Aufwand bereits vor Abschluss der Implementierung Transparenz bezüglich der zu erwartenden Übereinstimmung mit den SOA-Prinzipien zu schaffen und Hinweise auf mögliche Qualitätsprobleme zu bekommen. Insbesondere wurde die automatische Analyse und Aufbereitung der Ergebnisse von den beteiligten Experten als große Erleichterung im Gegensatz zu anderen, bis dahin verwendeten Verfahren zur Architekturbewertung wahrgenommen. Insbesondere wurde bei *Active Design Reviews* [117] ein erhöhter Aufwand in der Erstellung der benötigten Fragebögen sowie in der Auswertung der Ergebnisse beobachtet. Da viele andere Verfahren zur Architekturbewertung ebenfalls die Erstellung spezifischer Szenarien, Fragebögen oder Checklisten erfordern, sind ähnliche Ergebnisse zu erwarten. Diese Annahme wird gestützt durch Aufwandsschätzungen für die von Clements, Kazman und Klein [42] beschriebenen Verfahren, die z. B. bei ATAM von etwa 70 Personentagen für ein mittelgroßes Projekt ausgehen.

5.4 Zusammenfassung

Durch die in diesem Kapitel beschriebenen Maßnahmen konnte einerseits die Vollständigkeit und Angemessenheit des erarbeiteten SOA-Qualitätsmodells bestätigt werden, andererseits wurde ein breites Spektrum an Einsatzmöglichkeiten für das Modell aufgezeigt und mittels exemplarischer Betrachtungen die Tauglichkeit des Modells für diese Einsatzzwecke nachgewiesen.

Die Befragung von zwölf Experten auf dem Gebiet von SOA und Softwarequalität durch semi-strukturierte Interviews hat gezeigt, dass das Qualitätsmodell aussagekräftig und vollständig genug ist, um die für die Qualität von SOA-basierten Systemen relevanten Faktoren zu beschreiben und messbar zu machen. Durch die Definition des SOA-Qualitätsmodells als Ergänzung zu Sourcecode-basierten Qualitätsmodellen konnte auf SOA-unspezifische Anteile weitgehend verzichtet und die Komplexität des Modells in einem handhabbaren Rahmen gehalten werden, was in der Expertenbefragung bestätigt wurde. Darüber hinaus wurde in den Befragungen deutlich, dass je nach fachlicher Domäne eines SOA-Systems die für die Qualität als wichtig erachteten und zur Aufnahme in das Qualitätsmodell vorgeschlagenen Faktoren sehr unterschiedlich sind. Diese Beobachtung bestätigt den gewählten Ansatz, das Qualitätsmodell auf Basis einer modularen und leicht erweiterbaren Struktur zu definieren (vgl. Abschnitt 4.2.3), so dass projektspezifische Anpassungen für den praktischen Einsatz möglich sind. Das SOA-Qualitätsmodell wurde insgesamt als gut strukturiert und verständlich wahrgenommen. Ein besonderer Mehrwert des Modells liegt nach Ansicht eines Experten in der gesteigerten Transparenz und Objektivierbarkeit abstrakter Qualitätsbegriffe wie *Maintainability*. Die Architekturmerkmale, welche durch die im Modell enthaltenen Maße erfasst werden, könnten laut Einschätzung eines anderen Experten außer zur Qualitätsanalyse auch herangezogen werden, um etwa das zu erwartende Datenvolumen oder die Aufwände für Entwicklung, Support oder Test abzuschätzen.

Aus theoretischer Sicht konnten mit Hilfe des Qualitätsmodells Aussagen über den Zusammenhang zwischen SOA und Softwarequalität faktenbasiert beleuchtet werden.

Damit wird dem Umstand, dass Auswirkungen der Spezifika unterschiedlicher Architekturstile auf die Qualitätsattribute einer Software oft nur unzureichend untersucht und dokumentiert sind, zumindest für den Bereich SOA Rechnung getragen. Einerseits kann mit Hilfe des Qualitätsmodells begründet dargestellt werden, welche SOA-Prinzipien zur Verbesserung welcher Qualitätsaspekte beitragen. Andererseits wird aber auch transparent, welche übrigen Eigenschaften der Architektur einen Einfluss auf diese Qualitätsaspekte besitzen. Für die Flexibilität konnte etwa gezeigt werden, dass diese zwar durch die SOA-Prinzipien der Abstraktion und der losen Kopplung positiv beeinflusst wird, aber auch der Datengranularität und einer Reihe weiterer Faktoren eine entscheidende Bedeutung hinsichtlich der Flexibilität zukommt, so dass der Architekturstil SOA allein nicht als Garant für Flexibilität gelten kann. Auch andere Qualitätsziele werden von SOA durchaus positiv beeinflusst, können aber nicht allein durch die Umsetzung der SOA-Prinzipien erreicht werden. Eine zusammenfassende Analyse dieser Einflüsse ermöglicht es schließlich, das Qualitätsmodell als Entscheidungshilfe bei der Wahl eines Architekturstils zu nutzen, indem Qualitätsanforderungen an ein System mit den durch SOA bedingten Einflüssen auf die Produktqualität verglichen werden. Durch weitere Konkretisierung des Qualitätsmodells für unterschiedliche SOA-Frameworks würde es zudem möglich, auch diese auf Grundlage der von ihnen bedingten Auswirkungen auf die Qualität auszuwählen.

Schließlich konnte in einer Fallstudie nachgewiesen werden, dass sich das Qualitätsmodell grundsätzlich für die Bewertung serviceorientierter Architekturen eignet. Aus Sicht der beteiligten Architekten und Entwickler des in der Fallstudie betrachteten SOA-Systems besteht der größte Vorteil dieser Möglichkeit in einem drastisch reduzierten Aufwand der Analyse und Aufbereitung der Ergebnisse im Vergleich zu anderen Ansätzen zur Architekturbewertung. Dies ermöglicht bei gleichem Aufwand häufigere Bewertungen der Architektur, insbesondere in frühen Stadien der Softwareentwicklung, was mit erhöhter Transparenz bezüglich der spezifischen Eigenschaften der Architektur einhergeht und frühes Gegensteuern bei identifizierten Risiken hinsichtlich der Qualität der Software ermöglicht.

6 Zusammenfassung und Ausblick

Dieses Kapitel schließt die vorliegende Dissertation ab und gibt einen Ausblick auf zukünftige Arbeiten. Zunächst werden in Abschnitt 6.1 die Ergebnisse der Arbeit zusammengefasst und mit den eingangs erwähnten Forschungslücken und -fragen in Bezug gesetzt. In Anschluss greift Abschnitt 6.2 diese Ergebnisse auf und skizziert weiter gehende Fragestellungen, die sich daraus ergeben.

6.1 Zusammenfassung

Steigende Komplexität von Software sowie die zunehmende Serviceorientierung von Softwaresystemen erfordern neue Ansätze im Umgang mit Softwarequalität. Insbesondere wird bei einem zunehmenden Angebot funktional ähnlicher Dienste Qualität zu einem differenzierenden Faktor. In Verbindung mit der großen Bedeutung von SOA für heutige Software lassen sich die folgenden Probleme beobachten: (1) Es existieren kaum nachgewiesene Aussagen darüber, in welcher Form sich die charakteristischen Eigenschaften eines Architekturstils (insbesondere SOA) auf die Qualität einer Software auswirken, wodurch Architekturentscheidungen oft vor allem auf Basis persönlicher Erfahrung getroffen werden müssen. (2) Die Bewertung von Softwarearchitekturen ist im Allgemeinen sehr aufwändig, da entsprechende Verfahren oft ein hohes Maß an manuellem Aufwand erfordern. (3) Insbesondere für die Bewertung der Qualität von Diensten im Kontext serviceorientierter Architekturen ist kein etabliertes Verfahren bekannt, welches eine einfache Reproduzierbarkeit der Qualitätsanalyse sowie eine Vergleichbarkeit unterschiedlicher analysierter Dienste ermöglicht. (4) Im Hinblick auf den Software-Entwicklungsprozess existieren Medienbrüche im Qualitätsmanagement zwischen der Architektur- und der Implementierungsphase, so dass eine kontinuierliche Überwachung und Steuerung der Produktqualität erheblich erschwert wird.

Basierend auf diesen Lücken im Stand der Wissenschaft bzw. Technik widmete sich diese Dissertation der Forschungsfrage: „Welche Qualitätseigenschaften besitzt SOA, und wie kann die Qualität einer SOA mit Hilfe eines geeigneten Modells dargestellt und nachgewiesen werden?“ Diese Frage schließt sowohl die inhärent mit dem Architekturstil SOA verknüpften Qualitätseigenschaften ein, als auch solche, welche unabhängig davon durch geeigneten Entwurf des Systems beeinflusst werden können.

Zur Beantwortung der Forschungsfrage wurden in Kapitel 2 zunächst Grundlagen der Softwarequalität und serviceorientierter Architekturen zusammengestellt. Aus dem Bereich verwandter Arbeiten wurden insbesondere solche analysiert und dokumentiert, die den Zusammenhang zwischen SOA und Softwarequalität adressieren. Dies waren einerseits Arbeiten, welche allgemein die Auswirkungen von SOA auf die Qualität

einer Software thematisieren, andererseits Qualitätsmodelle für serviceorientierte Architekturen. Beide Arten verwandter Arbeiten wurden kritisch hinterfragt und zur Referenzierung innerhalb des selbst entwickelten Qualitätsmodells in Betracht gezogen. Generell war kein Qualitätsmodell für SOA verfügbar, welches sowohl einen allgemeinen Qualitätsbegriff zugrunde legt als auch konkrete Maße zur Bestimmung der Qualitätseigenschaften vorschlägt.

Zur Bestimmung der relevanten Faktoren für allgemeine Eigenschaften einer Architektur, welche die Softwarequalität beeinflussen, wurde eine umfangreiche Literaturanalyse durchgeführt und in Kapitel 3 dokumentiert. Diese Analyse basiert auf insgesamt 21 wissenschaftlichen Zeitschriften aus dem Umfeld des Software Engineerings der letzten 25 Jahre. In einem mehrstufigen Verfahren wurden aus einer Gesamtmenge von 17 294 Artikeln diejenigen identifiziert, welche sich spezifischen Einflüssen einzelner Aspekte von Architektur auf die Softwarequalität widmen und diese experimentell nachweisen. Die als Ergebnis dieser Analyse identifizierten und auf SOA angepassten Einflüsse bilden zusammen mit den in Kapitel 2 referenzierten Betrachtungen aus der SOA-Literatur die Grundlage für die Definition des Qualitätsmodells.

In Kapitel 4 wurde ein Qualitätsmodell für SOA entwickelt, welches sowohl die charakteristischen (und aus den SOA-Prinzipien hervorgehenden) Eigenschaften der Architektur betrachtet als auch weitere auf Ebene der Architektur zu erfüllende Eigenschaften, welche für deren Qualität von Bedeutung sind. Das Qualitätsmodell orientiert sich dabei an der aktivitätenbasierten Darstellung von Qualitätszielen. Das bedeutet, dass die Qualität einer Software mit Hilfe von Aktivitäten ausgedrückt wird, die an der Software oder mit ihr durchgeführt werden. Auf diese Weise lässt sich eine eindeutige Semantik der Verfeinerung von Qualitätszielen durch die Beschreibung von Teilaktivitäten erreichen. Die zentralen Elemente einer SOA und deren Beziehungen werden mittels eines SOA-Referenzmodells in Form von Entitäten im Qualitätsmodell verankert. Faktoren beschreiben beobachtbare Eigenschaften dieser Entitäten sowie deren Einflüsse auf die Qualitätsziele, basierend auf den Ergebnissen der in den Kapiteln 2 und 3 beschriebenen Arbeiten. Einen wichtigen Mehrwert stellt dabei die explizite Unterscheidung zwischen von den SOA-Prinzipien ausgehenden Einflüssen und Einflüssen aus weiteren Eigenschaften der Architektur dar, was eine getrennte Betrachtung dieser Dimensionen ermöglicht. Gleichzeitig bilden Faktoren ein Bindeglied zwischen allgemeinen Qualitätszielen und konkret zu messenden Größen, indem das Qualitätsmodell ihnen Maße zur Quantifizierung zuordnet und damit die Anbindung von Werkzeugen zur Erhebung der entsprechenden Messwerte sowie die Anwendung des Modells zur Qualitätsanalyse erlaubt.

Die Vollständigkeit und Angemessenheit des SOA-Qualitätsmodells wurde durch SOA-Experten aus Industrie und Forschung in Befragungen beurteilt. Dabei wurde das Modell als gut strukturiert und verständlich bewertet. Außerdem wurde deutlich, dass das Modell die als am wichtigsten eingeschätzten Eigenschaften der Architektur gut bis sehr gut durch entsprechende Faktoren und Maße abdeckt. Einige Experten schlugen die Einbeziehung weiterer Faktoren für ihre spezifische Anwendungsdomäne in das Qualitätsmodell vor. Dieses soll jedoch eine gemeinsame, domänenunabhängige Grundlage für die Definition von SOA-Qualität bilden und wurde deshalb in den

fraglichen Punkten bewusst nicht ergänzt. Stattdessen erlaubt die modulare Struktur des Modells domänen- und projektspezifische Erweiterungen bis hin zur Neugewichtung von Faktoren, etwa um Qualitätsziele abweichend zu priorisieren. Auch die Möglichkeit der Anbindung von Werkzeugen zur Qualitätsanalyse wurde sehr positiv wahrgenommen und zum produktiven Einsatz vorgeschlagen.

Über die inhaltliche Bewertung des Modells hinaus wurden mehrere mögliche Nutzungsszenarien für das Qualitätsmodell beschrieben und exemplarisch vollzogen. Dazu zählt die Überprüfung häufig mit SOA in Verbindung gebrachter Qualitätsversprechen. Hier können aufgrund der im Modell nachvollziehbaren Einflüsse Rückschlüsse darauf gezogen werden, inwiefern ein Qualitätsziel durch die SOA-Prinzipien beeinflusst wird und ob die Architektur zusätzliche Eigenschaften besitzen muss, um dieses Qualitätsziel zu erreichen. Am Beispiel der Flexibilität wird deutlich, dass diese zwar durch die SOA-Prinzipien der losen Kopplung und der fachlichen Abstraktion positiv beeinflusst wird, dass sich allerdings auch weitere Faktoren (z. B. die Granularität der Schnittstellenparameter) auf die Flexibilität der Architektur auswirken. Generell wirken sich dabei die SOA-Prinzipien positiv auf die Softwarequalität aus, können diese aber nicht allein sicherstellen, so dass eine Vielzahl weiterer Faktoren berücksichtigt werden muss, um Software von hoher Qualität zu entwickeln. Das Qualitätsmodell dient somit als Wissensbasis, um Argumente bezüglich der Vorteile von SOA hinsichtlich der Softwarequalität strukturiert zu begründen oder zu widerlegen.

Eine weitere Anwendung des Qualitätsmodells ist die modellbasierte Qualitätsbewertung serviceorientierter Architekturen. Dazu wurde exemplarisch ein Softwaresystem mit Hilfe des Qualitätsmodells analysiert. Trotz der zu diesem Zweck manuell erhobenen Maße beurteilten die beteiligten Architekten und Entwickler den für die Architekturanalyse benötigten Aufwand als wesentlich geringer im Vergleich zu bisher verwendeten Vorgehensweisen. Auch die durch die Analyseergebnisse aufgezeigten Architektureigenschaften wurden von den Experten als sehr zutreffend bewertet.

Die Dissertation konnte folglich zur Lösung wichtiger Probleme bezüglich SOA und Softwarequalität beitragen. Das Qualitätsmodell beschreibt Auswirkungen von SOA auf die Softwarequalität und belegt diese sowohl argumentativ als auch unter Verweis auf weitere Publikationen. Damit wird fragmentiert vorliegendes Wissen bezüglich dieser Auswirkungen gesammelt, konsistent abgebildet und konserviert. Das Qualitätsmodell kann verwendet werden, um ausgehend von einem Qualitätsziel zu analysieren, welche Einflüsse sich aufgrund der SOA-Prinzipien ergeben, und welche weiteren Einflüsse auf dieses Qualitätsziel von darüber hinaus gehenden Eigenschaften der Architektur bestehen. Damit wird es möglich zu argumentieren, inwieweit der Architekturstil SOA an sich bereits positive Auswirkungen auf die Softwarequalität besitzt und ob diese durch Nichtbeachtung weiterer Design-Eigenschaften möglicherweise relativiert werden.

Das Qualitätsmodell bildet außerdem die Grundlage für standardisierte und vergleichbare Qualitätsbewertungen serviceorientierter Architekturen. Dies wird durch eine Formalisierung der Beziehungen zwischen Maßen, Faktoren und Qualitätszielen erreicht, welche eine Operationalisierung des Modells zur Qualitätsanalyse erlaubt.

Grady [65] beschrieb 1992 seine Vision bezüglich der Rolle von Metriken im Software Engineering für das Jahr 2000 wie folgt:

> First, tools will automatically measure size and complexity for all the work products that engineers develop. Besides warnings and error messages, the tools will predict *potential* problem areas based on metric data thresholds. [65, S. 220]

Für das Arbeitsprodukt *Programmcode* mag dieses Ziel durch die breite Verfügbarkeit von *Dashboards* mittlerweile erreicht sein; bezogen auf Softwarearchitekturen ist diese Vision auch 13 Jahre nach dem genannten Zeitpunkt noch nicht Realität. Das Qualitätsmodell für SOA leistet hier einen Beitrag zur weiteren Realisierung dieser Vision, indem es klare Zusammenhänge zwischen Architekturmetriken und Qualitätszielen beschreibt und begründet sowie eine automatisierte Bewertung hinsichtlich dieser Ziele ermöglicht. Damit trägt das Qualitätsmodell zur Reproduzierbarkeit von Qualitätsanalysen bei, so dass etwa bei Änderungen an der Software sichergestellt ist, dass einer erneuten Qualitätsanalyse dieselben Berechnungsvorschriften zugrunde liegen. Diese Reproduzierbarkeit geht mit einer Aufwandsreduktion einher, da die Aggregation und Aufbereitung der Messwerte durch Anbindung bestehender Werkzeuge automatisiert erfolgen kann. Weitere Automatisierung wird ermöglicht, indem die im Qualitätsmodell enthaltenen Maße ebenfalls durch Werkzeuge erfasst und in die Analyse eingebunden werden können. Die Verwendung dieser bestehenden Infrastruktur trägt überdies dazu bei, Medienbrüche im Qualitätsmanagement von Softwareprojekten zu vermeiden. Hier können bereits auf der Ebene der Architektur Qualitätsanalysewerkzeuge zum Einsatz kommen, welche gleichermaßen für die Analyse des Quellcodes verwendet werden. Außer einer durch diese Konsistenz bedingten gesteigerten Erlernbarkeit und Akzeptanz kann durch die Konsolidierung von Qualitätsanalysewerkzeugen letztlich auch eine Kosteneinsparung im Qualitätsmanagement erreicht werden. Die vorliegende Dissertation liefert also wertvolle Beiträge sowohl für die Wissenschaft als auch für die industrielle Praxis.

6.2 Ausblick

Eine Erkenntnis aus der in Abschnitt 5.2 beschriebenen Expertenbefragung ist, dass je nach Anwendungsdomäne unterschiedliche Eigenschaften der Architektur für die Qualität des Softwaresystems von Bedeutung sind. Für den Quelltext von Software wurde im Forschungsprojekt Quamoco eine analoge Einschätzung getroffen: Auch dort erfordern unterschiedliche Anwendungsdomänen die Bestimmung unterschiedlicher Eigenschaften, ggf. verbunden mit einer Neugewichtung der vorhandenen Eigenschaften hinsichtlich der Wichtigkeit ihres Einflusses auf die Qualität. Ähnlich zu dem in Quamoco verfolgten Ansatz, spezifische Qualitätsmodelle als Erweiterungen eines allgemeingültigen Basismodells zu definieren, können basierend auf dem Qualitätsmodell für SOA domänenspezifische Ergänzungen entwickelt werden, um die Unterschiede in

der Architektur von Informationssystemen, Kollaborationsplattformen oder Anwendungen im Kontext des Internets der Dinge angemessen abzubilden. Erste Hinweise auf für diese Domänen spezifische Faktoren sind bereits in Abschnitt 5.2.3 zu finden.

Weitere Ansätze für zukünftige Arbeiten ergeben sich aus der Betrachtung zu Qualitätsmodellen von Deißenböck u. a. [47]: Die Autoren sehen ein Qualitätsmodell für Software unter anderem als Grundlage für die systematische Entwicklung von Werkzeugen zur statischen Analyse. Dementsprechend kann das SOA-Qualitätsmodell dazu dienen, Analysewerkzeuge gezielt um Funktionalität zu erweitern, welche die hier beschriebenen Maße implementiert. Dadurch kann der Automatisierungsgrad der modellbasierten Architekturanalyse weiter gesteigert werden, was den Aufwand einer einzelnen Analyse reduziert und damit häufigere Qualitätsanalysen ermöglicht. Darüber hinaus kann bei der Einführung einer neuen Technologie, welche auf SOA basiert, mit Hilfe des Qualitätsmodells identifiziert werden, wie groß die Bedeutung einzelner Maße für die Qualitätsanalyse ist und mit welcher Priorität an der Automatisierung dieser Maße gearbeitet werden sollte.

Um die Ergebnisse der modellbasierten Qualitätsbewertung effektiv nutzen zu können, bedarf es einer Interpretation der berechneten Werte. Die zur Qualitätsanalyse verwendeten Werkzeuge sehen eine Unterstützung dieser Interpretation durch Abbildung der Erfüllungsgrade von Faktoren (insbesondere der Qualitätsziele) auf eine Schulnotenskala sowie eine ampelähnliche Farbskala vor. Diese Abbildung erfordert eine einmalige Kalibrierung des Qualitätsmodells, einerseits hinsichtlich der in typischen Systemen zu erwartenden Wertebereiche jedes im Modell enthaltenen Maßes, andererseits hinsichtlich der Eignung dieser Maße, zwischen unterschiedlichen, subjektiv wahrgenommenen Qualitätsniveaus zu unterscheiden. Das Quamoco-Basismodell für Java wurde zur Kalibrierung der Maße auf zirka 6000 Softwareprodukte angewendet, um deren Wertebereiche mit großer Wahrscheinlichkeit einzugrenzen. Darüber hinaus wurden etwa 120 Softwareprodukte im Rahmen täglicher Analysen verwendet, um die Differenzierung zu optimieren. Schließlich wurden die Ergebnisse der Analyse von fünf Open-Source-Softwareprodukten mit den Ergebnissen manuell durchgeführter Qualitätsanalysen verglichen, um die Übereinstimmung der automatischen Analyse mit der durch Experten eingeschätzten Qualität dieser Produkte zu überprüfen [10, 11]. Generell haben Studien gezeigt, dass Analyse-Ergebnisse nur im Kontext einer Benchmark-Basis sinnvoll einzuschätzen sind [68]. Eine Möglichkeit für weitere Arbeiten stellt also die Automatisierung der Qualitätsbewertung in Verbindung mit der Schaffung einer solchen Benchmark-Basis dar. Dies würde nicht nur aus praktischer Sicht die Vergleichbarkeit SOA-basierter Systeme erhöhen, sondern im Rückschluss auf die im Qualitätsmodell zugrunde liegenden Konzepte weitere empirische Daten liefern, die für die wissenschaftliche Weiterentwicklung von Qualitätsmodellen im Allgemeinen von Bedeutung sein können.

Auf der konzeptuellen Ebene können in zukünftigen Arbeiten Qualitätsmodelle für weitere Architekturstile erarbeitet werden. Das SOA-Qualitätsmodell kann dabei insofern als strukturelle Vorlage dienen, dass grundsätzlich zwischen Prinzipien des Architekturstils und deren Ausprägung in beobachtbaren Eigenschaften auf der einen und weiteren, für die Qualität eines Softwaresystems relevanten Faktoren auf der an-

deren Seite unterschieden wird. Auf diese Weise könnten Architekturstile anhand ihrer Einflüsse auf die Softwarequalität miteinander verglichen werden. Damit würde es möglich, basierend auf Qualitätsmodellen unterschiedlicher Architekturstile für ein konkretes Projekt denjenigen zu identifizieren, dessen positive Auswirkungen auf die Qualität der Software am besten die für das Projekt definierten Qualitätsanforderungen abbilden. Es entstünde folglich ein Rahmenwerk für die Beschreibung von Architekturstilen aus Qualitätssicht, welches als Entscheidungsgrundlage in den frühen Phasen der Softwareentwicklung eingesetzt werden könnte.

Abbildungsverzeichnis

Tabellenverzeichnis

Verzeichnis der Qualitätsziele

Verzeichnis der Faktoren

Verzeichnis der Maße

Eigene Publikationen

Im Rahmen der Arbeiten an dieser Dissertation wurden folgende Beiträge zu Workshops, Konferenzen und wissenschaftlichen Zeitschriften verfasst, deren Inhalt in Teilen hier verwendet wurde.

[1] Sebastian Döweling, Benedikt Schmidt und Andreas Göb. „A model for the design of interactive systems based on activity theory“. In: *Proc. ACM 2012 Conf. Computer Supported Cooperative Work (CSCW)*. 2012, S. 539–548.

[2] Andreas Göb. „A Meta Model for Software Architecture Conformance and Quality Assessment“. In: *Electronic Communications of the EASST* 60 (2013).

[3] Andreas Göb und Klaus Lochmann. „A software quality model for SOA“. In: *Proc. 8th Int. Workshop Software Quality (WoSQ)*. 2011, S. 18–25.

[4] Andreas Göb, Daniel Schreiber, Louenas Hamdi, Erwin Aitenbichler und Max Mühlhäuser. „Reducing User Perceived Latency with a Middleware for Mobile SOA Access“. In: *Proc. 2009 IEEE Int. Conf. Web Services (ICWS)*. 2009, S. 366–373.

[5] Klaus Lochmann und Andreas Göb. „A Unifying Model for Software Quality“. In: *Proc. 8th Int. Workshop Software Quality (WoSQ)*. 2011, S. 3–10.

[6] Klaus Lochmann, Stefan Wagner, Andreas Göb und Dominik Kirchler. „Categorization of Software Quality Patterns“. In: *Tagungsband 3. Workshop zur Software-Qualitätsmodellierung und -bewertung (SQMB)*. 2010, S. 40–49.

[7] Daniel Schreiber, Erwin Aitenbichler, Andreas Göb und Max Mühlhäuser. „Reducing User Perceived Latency in Mobile Processes“. In: *Proc. 2010 IEEE Int. Conf. Web Services (ICWS)*. 2010, S. 235–242.

[8] Daniel Schreiber, Andreas Göb, Erwin Aitenbichler und Max Mühlhäuser. „Reducing User Perceived Latency with a Proactive Prefetching Middleware for Mobile SOA Access“. In: *International Journal of Web Services Research* 8.1 (2011), S. 68–85.

[9] Dirk Voelz und Andreas Göb. „What is Different in Quality Management for SOA?“ In: *Proc. 14th IEEE Int. Enterprise Distributed Object Computing Conf. (EDOC)*. 2010, S. 47–56.

[10] Stefan Wagner, Klaus Lochmann, Lars Heinemann, Michael Kläs, Andreas Seidl, Andreas Göb, Jonathan Streit, Adam Trendowicz und Reinhold Plösch. „The Quamoco Product Quality Modelling and Assessment Approach“. In: *Proc. 34th Int. Conf. Software Engineering (ICSE)*. 2012.

[11] Stefan Wagner, Klaus Lochmann, Lars Heinemann, Michael Kläs, Adam Trendowicz, Reinhold Plösch, Alois Mayr, Andreas Seidl, Andreas Göb und Jonathan Streit. „Practical Product Quality Modelling and Assessment: The Quamoco Approach“. In: *IEEE Transactions on Software Engineering* (2012). Eingereicht.

[12] Stefan Wagner, Klaus Lochmann, Sebastian Winter, Florian Deissenboeck, Elmar Juergens, Markus Herrmannsdoerfer, Lars Heinemann, Michael Kläs, Adam Trendowicz, Jens Heidrich, Reinhold Plösch, Andreas Göb, Christian Körner, Korbinian Schoder und Christian Schubert. *The Quamoco Quality Meta-Model*. Techn. Ber. TUM-I128. Technische Universität München, 2012.

[13] Stefan Wagner, Klaus Lochmann, Sebastian Winter, Andreas Göb und Michael Kläs. „Quality models in practice: A preliminary analysis“. In: *Proc. 3rd Int. Symp. Empirical Software Engineering and Measurement (ESEM)*. 2009, S. 464–467.

[14] Stefan Wagner, Klaus Lochmann, Sebastian Winter, Andreas Göb, Michael Kläs und Sabine Nunnenmacher. *Software Quality Models in Practice*. Techn. Ber. TUM-I129. Technische Universität München, 2012.

Literaturverzeichnis

[15] James S. Ackerman, Peter Collins, Alan Gowans und Roger Scruton. *Architecture*. 2011. URL: `http://www.britannica.com/EBchecked/topic/32876/architecture`.

[16] Keith Andrews und Helmut Heidegger. „Information Slices: Visualising and Exploring Large Hierarchies using Cascading, Semi-Circular Discs“. In: *Proc. IEEE Symp. Information Visualization (InfoVis)*. 1998, S. 4–7.

[17] Erik Arisholm. „Empirical assessment of the impact of structural properties on the changeability of object-oriented software“. In: *Information and Software Technology* 48.11 (2006), S. 1046–1055.

[18] Erik Arisholm, Lionel C. Briand, Siw Elisabeth Hove und Yvan Labiche. „The impact of UML documentation on software maintenance: an experimental evaluation“. In: *IEEE Transactions on Software Engineering* 32.6 (2006), S. 365–381.

[19] Ali Arsanjani, Liang-Jie Zhang, Michael Ellis, Abdul Allam und Kishore Channabasavaiah. „S3: A Service-Oriented Reference Architecture“. In: *IT Professional* 9.3 (2007), S. 10–17.

[20] Daniel Bachlechner, Katharina Siorpaes, Holger Lausen und Dieter Fensel. „Web service discovery-a reality check“. In: *Proc. 3rd European Semantic Web Conf. (ESWC)*. Bd. 308. 2006.

[21] Zain Balfagih und Mohd Fadzil Hassan. „Quality Model for Web Services from Multi-stakeholders' Perspective“. In: *Proc. 2009 Int. Conf. Information Management and Engineering*. 2009, S. 287–291.

[22] Jagdish Bansiya und Carl G. Davis. „A hierarchical model for object-oriented design quality assessment“. In: *IEEE Transactions on Software Engineering* 28.1 (2002), S. 4–17.

[23] Judith Barnard. „A new reusability metric for object-oriented software“. In: *Software Quality Journal* 7.1 (1998), S. 35–50.

[24] Victor R. Basili und David M. Weiss. „A Methodology for Collecting Valid Software Engineering Data“. In: *IEEE Transactions on Software Engineering* 10.6 (1984), S. 728–738.

[25] Len Bass, Paul Clements und Rick Kazman. *Software Architecture in Practice*. 2. Aufl. Boston: Addison-Wesley Professional, 2003.

[26] M. Brian Blake. „Decomposing Composition: Service-Oriented Software Engineers“. In: *IEEE Software* 24.6 (2007), S. 68–77.

[27] Barry W. Boehm. *Software Engineering Economics*. Prentice Hall, 1981.

[28] Barry W. Boehm, John R. Brown, Hans Kaspar, Myron Lipow, Gordon J. Macleod und Michael J. Merrit. *Characteristics of Software Quality*. North-Holland, 1978.

[29] Barry W. Boehm, Bradford Clark, Ellis Horowitz, Chris Westland, Ray Madachy und Richard W. Selby. „Cost models for future software life cycle processes: COCOMO 2.0“. In: *Annals of Software Engineering* 1.1 (1995), S. 57–94.

[30] L.C. Briand, J. Wüst, J.W. Daly und D. Victor Porter. „Exploring the relationships between design measures and software quality in object-oriented systems“. In: *Journal of Systems and Software* 51.3 (2000), S. 245–273.

[31] Lionel C. Briand, Khaled El Emam, Bernd G. Freimut und Oliver Laitenberger. „A comprehensive evaluation of capture-recapture models for estimating software defect content“. In: *IEEE Transactions on Software Engineering* 26.6 (2000), S. 518–540.

[32] Manfred Broy, Florian Deißenböck und Markus Pizka. „Demystifying Maintainability“. In: *Proc. 4th Int. Workshop Software Quality (WoSQ)*. 2006, S. 21–26.

[33] Steve Burbeck. *The Tao of e-business services*. 2000. URL: `http://www.ibm.com/developerworks/library/ws-tao/`.

[34] Kenneth P. Burnham und W. S. Overton. „Estimation of the Size of a Closed Population when Capture Probabilities vary Among Animals“. In: *Biometrika* 65.3 (1978), S. 625–633.

[35] Kai-Yuan Cai und David Card. „An analysis of research topics in software engineering - 2006“. In: *Journal of Systems and Software* 81.6 (2008), S. 1051–1058.

[36] Michael J. Carey. „SOA What?“ In: *IEEE Computer* 41.3 (2008), S. 92–94.

[37] Valentina Casola, Anna Rita Fasolino, Nicola Mazzocca und Porfirio Tramontana. „A framework for evaluating Quality and Security in Service Oriented Architectures“. In: *Proc. 2007 IEEE Int. Conf. Web Services (ICWS)*. 2007, S. 118–119.

[38] Anne Chao. „Estimating Population Size for Sparse Data in Capture-Recapture Experiments“. In: *Biometrics* 45.2 (1989), S. 427–438.

[39] J. Y. Chen und J. F. Lu. „A new metric for object-oriented design“. In: *Information and Software Technology* 35.4 (1993), S. 232–240.

[40] Jie-Cherng Chen und Sun-Jen Huang. „An empirical analysis of the impact of software development problem factors on software maintainability“. In: *Journal of Systems and Software* 82.6 (2009), S. 981–992.

[41] Si Won Choi, Jin Sun Her und Soo Dong Kim. „Modeling QoS Attributes and Metrics for Evaluating Services in SOA Considering Consumers' Perspective as the First Class Requirement“. In: *Proc. 2nd IEEE Asia-Pacific Service Computing Conf. (APSCC)*. 2007, S. 398–405.

[42] Paul Clements, Rick Kazman und Mark Klein. *Evaluating Software Architectures: Methods and Case Studies*. Addison-Wesley Professional, 2001.

[43] Francisco Curbera. „Component Contracts in Service-Oriented Architectures“. In: *IEEE Computer* 40.11 (2007), S. 74–80.

[44] Leigh A. Davis, Rose F. Gamble und Jamie Payton. „The impact of component architectures on interoperability“. In: *Journal of Systems and Software* 61.1 (2002), S. 31–45.

[45] Florian Deißenböck, Lars Heinemann, Markus Herrmannsdörfer, Klaus Lochmann und Stefan Wagner. „The quamoco tool chain for quality modeling and assessment“. In: *Proc. 33rd Int. Conf. Software Engineering (ICSE)*. 2011, S. 1007–1009.

[46] Florian Deißenböck, Elmar Jürgens, Benjamin Hummel, Stefan Wagner, Benedikt Mas y Parareda und Markus Pizka. „Tool Support for Continuous Quality Control“. In: *IEEE Software* 25.5 (2008), S. 60–67.

[47] Florian Deißenböck, Elmar Jürgens, Klaus Lochmann und Stefan Wagner. „Software Quality Models: Purposes, Usage Scenarios and Requirements“. In: *Proc. 7th Int. Workshop Software Quality (WoSQ)*. 2009, S. 9–14.

[48] Florian Deißenböck und Markus Pizka. „Concise and consistent naming“. In: *Software Quality Journal* 14.3 (2006), S. 261–282.

[49] Florian Deißenböck, Stefan Wagner, Markus Pizka, Stefan Teuchert und Jean-Francois Girard. „An Activity-Based Quality Model for Maintainability“. In: *Proc. IEEE Int. Conf. Software Maintenance (ICSM)*. 2007, S. 184–193.

[50] Ignatios Deligiannis, Martin Shepperd, Manos Roumeliotis und Ioannis Stamelos. „An empirical investigation of an object-oriented design heuristic for maintainability“. In: *Journal of Systems and Software* 65.2 (2003), S. 127–139.

[51] Ignatios Deligiannis, Ioannis Stamelos, Lefteris Angelis, Manos Roumeliotis und Martin Shepperd. „A controlled experiment investigation of an object-oriented design heuristic for maintainability“. In: *Journal of Systems and Software* 72.2 (2004), S. 129–143.

[52] R. Geoff Dromey. „A Model for Software Product Quality“. In: *IEEE Transactions on Software Engineering* 21.2 (1995), S. 146–162.

[53] Stéphane Ducasse und Damien Pollet. „Software Architecture Reconstruction: A Process-Oriented Taxonomy“. In: *IEEE Transactions on Software Engineering* 35.4 (2009), S. 573–591.

[54] Thomas Erl. *Service-Oriented Architecture: Concepts, Technology, and Design*. Prentice Hall International, 2005.

[55] Thomas Erl. *SOA: Entwurfsprinzipien für service-orientierte Architektur*. Addison-Wesley, 2008.

[56] Thomas Erl. *SOA: Principles of Service Design*. Prentice Hall, 2007.

[57] Letha H. Etzkorn, William E. Hughes und Carl G. Davis. „Automated reusability quality analysis of OO legacy software“. In: *Information and Software Technology* 43.5 (2001), S. 295–308.

[58] Jose Luiz Fiadeiro. „Designing for Software's Social Complexity“. In: *IEEE Computer* 40.1 (2007), S. 34–39.

[59] Andrew Forward und Timothy C. Lethbridge. „The relevance of software documentation, tools and technologies: a survey“. In: *Proc. 2002 ACM Symp. Document Engineering (DocEng)*. 2002, S. 26–33.

[60] Martin Fowler. *Patterns of Enterprise Application Architecture*. Addison-Wesley, 2002.

[61] Jean-Pierre Garbani. *Future Trends In The Enterprise Software Market*. Techn. Ber. E-RES53493. Forrester Research, Inc., 2009.

[62] David A. Garvin. „What Does “Product Quality” Really Mean?“ In: *MIT Sloan Management Review* 26.1 (1984), S. 25–43.

[63] Robert Geist, Madhu Chetuparambil, Stephen Hedetniemi und A. Joe Turner. „Computing research programs in the U.S.“ In: *Communications of the ACM* 39.12 (1996), S. 96–99.

[64] Robert L. Glass, Venkataraman Ramesh und Iris Vessey. „An analysis of research in computing disciplines“. In: *Communications of the ACM* 47.6 (2004), S. 89–94.

[65] Robert B. Grady. *Practical Software Metrics for Project Management and Process Improvement*. Prentice Hall, 1992.

[66] Juan Carlos Granja-Alvarez und Manuel José Barranco-García. „A method for estimating maintenance cost in a software project: a case study“. In: *Journal of Software Maintenance: Research and Practice* 9.3 (1997), S. 161–175.

[67] Harald Gruber. „Benchmarking-oriented Assessment of Source Code Quality“. Diss. Johannes Kepler Universität Linz, 2010.

[68] Harald Gruber, Reinhold Plösch und Matthias Saft. „On the validity of benchmarking for evaluating code quality“. In: *Proc. Int. Conf. Software Measurement IWSM/MetriKon/Mensura*. 2010.

[69] Roger Heutschi. *Serviceorientierte Architektur*. Springer-Verlag Berlin Heidelberg, 2007.

[70] Jared T. Howerton. „Service-Oriented Architecture and Web 2.0“. In: *IT Professional* 9.3 (2007), S. 62–64.

[71] Michael N. Huhns und Munindar P. Singh. „Service-oriented computing: key concepts and principles“. In: *IEEE Internet Computing* 9.1 (2005), S. 75–81.

[72] Watts S. Humphrey. *The Watts New Collection: Columns by the SEI's Watts Humphrey*. Techn. Ber. CMU/SEI-2009-SR-024. Software Engineering Institute, Carnegie Mellon University, Pittsburgh, Pennsylvania, 2009.

[73] Jens Hündling. „Modellierung von Qualitätsmerkmalen für Services". Diss. Universität Potsdam, 2008.

[74] IEEE. *Std 1074-2006 – IEEE Standard for Developing a Software Project Life Cycle Process*. Standard. 2006.

[75] ISO. *ISO/IEC 14598: Software engineering – Product evaluation*. Norm. 1999.

[76] ISO. *ISO/IEC 25000:2005 Software Engineering – Software product Quality Requirements and Evaluation (SQuaRE) – Guide to SQuaRE*. Norm. 2005.

[77] ISO. *ISO/IEC 25010:2011 Systems and software engineering – Systems and software Quality Requirements and Evaluation (SQuaRE) – System and software quality models*. Norm. 2011.

[78] ISO/IEC. *ISO/IEC 9126-4: Software engineering – Product quality – Part 4: Quality in use metrics*. Norm. 2004.

[79] Steve Jones. „Toward an acceptable definition of service [service-oriented architecture]". In: *IEEE Software* 22.3 (2005), S. 87–93.

[80] Nicolai Josuttis. *SOA in Practice: The Art of Distributed System Design*. O'Reilly Media, Inc., 2007.

[81] Nicolai Josuttis. „SOA und Performance". In: *SOA-Expertenwissen*. Hrsg. von Gernot Starke und Stefan Tilkov. dpunkt Verlag, 2007. Kap. 22, S. 381–388.

[82] Ho-Won Jung, Seung-Gweon Kim und Chang-Shin Chung. „Measuring Software Product Quality: A Survey of ISO/IEC 9126". In: *IEEE Software* 21.5 (2004), S. 88–92.

[83] Stephen H. Kan. *Metrics and Models in Software Quality Engineering*. 2. Aufl. Addison-Wesley, 2002.

[84] Sungwon Kang, Seonah Lee und Danhyung Lee. „A Framework for Tool-Based Software Architecture Reconstruction". In: *International Journal of Software Engineering and Knowledge Engineering* 19.2 (2009), S. 283–305.

[85] Lars M. Karg, Michael Grottke und Arne Beckhaus. „A Systematic Literature Review of Software Quality Cost Research". In: *Journal of Systems and Software* 84.3 (2011), S. 415–427.

[86] Rick Kazman und S. Jeromy Carrière. „Playing Detective: Reconstructing Software Architecture from Available Evidence". In: *Automated Software Engineering* 6.2 (1999), S. 107–138.

[87] Su Myeon Kim und Marcel Catalin Rosu. „A survey of public web services". In: *Proc. 13th Int. Conf. World Wide Web Alternate track papers & posters (WWW Alt.)* 2004, S. 312–313.

[88] Barbara Kitchenham und Shari Lawrence Pfleeger. „Software Quality: The Elusive Target“. In: *IEEE Software* 13.1 (1996), S. 12–21.

[89] Andreas Knöpfel, Bernhard Gröne und Peter Tabeling. *Fundamental Modeling Concepts: Effective Communication of IT Systems*. Wiley, 2006.

[90] Dirk Krafzig, Karl Banke und Dirk Slama. *Enterprise SOA: Service-Oriented Architecture Best Practices*. Prentice Hall, 2004.

[91] James P. Lawler und Hortense Howell-Barber. *Service-oriented Architecture - SOA Strategy, Methodology, and Technology*. Auerbach Publications, 2008.

[92] Wei Li und Sallie Henry. „Object-oriented metrics that predict maintainability“. In: *Journal of Systems and Software* 23.2 (1993), S. 111–122.

[93] Klaus Lochmann und Lars Heinemann. „Integrating quality models and static analysis for comprehensive quality assessment“. In: *Proc. 2nd Int. Workshop Emerging Trends in Software Metrics (WETSoM)*. 2011, S. 5–11.

[94] Chung-Horng Lung und Kalai Kalaichelvan. „An Approach to Quantitative Software Architecture Sensitivity Analysis“. In: *International Journal of Software Engineering and Knowledge Engineering* 10.1 (2000), S. 97–114.

[95] Eric A. Marks und Michael Bell. *Service Oriented Architecture (SOA): A Planning and Implementation Guide for Business and Technology*. Bd. 2008. John Wiley & Sons, 2008.

[96] Dieter Masak. *Moderne Enterprise Architekturen*. Springer-Verlag Berlin Heidelberg, 2005.

[97] Eyhab Al-Masri und Qusay H. Mahmoud. „Investigating web services on the world wide web“. In: *Proc. 17th Int. Conf. World Wide Web (WWW)*. 2008, S. 795–804.

[98] Alois Mayr, Reinhold Plösch, Michael Kläs, Constanza Lampasona und Matthias Saft. „A Comprehensive Code-based Quality Model for Embedded Systems – Systematic Development and Validation by Industrial Projects“. In: *Proc. 23rd Int. Symp. Software Reliability Engineering (ISSRE)*. 2012.

[99] Kevin McArthur, Hossein Saiedian und Mansour Zand. „An evaluation of the impact of component-based architectures on software reusability“. In: *Information and Software Technology* 44.6 (2002), S. 351–359.

[100] Jim A. McCall, Paul K. Richards und Gene F. Walters. *Factors in Software Quality*. Techn. Ber. RADC-TR-77-369. Rome Air Development Center, Air Force Systems Command, 1977.

[101] Ben Swarup Medikonda und Seetha Ramaiah Panchumarthy. „An Approach to Modeling Software Safety“. In: *Proc. 9th ACIS Int. Conf. Software Engineering, Artificial Intelligence, Networking, and Parallel/Distributed Computing (SNPD)*. 2008, S. 800–806.

[102] Daniel Méndez Fernández und Marco Kuhrmann. *Artefact-based Requirements Engineering and its Integration into a Process Framework*. Techn. Ber. TUM-I0929. Technische Universität München, 2009.

[103] Anton Michlmayr, Florian Rosenberg, Christian Platzer, Martin Treiber und Schahram Dustdar. „Towards recovering the broken SOA triangle“. In: *Proc. 2nd Int. Workshop Service Oriented Software Engineering (IW-SOSWE)*. 2007, S. 22–28.

[104] Subhas Chandra Misra. „Modeling design/coding factors that drive maintainability of software systems“. In: *Software Quality Journal* 13.3 (2005), S. 297–320.

[105] Karine Mordal-Manet, Francoise Balmas, Simon Denier, Stéphane Ducasse, Harald Wertz, Jannik Laval, Fabrice Bellingard und Philippe Vaillergues. „The squale model - A practice-based industrial quality model“. In: *Proc. IEEE Int. Conf. Software Maintenance (ICSM)*. 2009, S. 531–534.

[106] Samar Mouchawrab, Lionel C. Briand und Yvan Labiche. „A measurement framework for object-oriented software testability“. In: *Information and Software Technology* 47.15 (2005), S. 979–997.

[107] Eric Newcomer und Greg Lomow. *Understanding SOA with Web Services*. Addison-Wesley, 2004.

[108] Volker Nissen, Mathias Petsch und Hagen Schorcht, Hrsg. *Service-orientierte Architekturen - Chancen und Herausforderungen bei der Flexibilisierung und Integration von Unternehmensprozessen*. Deutscher Universitäts-Verlag, 2007.

[109] OASIS. *Quality Model for Web Services, Working Draft*. Techn. Ber. WSQM2.0. Organization for the Advancement of Structured Information Standards (OASIS), 2005.

[110] Object Management Group (OMG). *OMG Meta Object Facility (MOF) Core Specification*. 2011. URL: `http://www.omg.org/spec/MOF/2.4.1`.

[111] Object Management Group (OMG). *Service oriented architecture Modeling Language (SoaML) Specification*. 2012. URL: `http://www.omg.org/spec/SoaML/1.0.1`.

[112] Liam O'Brien, Len Bass und Paulo Merson. *Quality Attributes and Service-Oriented Architectures*. Techn. Ber. CMU/SEI-2005-TN-014. Software Engineering Institute, Carnegie Mellon University, Pittsburgh, Pennsylvania, 2005.

[113] Liam O'Brien, Paulo Merson und Len Bass. „Quality Attributes for Service-Oriented Architectures“. In: *Proc. Int. Workshop Systems Development in SOA Environments (SDSOA)*. 2007.

[114] David L. Otis, Kenneth P. Burnham, Gary C. White und David R. Anderson. „Statistical Inference From Capture Data on Closed Animal Populations“. In: *Wildlife Monographs* 62 (1978), S. 3–135.

[115] Claus Pahl, Yaoling Zhu und Veronica Gacitua-Decar. „A Template-Driven Approach for Maintainable Service-Oriented Information Systems Integration“. In: *International Journal of Software Engineering and Knowledge Engineering* 19.7 (2009), S. 889–912.

[116] Michael P. Papazoglou, Paolo Traverso, Schahram Dustdar und Frank Leymann. „Service-Oriented Computing: State of the Art and Research Challenges“. In: *IEEE Computer* 40.11 (2007), S. 38–45.

[117] David L. Parnas und David M. Weiss. „Active design reviews: principles and practices“. In: *Proc. 8th Int. Conf. Software Engineering (ICSE)*. 1985, S. 132–136.

[118] Mikhail Perepletchikov, Caspar Ryan und Zahir Tari. „The Impact of Service Cohesion on the Analyzability of Service-Oriented Software“. In: *IEEE Transactions on Services Computing* 3.2 (2010), S. 89–103.

[119] Dewayne E. Perry und Alexander L. Wolf. „Foundations for the study of software architecture“. In: *ACM SIGSOFT Software Engineering Notes* 17.4 (1992), S. 40–52.

[120] Håkan Petersson, Thomas Thelin, Per Runeson und Claes Wohlin. „Capture–recapture in software inspections after 10 years research—theory, evaluation and application“. In: *Journal of Systems and Software* 72.2 (2004), S. 249–264.

[121] Vinay Kumar Reddy, Alpana Dubey, Sala Lakshmanan, Srihari Sukumaran und Rajendra Sisodia. „Evaluating legacy assets in the context of migration to SOA“. In: *Software Quality Journal* 17.1 (2009), S. 51–63.

[122] A.T. Rivers und M.A. Vouk. „Resource-constrained non-operational testing of software“. In: *Proc. 9th Int. Symp. Software Reliability Engineering*. 1998, S. 154–163.

[123] Dmytro Rud. „Performancebewertung und -sicherung von orchestrierten Serviceangeboten“. Diss. Universität Magdeburg, 2009.

[124] SAP AG. *Standardized Technical Architecture Modeling*. Techn. Ber. 2007.

[125] Bradley Schmerl, Jonathan Aldrich, David Garlan, Rick Kazman und Hong Yan. „Discovering Architectures from Running Systems“. In: *IEEE Transactions on Software Engineering* 32.7 (2006), S. 454–466.

[126] Arun Sharma, Rajesh Kumar und P. S. Grover. „Empirical Evaluation and Validation of Interface Complexity Metrics for Software Components“. In: *International Journal of Software Engineering and Knowledge Engineering* 18.7 (2008), S. 919–931.

[127] Martin Shepperd und Darrel Ince. „Design metrics and software maintainability: An experimental investigation“. In: *Journal of Software Maintenance: Research and Practice* 3.4 (1991), S. 215–232.

[128] Bingu Shim, Siho Choue, Suntae Kim und Sooyong Park. „A Design Quality Model for Service-Oriented Architecture“. In: *Proc. 15th Asia-Pacific Software Engineering Conf. (APSEC)*. 2008, S. 403–410.

[129] Frank Simon, Olaf Seng und Thomas Mohaupt. *Code Quality Management - Technische Qualität industrieller Softwaresysteme transparent und vergleichbar gemacht*. Dpunkt.Verlag GmbH, 2006.

[130] Michael Stal. „Using architectural patterns and blueprints for service-oriented architecture“. In: *IEEE Software* 23.2 (2006), S. 54–61.

[131] Wayne P. Stevens, Glenford J. Myers und Larry L. Constantine. „Structured design“. In: *IBM Systems Journal* 13.2 (1974), S. 115–139.

[132] Mikael Svahnberg und Claes Wohlin. „An investigation of a method for identifying a software architecture candidate with respect to quality attributes“. In: *Empirical Software Engineering* 10.2 (2005), S. 149–181.

[133] The European Network of Excellence in Software Services and Systems (S-Cube). *Quality Reference Model for SBA*. Techn. Ber. CD-JRA-1.3.2. S-Cube, 2008.

[134] The Open Group. *Service-Oriented Architecture Ontology*. Techn. Ber. C104. 2010.

[135] Eirik Tryggeseth. „Report from an Experiment: Impact of Documentation on Maintenance“. In: *Empirical Software Engineering* 2.2 (1997), S. 201–207.

[136] Wei-Tek Tsai, Xinyu Zhou, Yinong Chen und Xiaoying Bai. „On Testing and Evaluating Service-Oriented Software“. In: *IEEE Computer* 41.8 (2008), S. 40–46.

[137] Sira Vegas und Victor R. Basili. „A Characterisation Schema for Software Testing Techniques“. In: *Empirical Software Engineering* 10.4 (2005), S. 437–466.

[138] Oliver Vogel, Ingo Arnold, Arif Chughtai, Edmund Ihler, Timo Kehrer, Uwe Mehlig und Uwe Zdun. *Software-Architektur: Grundlagen - Konzepte - Praxis*. 2. Aufl. Spektrum Akademischer Verlag Heidelberg, 2009.

[139] W3C. *Web Services Architecture*. W3C Working Group Note. 2004. URL: `http://www.w3.org/TR/2004/NOTE-ws-arch-20040211/`.

[140] W3C. *Web Services Glossary*. W3C Working Group Note. 2004. URL: `http://www.w3.org/TR/2004/NOTE-ws-gloss-20040211/`.

[141] Stefan Wagner, Daniel Mendez Fernandez, Shareeful Islam und Klaus Lochmann. „A Security Requirements Approach for Web Systems“. In: *Proc. Workshop Quality Assessment in Web (QAW 2009)*. 2009.

[142] Mohamed El-Wakil, Ali El-Bastawisi, Mokhtar Boshra und Ali Fahmy. „Object-Oriented Design Quality Models A Survey and Comparison“. In: *Proc. Int. Conf. Informatics and Systems (INFOS)*. 2004.

[143] Ricardo Wickel. „Agile Architekturmodellierung am Rande der UML“. In: *OBJEKTspektrum* 3 (2012), S. 34–39.

[144] Sebastian Winter, Stefan Wagner und Florian Deißenböck. „A Comprehensive Model of Usability“. In: *Proc. Engineering Interactive Systems (EIS)*. Bd. 4940. Lecture Notes in Computer Science. 2007, S. 106–122.

[145] Claes Wohlin. „An analysis of the most cited articles in software engineering journals - 2000“. In: *Information and Software Technology* 49.1 (2007), S. 2–11.

[146] W. Eric Wong, T. H. Tse, Robert L. Glass, Victor R. Basili und Tsong Yueh Chen. „An assessment of systems and software engineering scholars and institutions (2002-2006)“. In: *Journal of Systems and Software* 82.8 (2009), S. 1370–1373.

[147] PengCheng Xiong, YuShun Fan und MengChu Zhou. „Web Service Configuration Under Multiple Quality-of-Service Attributes“. In: *IEEE Transactions on Automation Science and Engineering* 6.2 (2009), S. 311–321.

[148] John A. Zachman. „A framework for information systems architecture“. In: *IBM Systems Journal* 26.3 (1987), S. 276–292.

[149] Jianjun Zhao. „Bibliography of Software Architecture Analysis“. In: *Software Engineering Notes* 24.4 (1999), S. 61–62.