TECHNISCHE UNIVERSITÄT MÜNCHEN
FAKULTÄT FÜR INFORMATIK

Forschungs- und Lehreinheit XI
Angewandte Informatik / Kooperative Systeme

Integrierter Prozess zu Entwurf und Bewertung einer Cloud-basierten Softwarearchitektur

Holger N. Sirtl

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. Martin Bichler
Prüfer der Dissertation:	1. Univ.-Prof. Dr. Johann Schlichter
	2. Univ.-Prof. Dr. Florian Matthes

Die Dissertation wurde am 25.03.2015 bei der Technischen Universität München eingereicht und durch die Fakultät für Informatik am 25.07.2015 angenommen.

Abstract

Within only few years, cloud computing transformed from a pure hype-phenomenon into a real option for operations and procurement of IT resources. Among other factors it promises increased flexibility in its use and a corresponding optimization in operational cost. The use of related integration technologies allows hybrid-approaches with only parts of a system being executed in the cloud. During the design process of such a software system this option should be considered and evaluated for each component according to requirements of different stakeholders. This assessment should be carried out as objective as possible.

Designing a software system is a complex process in context of limited resources on one side and expectations and requirements of different stakeholders on the other. In case these stakeholders favor certain personal concerns over the common goal of developing a system that maximizes utility for all, irrational influences can arise restraining an orderly, comprehensible decision process. Thus, even though there are various methods for certain design steps that aim to rationalize these steps, this goal can be jeopardized by irrational behavior of certain interest groups.

During the design process not only the best decisions should be taken, but it should also be documented on what basis the assessment and choice of one specific alternative was done. Thus, there is an explicit need for modeling alternatives, where each one of them fulfils the requirements associated to the corresponding subsystem. The architecture description shall support identifying the one with highest "return-on-investment" under given prioritized requirements and expected cost.

The work presented in this thesis is based on preliminary work in two central aspects, merges these and derives an integrated development- and evaluation process for cloud-based software architectures: firstly it is based on well-established process models for developing and evaluating software architectures developed at Carnegie Mellon University: the Architecture Driven Design Method (ADD) for iteratively developing an architecture description, the Architecture Tradeoff Analysis Method (ATAM) for analyzing architecture alternatives in terms of different quality attributes and the Cost Benefit Analysis Method (CBAM) for valuating overall benefit in relation to implementation effort. These process models describe the process steps but not the formal notation of necessary artefacts (requirements, architecture elements, etc.). For filling this gap this thesis is also based on preliminary work done at University of Stuttgart. [Mie10] introduces a meta-model for describing software architectures. This model

not only provides a notation framework for software architectures but also allows considering design alternatives. However the focus lies only on describing a final architecture model, but not on the process of deriving it.

Both approaches – development and evaluation methodology on the one side, notation methodology on the other side – are merged into one integrated process in the work presented in this thesis. In a first step, each methodology mentioned above is supplemented by a formal model for describing artifacts (requirements, architecture components, etc.) being used. Then, these methodologies are merged into an integrated formal approach for designing the architecture, and analyzing and evaluating architecture elements and alternatives. This approach includes all process steps of the individual methods. Artifacts like functional and quality requirements are formalized and standardized allowing for reusing them in all process steps as applicable. Input of the overall process consists of functional and quality requirements and design constraints. Output consists of a formal description of the architecture including possible inherent alternatives (e.g. use of a cloud-based database system versus an on-premises database system), its assessment and documentation of decisions. These results can be basis for subsequent system development steps like detailed design and implementation.

The process not only supports the architecture design but also evaluation of architecture alternatives and documentation of decisions being taken during design process. Even though irrational behavior cannot be eliminated, decisions become transparent, reproducible and verifiable in respect of their consequences for the benefit of the overall system.

Kurzfassung

Cloud Computing hat sich innerhalb weniger Jahre von einem reinen Hype-Begriff zu einer realen Option für Betrieb und Beschaffung von IT-Ressourcen entwickelt. Verspricht es doch unter anderem eine erhöhte Flexibilität bei deren Einsatz und eine entsprechende Optimierung der Betriebskosten. Durch vorhandene Integrationstechnologien ist es möglich, im Rahmen von Hybrid-Lösungen auch nur einzelne Teile eines Softwaresystems in der Cloud zu betreiben. Beim Entwurf eines Softwaresystems sollte diese Option bei jeder Komponente des Systems betrachtet und entsprechend den Anforderungen verschiedener Stakeholder, d.h. vom System betroffener Interessensgruppen, möglichst objektiv bewertet werden.

Der Entwurf eines Softwaresystems ist dabei ein komplexes Unterfangen, das im Spannungsfeld zwischen begrenzt zur Verfügung stehenden Ressourcen auf der einen und den Wünschen und Anforderungen verschiedener Interessensgruppen auf der anderen Seite stattfindet. Sofern diese bestimmte persönliche Interessen über das gemeinsame Ziel einer Entwicklung des für alle bestmöglichen Systems stellen, können sich irrationale Einflüsse ergeben, die einen geordneten, nachvollziehbaren Entscheidungsprozess behindern. D.h. wenngleich es für einzelne Entwurfsschritte Methoden gibt, die das Ziel haben, diese Schritte rational zu gestalten, können einzelne Interessensgruppen durch irrationales Verhalten diese Bestrebung konterkarieren.

Während des Entwurfsprozesses sollen nicht nur die jeweils besten Entscheidungen getroffen werden, sondern es soll nachvollziehbar sein, auf welcher Grundlage diese Bewertung erfolgte und welche Motivation hinter der Auswahl einer bestimmten Alternative steht. Es sollen im Entwurf also explizit Alternativen modelliert werden können, von denen jede alle die an das betreffende Subsystem gerichteten Anforderungen erfüllt. Der erstellte Architekturentwurf soll dabei unterstützen, bei gegebenen priorisierten Anforderungen und erwarteten Umsetzungsaufwänden die Alternativen mit dem besten „Return-on-Investment“ zu identifizieren.

Die vorliegende Arbeit setzt in zwei zentralen Aspekten auf Vorarbeiten auf, führt diese zusammen und leitet dann einen integrierten Entwurfs- und Bewertungsprozess für Cloud-basierte Softwarearchitekturen ab: zum ersten fußt sie auf bereits etablierten Vorgehensmodellen zum Entwurf und der Bewertung von Softwarearchitekturen der Carnegie Mellon Universität: die Attribute Driven Design Method (ADD) dient der iterativen Entwicklung einer Architekturbeschreibung, die Architecture Tradeoff Analysis Method (ATAM) der Analyse von

Entwurfsalternativen hinsichtlich verschiedener Qualitätsattribute und die Cost Benefit Analysis Method (CBAM) der Bewertung des Gesamtnutzens im Verhältnis zu den Implementierungsaufwänden. Die Vorgehensmodelle beschreiben die Prozessschritte, lassen jedoch weitgehend offen, wie die bearbeiteten Artefakte (Anforderungen, Architekturelemente etc.) formal beschrieben werden können. Deshalb setzt die Arbeit zum zweiten auf Vorarbeiten auf, die an der Universität Stuttgart durchgeführt wurden. In [Mie10] wird ein Metamodell zur Beschreibung von Softwarearchitekturen definiert. Dieses Modell liefert nicht nur Werkzeuge zur Notation einer Softwarearchitektur, sondern erlaubt auch die Berücksichtigung von Alternativen im Entwurf. Dabei liegt der Fokus in der genannten Arbeit nur auf der Beschreibung eines finalen Architekturmodells, nicht aber auf dem Prozess zu dessen Herleitung.

Die beiden Ansätze – Entwurfs- und Bewertungsmethodik auf der einen, Beschreibungsmethodik auf der anderen Seite – werden in der vorliegenden Arbeit zu einem integrierten Gesamtprozess zusammengeführt. Die oben erwähnten Methoden werden in dieser Arbeit zunächst einzeln mit einem formalen Modell zur Beschreibung der verwendeten Artefakte (Anforderungen, Architekturkomponenten etc.) unterlegt. Für die Durchführung des Architekturentwurfs und die einhergehende Analyse und Bewertung von Elementen und Alternativen der Architektur werden diese dann zu einem integrierten, formalen Prozess zusammengeführt. In diesem werden die zentralen Bearbeitungsschritte der Einzelmethoden durchgeführt, wobei die einzelnen Artefakte wie beispielsweise fachliche und qualitative Anforderungen sowie Designeinschränkungen vereinheitlicht sind und durchgehend Verwendung finden. Eingabe des Prozesses sind fachliche und qualitative Anforderungen sowie Designeinschränkungen. Ausgabe des Prozesses ist die formale Beschreibung der Architektur einschließlich etwaiger inhärenter Alternativen (z.B. Nutzung einer Cloud-basierten versus einer lokalen Datenbank), deren Bewertung und die Dokumentation von Entscheidungen. Damit ist die Grundlage für nachfolgende Schritte der Softwareentwicklung wie Detailentwurf und Implementierung gegeben.

Der Prozess hilft somit nicht nur beim Architekturentwurf selbst sondern auch bei der Bewertung von Architekturalternativen und der Dokumentation von Entscheidungen, die während des Entwurfsvorgangs getroffen wurden. Wenngleich irrationales Verhalten der Beteiligten dadurch nicht vermieden wird, können Entscheidungen damit transparent, nachvollziehbar und hinsichtlich ihrer Auswirkungen auf den Nutzen des Gesamtsystems überprüfbar gemacht werden.

Danksagung

An dieser Stelle möchte ich mich bei verschiedenen Personen bedanken, die alle einen wichtigen Beitrag zum erfolgreichen Abschluss dieser Arbeit geleistet haben.

An erster Stelle möchte ich meinem Doktorvater Herrn Prof. Dr. Johann Schlichter, Inhaber des Lehrstuhls XI: Angewandte Informatik - Kooperative Systeme der Technischen Universität München, meinen herzlichsten Dank aussprechen. Ich danke ihm für die Möglichkeit, diese Arbeit an seinem Lehrstuhl anzufertigen, dafür dass er sich stets Zeit für regelmäßige Abstimmungsgespräche genommen, mir immer mit Rat und Tat zur Seite gestanden und mir zugleich die kreative Freiheit für die inhaltliche Ausgestaltung gelassen hat.

Ebenso danken möchte ich Herrn Prof. Dr. Florian Matthes, Inhaber des Lehrstuhls XIX: Software Engineering betrieblicher Informationssysteme der Technischen Universität München, für seine Bereitschaft diese Arbeit als Zweitgutachter zu betreuen. Unsere fachlichen Gespräche waren äußerst konstruktiv und haben der Arbeit wertvolle Impulse gegeben.

Darüber hinaus bedanke ich mich bei meinen Kolleginnen und Kollegen des Lehrstuhls für ihre freundschaftliche Aufnahme in ihren Kreis und für die vielen interessanten und wertvollen Diskussionen zu verschiedenen Aspekten meiner Arbeit. Ihr Feedback auf zahlreichen Doktorandenseminaren war hochgeschätzt und äußerst wichtig für den Erfolg dieser Arbeit.

Bedanken möchte ich mich auch bei Herrn Dr. Said Zahedani, ehemals Senior Director und Mitglied der Geschäftsleitung der Microsoft Deutschland GmbH. Nicht zuletzt seiner Ermutigung und Unterstützung verdanke ich es, mich der Herausforderung dieser wissenschaftlichen Arbeit neben meinem beruflichen Alltag gestellt und diese erfolgreich zu Ende geführt zu haben.

Meine große Dankbarkeit gilt darüber hinaus meiner Familie, insbesondere meinen Eltern, für Ihre stetige Unterstützung und Ermutigung während meiner Arbeit sowie meiner über alles geliebten Freundin Silke für ihre Geduld, Unterstützung und ihr Verständnis.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Abkürzungsverzeichnis

AAlt	Architekturalternative
AD	Architecture Draft
ADD	Architecture Driven Design Method
ALMA	Architecture-Level Modifiability Analysis
ARID	Active Reviews for Intermediate Designs
ASR	Architecturally Significant Requirements
ATAM	Architecture Tradeoff Analysis Method
BM	Business Model
Café	Composite Application Framework
CBAM	Cost Benefit Analysis Method
CFML	Capability Focused Modeling Language
DCAR	Decision Centric Architecture Review
DoDAF	Department of Defense Architecture Framework
HPC	High Performance Computing
IaaS	Infrastructure-as-a-Service
IEEE	Institute of Electrical and Electronics Engineers
PaaS	Platform-as-a-Service
RbAD	Requirements based Architecture Design
RDBMS	Relational Database Management System
RM	Requirements Model
RiskM	Risk Model
ROI	Return-on-Investment
SAAM	Software Architecture Analysis Method
SaaS	Software-as-a-Service
SbRUA	Scenario based Risk- and Utility Assessment
SLA	Service Level Agreement
SM	Szenarienmodell
SOA	Service-orientierte Architektur
TOGAF	The Open Group Architecture Framework
UI	User Interface
UM	Utility Model
VP	Variabilitätspunkt

1 Einführung

Betriebsmodelle und Ausführungsumgebungen für softwareintensive Systeme haben im Laufe der Zeit eine Reihe von Entwicklungsschritten durchlaufen. Jeder dieser Schritte hatte einschneidende Auswirkungen auf die Architektur betreffender Software, indem er neue Möglichkeiten für die Verteilung und den optimalen Ausführungsort einzelner Anwendungskomponenten mit sich brachte. Mit Cloud Computing steht nun eine neue Alternative für den Bezug und Betrieb von Software zur Verfügung, die zum einen gänzlich neue Anwendungen ermöglichen, Hürden für die Einführung neuer Software deutlich absenken, und den Betrieb verteilter Komponenten effektiver und effizienter gestalten kann.

1.1 Evolution der Software-Betriebsmodelle

Charles Darwin, Enkel des berühmten britischen Naturforschers und Leiter des britischen National Physical Laboratory, wagte 1946 in einem Antrag zum Bau einer „Automatic Computing Engine (ACE)" [Dar46] die Prognose *„es ist gut möglich, dass diese eine Maschine ausreicht, alle aus dem gesamten Land an sie gerichteten Problemstellungen zu lösen"* (*„it is very possible that ... one machine would suffice to solve all the problems that are demanded of it from the whole country"* [Dar46]).

Nicht nur, dass sich diese Prognose innerhalb kurzer Zeit als falsch erwiesen hat, sondern – getrieben durch die stetig steigenden Anforderungen an Rechen- und Speicherleistung informationsverarbeitender Systeme – haben Computer über die Jahre eine Verbreitung erfahren, die mit Aufkommen des Cloud Computings einen jüngsten Höhepunkt erreicht hat, bei dem in Rechenzentren einzelner Anbieter mehrere hunderttausend Server im Einsatz sind. Entscheidende Treiber dieser Entwicklung waren fortschreitende Miniaturisierung bei gleichzeitig stark wachsender Leistungsfähigkeit von Prozessoren und Speicherbausteinen sowie zunehmender Vernetzungsgrad bei wachsenden Bandbreiten der beteiligten Netzwerke.

Nicholas Carr beschreibt in [Car08] die durch diese Phänomene durchlaufene Evolution der Software-Betriebsmodelle: Vor dem Hintergrund des Wandels von reinen Produktions- zu Dienstleistungsunternehmen wuchsen auch die Investitionen in IT-Systeme, die in Gestalt großer, monolithischer Mainframe-Systeme in den 60er und 70er Jahren vorherrschende Infrastruktur für die Ausführung für Software darstellten. Software war primär als Batch-Prozess angelegt. Diese wurden per Stapelverarbeitung in Form von Arbeitsaufträgen nacheinander abgearbeitet. Durch diese Art der Datenverarbeitung konnte eine hohe Auslastung der

Systeme erreicht und damit die hohen Anschaffungs- und Betriebskosten gerechtfertigt werden. Weiter fortschreitende Miniaturisierung und Herausbildung standardisierter Hardware-Komponenten ermöglichte die Produktion kleinerer und billigerer Minicomputer, die (mit den ersten höheren Programmiersprachen anstelle von Zahlenkolonnen) deutlich leichter programmiert werden konnten. Software war aber weiterhin primär für die Ausführung auf einem einzelnen System bestimmt. Mit der Entwicklung von Personal Computern – unter anderem ermöglicht durch die Erfindung des Mikroprozessors sowie Bestückung der Computer mit entsprechenden Betriebssystemen – wanderte die Rechenleistung von zentralen Systemen auf die Schreibtische der Anwender. Personal Computer konnten dabei vernetzt werden, um Daten auszutauschen und zentrale Ressourcen gemeinsam zu nutzen. Vernetzung, Verteilung von Programmlogik im lokalen Netz sowie die Herausbildung von Client- und Server-Rollen mussten in der Architektur von Softwaresystemen berücksichtigt werden. Um die Auslastung der Serversysteme zu optimieren, wurden schrittweise Virtualisierungstechnologien eingesetzt, die es erlaubten, mehrere logische Systeme virtualisiert auf einer physischen Hardware auszuführen, um damit die Ressourcennutzung der Hardware zu maximieren. Virtuelle Systeme wurden mobil, d.h. sie konnten leicht von einer physischen Umgebung auf eine andere umziehen. Auch dieser Umstand musste von Seiten der Softwarearchitektur berücksichtigt werden, da von konkreten Ausführungsumgebungen abstrahiert werden musste. Die Virtualisierungsansätze lebten von der Idee, dass sich Spitzenlasten der einzelnen virtuellen auf einer physischen Umgebung ausgleichen, und auch kleinere (virtuelle) Maschinen ohne Ressourcenverschwendung betrieben werden konnten.

Cloud Computing kann in diesem Zusammenhang als weitere Evolutionsstufe verstanden werden, bei dem der Virtualisierungsansatz auf die Spitze getrieben wird. Von einigen wenigen Anbietern werden dabei in großen Rechenzentren riesige Serverfarmen betrieben. Virtuelle Umgebungen können dort extrem schnell, kostengünstig und weitestgehend automatisiert bereitgestellt werden. Durch die Größe der Rechenzentren profitieren die Anbieter zudem von Skaleneffekten, die sich aus der Aufteilung von Betriebskosten (Räume, Klimatisierung, Stromversorgung etc.) auf viele Nutzer ergeben. Cloud Computing bietet gegenüber klassischen Formen des Softwarebetriebs unter anderem folgende Vorteile:

- Über Cloud Computing können Betriebsumgebungen für Softwarekomponenten sehr schnell bereitgestellt werden, was die Time-to-Market des Systems positiv beeinflusst.
- Der Betrieb von Softwarekomponenten in der Cloud kann in vielen Fällen deutlich kostengünstiger erfolgen, was sich positiv auf das Kosten-Nutzen-Verhältnis eines Softwaresystems auswirkt.
- Administrative Aufwände für Virtualisierung sowie die Verantwortung für den Lastausgleich verschiedener virtueller Umgebungen können an den Cloud Anbieter übertragen werden.

- Der hohe Automatisierungsgrad in Cloud Computing kann die Verwaltungskosten für das einzelne System beim Anwender deutlich senken.

1.2 Cloud Computing

Für Cloud Computing hat sich in der Literatur noch keine einheitliche Definition herausgebildet. [Gee09] enthält Antworten einiger Experten auf die Frage nach einer Definition dieses Begriffes. Häufig wird der Begriff normativ definiert, d.h. über bestimmte Eigenschaften, die man im Allgemeinen mit Cloud Computing verbindet. Armbrust u.a. identifizieren in [AFG09] folgende Aspekte, die neu für die Cloud sind:

- *Die Illusion unendlicher Rechenressourcen, die bei Bedarf zur Verfügung stehen* – damit vereinfachen sich Kapazitätsplanungen deutlich.

- *Die Ausschaltung von Vorab-Investitionen bei Cloud Anwendern* – die Menge der bereitgestellten Cloud Ressourcen lässt sich zu jedem Zeitpunkt flexibel an den jeweiligen Bedarf anpassen.

- *Die Möglichkeit, die Nutzung bereitgestellter Ressourcen kurzfristig und rein verbrauchsabhängig zu bezahlen* – damit wird es möglich, Kosten zu senken, indem nicht benötigte Ressourcen freigegeben und nur bei Bedarf wieder hinzugeschaltet werden.

Das Marktforschungsinstitut Gartner definiert in [Gar09] Cloud Computing als *„eine Form der Bereitstellung von Rechenleistung, bei der skalierbare und elastische IT-basierte Funktionalitäten als Dienst externen Kunden über Internet-Technologien verfügbar gemacht werden“*. Dabei werden folgende fünf Wesensmerkmale der Cloud identifiziert:

- *Serviceorientierung* – die innere Funktionsweise der bereitgestellten Ressourcen bleibt dem Anwender verborgen. Der Zugriff erfolgt über wohldefinierte Schnittstellen.

- *Skalierbarkeit und Elastizität* – die Menge der bereitgestellten Ressourcen lässt sich zur Laufzeit flexibel an den jeweiligen Bedarf anpassen.

- *Multi-Mandanten-Umgebungen* – Anbieter realisieren Skaleneffekte, indem sie viele Kunden zeitgleich über ihre Umgebungen bedienen. Durch Einsatz geeignete Virtualisierungstechnologien bleibt dieser Umstand vor jedem einzelnen Mandanten verborgen, der den Eindruck eines exklusiven Zugangs zu den Ressourcen hat.

- *Nutzungsabhängige Abrechnung* – neben verschiedenen alternativen Abrechnungsmodellen (Abonnement, Flatrates etc.) haben Nutzer auch die Option eines rein nutzungsabhängigen Kostenmodells.

- *Einsatz von Internet-Technologien* – beim Zugriff auf die bereitgestellten Funktionalitäten wird auf standardisierte Schnittstellen, Formate, Protokolle zurückgegriffen.

Vaquero, Rodero-Merino, Cáceres und Lindner untersuchen in [VRC+09] eine Reihe von Definitionen des Begriffes Cloud Computing und leiten daraus folgende konsolidierte Definition ab:

> *„Clouds sind riesige Verbünde einfach nutz- und zugreifbarer virtualisierter Ressourcen (z.B. Hardware, Entwicklungsplattformen und/oder Dienste). Diese Ressourcen können dynamisch rekonfiguriert werden, um sich an variable Last anzupassen und damit eine optimale Ressourcennutzung zu erreichen. Dieser Ressourcenverbund wird typischerweise nutzungsabhängig abgerechnet, wobei der Infrastrukturanbieter bestimmte Garantien hinsichtlich festgelegter SLAs[1] abgibt." [VRC+09]*

Aus den genannten Quellen lassen sich einige Wesensmerkmale des Cloud Computings zusammenfassen, die in nahezu allen Definitionen in der einen oder anderen Art zu finden sind:

- Verfügbarkeit quasi unendlicher IT-Ressourcen (Speicher, Rechenleistung etc.)
- Bedarfsgerechte Bereitstellung - ohne Vorlauf und ohne Vorab-Investitionen
- Skalierbarkeit und Elastizität zur bedarfsgerechten Anpassung der Ressourcen
- Möglichkeit zur rein verbrauchsabhängigen Bezahlung
- Zugriff über wohldefinierte Schnittstellen, die auf Internet-Standards basieren
- Garantie einer vom Anbieter bestimmten Dienstqualität auf Basis vereinbarter SLAs

Für einen Entwurf einer nutzenmaximierenden Anwendungsarchitektur müssen all diese Punkte berücksichtigt werden, aus denen sich unter anderem folgende architekturrelevante Schlüsse ziehen lassen:

- Die Möglichkeit, Softwarekomponenten bedarfsgerecht mit IT Ressourcen in quasi unbegrenzter Menge unterlegen zu können, senkt das Risiko bei Kapazitätsplanungen.

- Anwendungskomponenten, deren Nutzung starken Schwankungen unterliegt, können kostengünstiger betrieben werden.

- Die Kosten für den Betrieb einer Anwendung können in der Cloud stärker an der Nutzung des Systems ausgerichtet werden. Das Risiko eines Business Case kann dadurch gesenkt werden. Auch kleine Installationen können sich rechnen.

[1] SLA = Service Level Agreement

Aus den genannten Wesensmerkmalen der Cloud lassen sich folgende typische Einsatzszenarien ableiten:

- *Temporärer Ressourcenbedarf* - Immer dann, wenn Rechen- und Speicherleistung nur zu bestimmten, begrenzten Zeiträumen benötigt wird, kann Cloud Computing diese Leistung bedarfsgenau bereitstellen. Damit fallen auch nur dann Kosten an, wenn die eingesetzte Software tatsächlich genutzt wird.

- *Schnelles beständiges Wachstum* - Cloud Computing abstrahiert von der Dimensionierung der bereitgestellten IT-Ressourcen. Software kann zunächst für einen sehr kleinen Ressourcenpool (für den entsprechend nur geringe Kosten anfallen) entwickelt werden, der im Erfolgsfall der Anwendung dann auf Knopfdruck bedarfsgerecht erweitert werden kann. Die Software muss dabei nicht angepasst werden.

- *Schwankungen im Lastverlauf* - Bei vielen Anwendungen besteht die Herausforderung bei der IT-Kapazitätsplanung darin, dass der Lastverlauf von regelmäßigen oder unregelmäßigen Spitzen geprägt ist. Ein Beispiel hierfür sind e-Commerce-Seiten, die vor Weihnachten hohe Last und unterjährig geringe Last beobachten lassen. Mit Cloud Computing ist es möglich, die eingesetzten Kapazitäten schnell und flexibel an die jeweils anliegende Last anzupassen.

- *Weltweiter Zugriff* - Die großen Cloud Computing Anbieter halten Rechen- und Speicherleistung in weltweit verteilten Rechenzentren vor. Auf diesen Plattformen ist es möglich, Cloud Services in regionaler Nähe zum jeweiligen Anwender anzubieten, was sich günstig auf die Netzwerklatenz und damit die Zugriffsgeschwindigkeit auswirkt.

Diese Einsatzszenarien geben Software-Architekten bereits erste Hinweise darauf, in welchen Situationen Cloud Computing als Betriebsumgebung für Anwendungskomponenten betrachtet werden sollte. In den weiteren Ausführungen soll dies näher untersucht und Hinweise für die Bewertung des Einsatzes alternativer Umgebungen gegeben werden.

Bei all den Vorteilen, die Cloud Computing im Softwarebetrieb bietet, zeigt die Erfahrung auch, dass in konkreten Projekten von Prozessbeteiligten immer wieder ähnliche Argumente vorgebracht werden, die aus ihrer Sicht gegen eine Nutzung von Cloud Technologien sprechen. Dazu gehören unter anderem

- *Sicherheit* – es wird argumentiert, Cloud Anbieter könnten nicht das gleiche Sicherheitsniveau wie im eigenen Rechenzentrum sicherstellen.

- *Compliance* – juristische Vorgaben würden den Einsatz von Cloud Technologien (insbesondere bei Ausführungs- und Speicherressourcen im Ausland) ausschließen.

- *Kontrollverlust* – die fehlende Möglichkeit zum unbeschränkten physischen Zugriff auf die Cloud Ressourcen wäre generell nicht zu akzeptieren.

Teils sind diese Argumente in konkreten Umständen begründet und nachvollziehbar (z.B. gesetzliche Rahmenbedingungen, die Vorgaben für Architekturaspekte wie den Speicherort für Daten vorgeben), teils kann aber auch angenommen werden, dass diese Argumente vorgebracht werden, weil eine grundsätzliche, subjektive Ablehnung von Cloud Computing besteht, diese aber hinter vermeintlich objektiven Begründungen versteckt werden soll. Nicht immer ist es angebracht, diese Argumente infrage zu stellen. Es ist jedoch in jedem Fall sinnvoll, zumindest die Konsequenzen offenzulegen, die sich aus derartigen Argumenten und einer Entscheidung gegen den Einsatz von Cloud Technologien ergeben (z.B. Preis des Verzichts auf eine kostengünstigere Betriebsalternative für Softwarekomponenten).

Die Erfahrungen des Autors dieser Arbeit zeigen, dass in Diskussionen zur Architektur von Softwaresystemen deren grundsätzlicher Aufbau unter den Beteiligten oft unstrittig ist. Häufig herrscht Einigkeit darin, dass die Software beispielsweise über eine klassische 3-Schichten-Architektur implementiert werden soll, dass ein relationales Datenbanksystem benötigt wird usw. Lediglich zur Frage, ob bzw. welche Teile der Anwendung ggf. in der Cloud ausgeführt werden sollen, herrscht Uneinigkeit. Klems u.a. merken in [KNT08] an, "*However, there is no systematic, dedicated approach to measure the benefit from Cloud Computing that could serve as a guide for decision makers to tell when outsourcing IT resources into the Cloud makes sense*" [KNT08, S.11]. Motiviert durch diese Erkenntnis entwickeln Klems u.a. ein erstes Framework zur Entscheidungshilfe. Es existiert also noch kein speziell auf die Anforderungen von Cloud Computing zugeschnittener Prozess mit entsprechender Beschreibungsmethode, der begleitend zum Entwurf von Softwarearchitekturen bei der Entscheidung pro oder contra Cloud helfen könnte. Diese Lücke soll im Rahmen dieser Arbeit geschlossen werden.

1.3 IT-as-a-Service

Der Begriff Cloud Computing selbst gibt noch keine Auskunft darüber, welche Art von „Computing", d.h. IT-Leistung, über Cloud-Ansätze bereitgestellt wird. Die oben genannten Eigenschaften des Cloud Computings geben aber bereits Hinweise auf die Art und Weise auf die die betreffenden Leistungen bereitgestellt werden: als Service. Dies ist in Abgrenzung zu „IT-as-a-Product" zu sehen, bei dem für den reinen Besitz von IT-Ressourcen (z.B. Software) Kosten anfallen, d.h. unabhängig davon, ob die IT-Ressourcen auch tatsächlich genutzt werden.

1.3.1 Serviceorientierung der IT

Der BITKOM sieht diesen Trend in einem größeren Kontext: „Seit der Jahrtausendwende befindet sich die IT-Industrie in der Transformation in eine Service-orientierte IT-Welt." [BIT09,

S.21]. Motiviert durch den Wunsch nach hoher Agilität und Flexibilität, erhalten Anwendungssysteme zunehmend eine serviceorientierte Architektur (SOA), bei der bestimmte Funktionalitäten in lose gekoppelte Services gekapselt werden, die flexibel miteinander kombiniert werden können. Die für die Koppelung von Services verwendete Infrastruktur erlaubt in diesem Zusammenhang auch eine gewisse Mobilität von Services, d.h. der Ort der Service-Ausführung kann sich zur Laufzeit ändern. Services können dabei auch an externe Ausführungsumgebungen (z.B. Service Provider) ausgelagert werden.

Für den Endanwender eines derartigen Systems sollten Verteilung und Kombination von Services verborgen bleiben. Hierzu müssen von Seiten der Softwarearchitektur gegebenenfalls entsprechende Maßnahmen ergriffen werden. So kann es erforderlich sein, bei Auslagerung von Services an Dienstleister Komponenten einzuführen, die z.B. bei Ausfall der Netzwerkverbindung zum Dienstleister auch einen „Offline"-Modus der Anwendung und damit ein Weiterarbeiten des Anwenders ermöglichen.

1.3.2 Ebenen von IT-as-a-Service

Die vom BITKOM beobachtete Transformation in eine Service-orientierte IT-Welt beschränkt sich allerdings nicht auf den Bereich von Softwarearchitekturen, sondern erstreckt sich auf alle Ebenen des IT-Stacks: von der Hardware- und Software-Infrastruktur über Entwicklungs- und Ausführungsplattform bis hin zur Anwendungssoftware. Das Cloud Computing Paradigma lässt sich in diesem Zusammenhang als Fundament für die Dienste aller Ebenen sehen. In der Literatur haben sich für die einzelnen Ebenen entsprechende Begriffe durchgesetzt [BIT09, S.22ff], [LKN+09].

- *Infrastructure-as-a-Service (IaaS)*
  Auf der untersten Ebene finden sich IT-Leistungen der Infrastruktur. Hierbei handelt es sich häufig um standardisierte (zum Teil virtualisiert bereitgestellte) Rechen-, Speicher- und Netzwerkkomponenten. System-Administratoren können hierüber flexibel eine lokale Infrastruktur um Bestandteile ergänzen, um z.B. virtuelle Maschinen in der Cloud anstelle einer lokal Umgebung zu betreiben. Die bereitgestellte Infrastruktur kann sehr flexibel eingesetzt werden, allerdings bleiben Implementierungs-, Konfigurierungs- und Betriebsdetails der höheren Schichten in der Verantwortung des Nutzers. Beispiele für IaaS-Angebote sind Microsoft Azure Virtual Machines [Mic15], Amazon EC2 [Aws15], AppNexus [App15], HP Helion CloudSystem [HP15], VMware vCloud [MV15], Oracle IaaS [Ora15].

- *Platform-as-a-Service (PaaS)*
  Eine Ebene darüber sind IT-Leistungen angesiedelt, die Architekten, Entwicklern und Systemintegratoren die Möglichkeit geben, Software und Softwarekomponenten für die Ausführung in der Cloud zu entwickeln, Cloud-Speicherdienste in Anwendungen

zu nutzen oder die Kommunikation über die Cloud abzuwickeln. Details der zugrundeliegenden Infrastruktur bleiben dabei verborgen bzw. sind für den Nutzer irrelevant. Dies kann den Entwurf und Aufbau von Softwarearchitekturen deutlich vereinfachen, da Fragen des Betriebs in der Verantwortung des Service-Anbieters verbleiben. Beispiele für PaaS-Angebote sind Force.com [Sal15], Google App Engine [Goo14] und entsprechende Dienste in Amazon Web Services [Aws15] oder Microsoft Azure [Mic15].

- *Software-as-a-Service (SaaS)*
  Auf der obersten Ebene sind Anwendungen und Anwendungsdienste angesiedelt, die auf Basis von Plattform- und Infrastrukturdiensten als Cloud Services angeboten werden. Diese Dienste richten sich primär an Anwender, die flexibel Software nutzen wollen. Beispiele für SaaS-Angebote sind Google Apps for Business [Goo15], Microsoft Office 365 [Mic15-1], CRM Online [Mic15-2], Salesforce.com [Sal15-1], WebEx [Wex15], etc.

In Abschnitt 2.3 findet sich eine detailliertere Übersicht über konkrete, für Software-Architekten relevante Angebote auf den einzelnen Ebenen.

1.3.3 Sourcing Optionen von Cloud Computing Angeboten

Während bezüglich der Schichtung von Cloud-Angeboten in der Literatur eine gewisse Einigkeit herrscht, hat sich eine Reihe von weiteren Abgrenzungskriterien für Cloud-Angebote am Markt etabliert, die wiederum eine Vielzahl weiterer Bezeichnungen nach sich zieht. An dieser Stelle seien die vom BITKOM vorgenommenen Bezeichnungen [BIT09] aufgeführt, die in dieser Form auch in den weiteren Ausführungen so verwendet werden:

- *Private Cloud* – darunter ist eine unternehmenseigene und vom Unternehmen selbst betriebene Cloud-Umgebung zu verstehen. Über Virtualisierungs- und Automatisierungsmaßnahmen können IT-Ressourcen Anwendern schnell und flexibel bereitgestellt werden.

- *Managed Private Cloud* – im Gegensatz zur Private Cloud im engeren Sinn erfolgt hier der Betrieb der Umgebung, die weiterhin beim Unternehmen verbleibt, durch einen externen Dienstleister.

- *Outsourced Private Cloud* – bei diese Form der Private Cloud wird nicht nur der Betrieb sondern auch die Bereitstellung der IT-Ressourcen vom Dienstleister übernommen. Dieser hält für einen bestimmten Kunden eine dedizierte Umgebung vor.

- *Public Cloud* – auch hier werden Bereitstellung und Betrieb der IT-Ressourcen von einem externen Dienstleister übernommen. Dieser unterhält große Rechenzentren, in

denen er stark standardisierte Ressourcen in einer industrialisierten Form vielen Nutzern bereitstellt. Durch die große Dimensionierung der Umgebungen kann er hohe Skaleneffekte realisieren, die es ihm erlaubt, die IT-Ressourcen sehr kostengünstig anzubieten.

Abb. 1-1 gibt einen Überblick über die verschiedenen Ebenen und Bereitstellungsformen von Cloud Services.

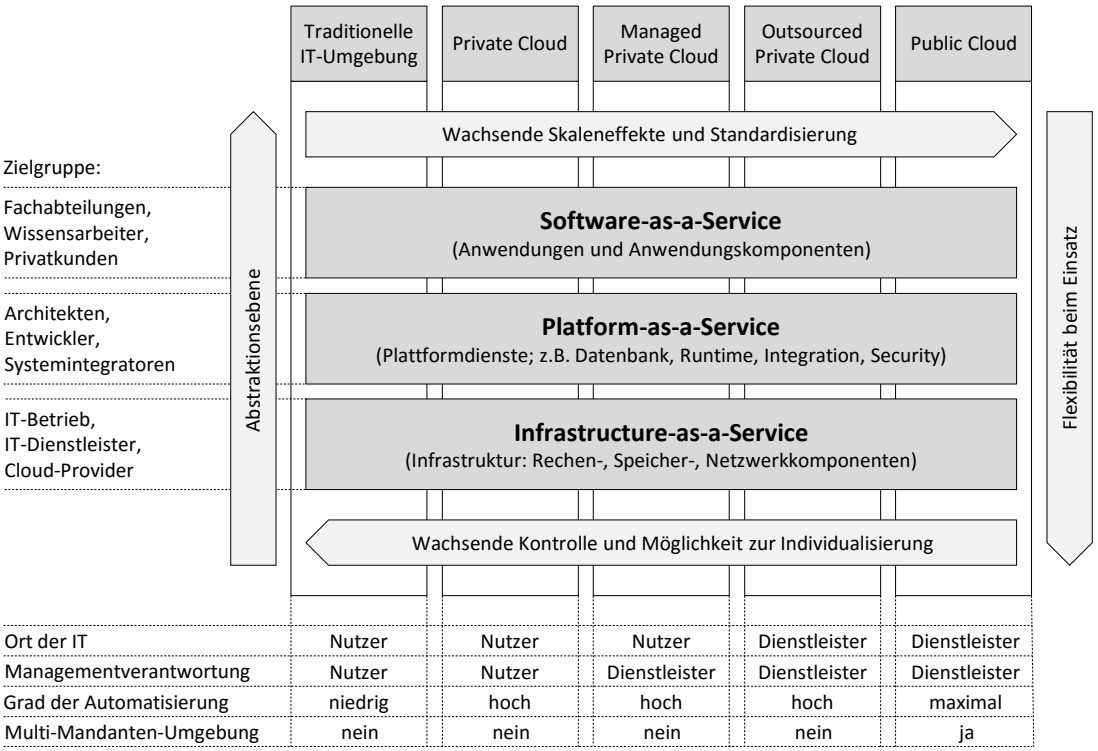

Abb. 1-1: Ebenen und Sourcing Optionen für Cloud Services

Cloud Computing ist also Basis für eine Vielzahl an IT-bezogenen Funktionalitäten. Da in dieser Arbeit primär die Auswirkungen auf das Handlungsspektrum von Software-Architekten untersucht werden sollen, liegt der Schwerpunkt der weiteren Betrachtungen auf der Platform-as-a-Service-Ebene. Die Angebote dieser Ebene richten sich vorwiegend an Entwickler und Software-Architekten, die die betreffenden Dienste beim Aufbau von Software-Systemen einsetzen.

1.4 Architekturbasierter Softwareentwurf

Philippe Kruchten weist in [Kru95] auf die Wichtigkeit hin, Stakeholder über verschiedene Sichten beim Entwurf einer Softwarearchitektur einzubeziehen. Der Entwurf einer Softwarearchitektur sollte auf der einen Seite sehr sorgfältig erfolgen, da die dort getroffenen Entscheidungen in späteren Projektphasen nur mit größeren Aufwänden revidiert werden können. Benestad u.a. haben in [BAA10] eine Vielzahl von Aufwandstreibern im Zusammenhang mit Änderungen an Software-Systemen identifiziert und beschrieben. Auf der anderen Seite, soll die methodische Ausgestaltung des Architekturentwurfs nach [ABK10] agile Softwareentwicklung ermöglichen, d.h. effizient ablaufen. Keuler u.a. bestätigen in [KKN11] die Bedeutung eines architekturbasierten Softwareentwurfs: *„it connects business aspects with technology and balances the interests of all stakeholders of a system and their communication."* [KKN11, S.7].

Die Erstellung eines Softwareentwurfs sollte nach einem geregelten Prozess durchgeführt werden. In [BCK03] wird für die Erstellung einer Softwarearchitektur eine Reihe von Schritten identifiziert. Cloud Computing kann in allen Schritten den Handlungsspielraum der betreffenden Akteure erweitern:

- *Aufstellung eines Business Case für das zu entwickelnde System* – in diesem Schritt werden bereits einige Punkte definiert, die Hinweise darauf geben, nach welchen Kriterien und in welchen Anwendungsszenarien der Nutzen des fertiggestellten Anwendungssystems bewertet wird. Bei Verwendung von Cloud Computing können Anforderungen an den Produktivsetzungstermin, anfallende Kosten etc. deutlich anspruchsvoller ausfallen (z.B. schnellere Bereitstellung, geringere Kosten, …) als dies bei einem klassischen Betriebsmodell möglich wäre.

- *Formulierung und Analyse der Anforderungen* – über verschiedene mögliche Vorgehensweisen sollen anvisierte fachliche und qualitative Eigenschaften des Systems herausgearbeitet werden. Mögliche Techniken sind in diesem Zusammenhang Use Case Analysen und Prototypen. Die konkreten Anforderungen bei den einzelnen Attributen können bereits Hinweise darauf geben, ob Cloud Computing eingesetzt werden kann bzw. eingesetzt werden sollte. Extreme Sicherheitsanforderungen können die Verwendung ausschließen, hohe Anforderungen an die Skalierbarkeit können sie andererseits nahelegen.

- *Erstellung oder Auswahl der Softwarearchitektur* – die benötigten Anwendungskomponenten müssen identifiziert, deren Verteilung und Interaktion modelliert, Installation und Betrieb geplant werden. Der Betrieb der Anwendung bzw. einzelner Komponenten in der Cloud muss berücksichtigt und entsprechend formuliert werden.

- *Dokumentation und Kommunikation der Softwarearchitektur* – in der Beschreibung der Architektur muss die Verteilung der einzelnen Komponenten berücksichtigt und etwaige Abhängigkeiten untereinander und in Bezug auf die jeweilige Ausführungsumgebung beschrieben werden. Je nach verwendeter Beschreibungstechnik können bereits Lücken der Architektur identifiziert werden bzw. Tests auf Vollständigkeit und Konsistenz erfolgen.

- *Analyse und Evaluation der Softwarearchitektur* – die beschriebene Architektur muss in Bezug auf die identifizierten Qualitätskriterien überprüft werden. Bestehende Zielkonflikte müssen identifiziert und aufgelöst werden. Dabei müssen Entscheidungen bezüglich einzelner Umsetzungsalternativen getroffen werden. In einzelnen Fällen kann dies auch eine Entscheidung pro oder contra Cloud Computing bedeuten.

- *Implementierung des Systems auf Basis der Softwarearchitektur* – in diesem Umsetzungsschritt muss die erhaltene Architekturbeschreibung in eine konkrete Softwareimplementierung überführt werden. Dabei muss die Übereinstimmung mit den Architekturentscheidungen sichergestellt werden.

- *Sicherstellung, dass die Implementierung konform zur Softwarearchitektur bleibt* – auch nach Produktivsetzung muss sichergestellt werden, dass die Software – z.B. im Rahmen von Wartungsarbeiten - ihre Konformität zur Architektur behält.

Der Schwerpunkt dieser Arbeit liegt auf den Schritten, die den Entwurf und die Bewertung der Softwarearchitektur zum Gegenstand haben. Die letzten beiden Schritte, die im Rahmen der Implementierung durchgeführt werden, werden hier nicht weiter betrachtet.

Software-Architekten kommt in dieser Methode eine zentrale Rolle zu. Wie auch Starke in [Sta02] hervorhebt, *„bilden [Software-Architekten] die Schnittstelle zwischen Analyse, Entwurf, Implementierung, Management und Betrieb von Software. […] Effektivität, Agilität und Pragmatismus prägen die Grundhaltung erfolgreicher Software-Architekten“*. Die Erfahrungen eines Architekten stellen einen unschätzbaren Mehrwert bei Entwurf und Bewertung eines Architekturentwurfs dar. Sie helfen, den Entwurfs- und Bewertungsprozess einfach zu halten, indem bestimmte Schritte aufgrund der Expertise des Architekten abgekürzt werden können anstelle aufwändige, komplexe Berechnungs- und Prognosemodelle zu erzeugen.

1.5 Irrationales Verhalten der Prozessbeteiligten

In Theorie klingt der Prozess zur Herleitung einer für alle Beteiligten besten Architekturentwurfs einfach: berechne bzw. schätze den Wert, den einzelne Architekturentscheidungen ha-

ben, und dividiere diesen durch den jeweils zugehörigen Aufwand. Wähle dann die Entwurfsalternative, für die die Summe der Quotienten aus Wert und Aufwand – also der Return-on-Investment – maximal wird.

In der Praxis setzt diese Vorgehensweise voraus, dass Wert und Aufwand objektiv bestimmt bzw. prognostiziert werden können und sich die Prozessbeteiligten rational verhalten. Tatsächlich ist dies aus mehreren Gründen nicht notwendigerweise gegeben:

- Die Wichtigkeit einzelner Anforderungen an das zu entwickelnde System kann von den Stakeholdern, d.h. vom System betroffenen Interessensgruppen, sehr unterschiedlich bewertet werden. Häufig misst eine Person, einzelnen Anforderungen nur deshalb eine höhere Wichtigkeit zu, weil sie ggf. selbst von ihnen betroffen ist. Subjektiv betrachtet ist dagegen zunächst auch nichts einzuwenden. Im Sinne einer Maximierung des Gesamtnutzens muss jedoch ein Ausgleich zwischen den Interessen und Anforderungen der einzelnen Beteiligten angestrebt werden.

- Sowohl der Wert als auch der Nutzen einer Architekturalternative kann von verschiedenen Prozessbeteiligten sehr unterschiedlich angesehen werden. Dabei kann bei den betreffenden Personen nicht immer ein ausschließlich rationales Verhalten unterstellt werden. Dieser Umstand wird in der Verhaltensökonomie unter anderem von Lin in [Lin11] mit psychologischen, sozialen, kognitiven und emotionalen Einflüssen begründet, denen die Personen ausgesetzt sind.

- Ob der Wert bzw. der Aufwand in einer späteren Implementierung tatsächlich erzielt werden kann bzw. anfällt, ist mit einem Risiko behaftet. Auch dieses kann – aus ähnlichen Gründen – von den Prozessbeteiligten sehr unterschiedlich bewertet werden.

Dass diese Irrationalität der Beteiligten in den einzelnen Schritten zur Herleitung eines Architekturentwurfs nicht nur „graue Theorie“ ist, lässt sich in der Praxis insbesondere im Rahmen von Diskussionen zum Einsatz von Cloud-Technologien sehr häufig beobachten. Ein paar Beispiele:

- Vorurteile beeinflussen Entscheidungen: Neues, d.h. eine von bisherigen Vorgehensweisen abweichende Lösungsalternative, wird per se als riskanter und damit als weniger geeignet angesehen. Das Sicherheitsniveau der Cloud-Systeme wird beispielsweise in Frage gestellt. Dabei wird ignoriert, dass dieses in der Regel von großen Cloud-Anbietern weit höher umgesetzt wird als dies im eigenen Rechenzentrum der Fall ist.

- Die Beteiligten zögern, die Ersten zu sein: Ein speziell in Deutschland verbreitetes Phänomen, dass neue Lösungsalternativen erst „von anderen“ evaluiert und für praktikabel befunden werden sollen, bevor man sich selbst an deren Einsatz wagt.

- Insbesondere bei einer vorgefestigten negativen Meinung werden schlechte Nachrichten über bestimmte Alternativen (z.B. Cloud Computing) über-, positive hingegen unterbewertet.

- David Tang beschreibt in [Tan13] „Besitzstandseffekte“, die zu irrationaler Bewertung von Alternativen führen. Dies lässt sich ebenfalls in konkreten Diskussionen um den Einsatz von Cloud Technologien beobachten: Alleine die Tatsache, dass IT-Ressourcen in die Cloud abgegeben werden sollen und ein gewohntes Verhalten damit aufgegeben werden soll, wird als per se schwerwiegender eingestuft als mögliche Gewinne durch die Cloud (Kostensenkungen, Flexibilität etc.).

Es ist schwer, genau zu beziffern, wie groß der durch dieses Verhalten verursachte Schaden durch Wahl einer sub-optimalen Lösungsalternative ist. Es ist anzunehmen, dass Cloud-Optionen in vielen Projekten, in denen sie zum Einsatz kommen könnten, bereits vor einem Architekturentwurf verworfen werden und sich so einer eingehenderen Untersuchung entziehen. Hinweise auf die Dimension des Problems geben allerdings Untersuchungen verschiedener Marktbeobachter und Cloud-Anbieter:

- Die Analysten von comScore haben in ihrer Studie [Pra13] Vertreter von mittelständischen Unternehmen befragt. 60% der Befragten nannten Bedenken über die Datensicherheit als Einführungsbarriere. 45% sorgten sich, dass die Cloud zu einem Verlust der Datenkontrolle führen würde. In der gleichen Studie gaben aber auch 94% der Befragten an, nach Einführung von Cloud-Technologien Sicherheitsvorteile erzielt zu haben, 62% nannten einen Anstieg im Datenschutz als Ergebnis der Cloud Einführung.

- Amazon zeigt in Kundenreferenzen jeweils durch die Cloud realisierte Vorteile auf: ein Umsetzungsprojekt bei der Hess Corporation (siehe [Aws14-1]) hätte ohne Cloud doppelt so viel Zeit gekostet; in einem Projekt mit Airbnb (siehe [Aws14-2]) konnten durch den Einsatz von Cloud-Technologien im Systembetrieb Personalkosten im Äquivalent eines Mitarbeiters eingespart werden.

- Microsoft zitiert ebenfalls aus erfolgreichen Cloud-Projekten: Harper Collins konnte ein Projekt mit Cloud Technologien innerhalb von zwei Wochen anstelle in 6 bis 12 Monaten ohne Cloud umsetzen (siehe [Mic13-1]); Lufthansa Systems berichtet im Zusammenhang mit der Nutzung von Cloud Technologien von Betriebsverbesserungen hinsichtlich Bereitstellungszeiten bei gleichzeitig gesunkenen Investitions- und Betriebskosten (siehe [Mic13-2]).

Wenngleich auch derartige Studien wiederum durch bestimmte Interessen beeinflusst sein können, lässt sich doch festhalten, dass sich durch Cloud Computing für Architekturentscheidungen neue Optionen auftun, die Vorteile hinsichtlich Kosten, Umsetzungsgeschwindigkeit, Flexibilität etc. versprechen. Diese leichtfertig auszuschließen, wäre bestenfalls fahrlässig.

1.6 Evaluation von Software-Architekturen

Es sollte Ziel aller Prozessbeteiligten sein, so früh wie möglich beurteilen zu können, inwieweit Anforderungen durch das System erfüllt werden. Je früher Probleme identifiziert werden können, desto besser. Nach [CKK02] ist der beste Zeitpunkt, eine Software-Architektur zu evaluieren, nach Abschluss des Architekturentwurfs und bevor eine Implementierung begonnen wird. Grundlage der Evaluierung sind die Anforderungen aller Beteiligten.

Der in dieser Arbeit hergeleitete Entwurfs- und Planungsansatz für Softwarearchitekturen kann Irrationalitäten den Anforderungen der Beteiligten natürlich nicht beseitigen. Auch ist es nicht möglich, einen vollständigen Kriterienkatalog aufzustellen, in dem alle Verhaltensparameter erfasst und mit dem Entscheidungen durch Überprüfung dieser Kriterien objektiv getroffen und damit Irrationalitäten vermieden werden könnten. Irrationalitäten sollen durchaus zugelassen und Entscheidungen entsprechend beeinflusst werden. Durch regelmäßige Plausibilitätsprüfungen im Rahmen des Architekturentwurfs sollen die Entscheidungen aber hinterfragt und durch entsprechende Rückkopplungsschleifen, in denen Entscheidungsparameter angepasst werden können, revidiert werden können.

Ziel ist es also, Mittel zu schaffen, Motivationen und Konsequenzen von Entwurfsentscheidungen offen zu legen, einer Diskussion zugänglich zu machen und damit Transparenz zu gewinnen. Folgende Anforderungen gelten demnach für den in dieser Arbeit entwickelten Prozess:

- Die Wichtigkeit einer Anforderung an das Softwaresystem soll für die einzelnen Beteiligten jeweils getrennt erfasst und dokumentiert werden.
- Wert und Nutzen von Architekturalternativen sollen von allen Beteiligten geschätzt und mit den daraus abgeleiteten Kennzahlen entsprechend dokumentiert werden.
- Auch die Risikoeinschätzung soll jeweils einzeln dokumentiert werden.

Es kann nicht ausgeschlossen werden, dass der neue Prozess als Ergebnis den gleichen Architekturentwurf ergibt, wie dies mit alternativen Vorgehensweisen der Fall gewesen wäre. Vorteil hier ist allerdings, dass die Annahmen, Bewertungen und Entscheidungen – insbesondere für nachfolgende Architekten und Entwickler – dokumentiert und nachvollziehbar sind.

1.7 Zielsetzung dieser Arbeit

Bestehende Vorgehensmodelle zur Erstellung, Beschreibung und entsprechender Bewertung von Architekturen verteilter Unternehmensanwendungen berücksichtigen die neuen Optionen des Cloud Computings noch zu wenig bis überhaupt nicht. Ziel dieser Arbeit ist es, diese

Lücke zu schließen. Hierzu soll auf Basis bestehender Entwurfs- und Bewertungsmethoden eine formale Vorgehensweise zur Bestimmung einer nutzenmaximierenden Systemarchitektur bei gegebenen Anforderungen hergeleitet werden. Dies schließt eine methodische und systematische Unterstützung des Entwurfs einer Softwarearchitektur, eine Methodik zur Bewertung des „Nutzens" der Architektur sowie die Unterstützung der Konsequenzanalyse bei Veränderung der Architektur (z.B. durch Änderung der Verteilung der einzelnen Komponenten) ein. Im Einzelnen werden folgende Punkte behandelt:

- Erweiterung bestehender Entwurfs- und Bewertungsmethoden, um Alternativen in der Architektur und der Verteilung von Softwarekomponenten angemessen zu berücksichtigen
- Erweiterung einer bestehenden Methode zur Beschreibung von Software-Architekturen, um durchgängig im Rahmen des Entwurfs und der Bewertung genutzt werden zu können
- Ableitung und Verifikation eines durchgängigen Vorgehensmodells zur Erstellung nutzenmaximierender Architekturen verteilter, Cloud-basierter Unternehmensanwendungen

Um die Aufgabenstellung beherrschbar zu machen, wird bei den Ausführungen spezielles Augenmerk auf die Betriebsalternative Cloud Computing im Kontext verteilter Unternehmensanwendungen gelegt.

1.8 Inhalt und Aufbau

Die einzelnen Kapitel der Arbeit lassen sich wie folgt inhaltlich skizzieren:

- *Kapitel 1: Einführung* – Im ersten Kapitel werden die für die Arbeit zentralen Begriffe Cloud Computing und architekturbasierter Softwareentwurf eingeführt und abgeleitet, in welcher Beziehung diese Begriffe zur Zielsetzung der Arbeit stehen.
- *Kapitel 2: Grundlagen* – Mit Hilfe von Definitionen bestehender Technologien und Begriffe wird in Kapitel 2 der Untersuchungsgegenstand der Arbeit beschrieben. Zum einen wird festgelegt, welche Problemstellung betrachtet wird, zum anderen erfolgt auch eine Abgrenzung zu explizit nicht betrachteten Bereichen.
- *Kapitel 3: Anforderungen an eine integrierte Entwurfs- und Bewertungsmethodik* – In Kapitel 3 werden die Anforderungen an Prozesse, Entwurfsartefakte und Werkzeuge beschrieben, die für einen möglichst allgemeingültigen Ansatz zur Erstellung einer nut-

zenmaximierenden Software-Architektur Cloud-basierter Unternehmensanwendungen benötigt werden. Dabei steht die Frage nach einer sinnvollen Nutzenbewertung und Nutzenoptimierung im Mittelpunkt.

- *Kapitel 4: Stand der Forschung* – In diesem Kapitel wird beleuchtet, wo bereits Lösungen und Bausteine für den in Kapitel 3 beschriebenen Entwurfsansatz bestehen, und in welchen Bereichen die vorliegende Arbeit Lücken schließt.

- *Kapitel 5: Erweiterung einzelner bestehender Beschreibungs- und Vorgehensmethoden* – Hier werden bestehende Vorgehensmodelle einzeln betrachtet und mit Beschreibungsmodellen für Softwarearchitekturen in Einklang gebracht, d.h. entsprechend erweitert.

- *Kapitel 6: Zusammenführung der erweiterten Methoden zu einem Gesamtprozess* – Die Vorgehensmodelle, die einzeln jeweils nur Teilaspekte des Entwurfs- und Bewertungsprozesses berücksichtigen, werden in diesem Kapitel zu einem integrierten Modellierungs- und Bewertungsprozess verknüpft.

- *Kapitel 7: Anwendung des Leitfadens auf Cloud-Anwendungen* – In diesem Kapitel wird der abgeleitete, integrierte Prozess auf Cloud-basierte Unternehmensanwendungen angewandt, d.h. anhand stellvertretender Fallbeispiele wird der Prozess auf verschiedene Qualitätskriterien (Vollständigkeit, Anwendbarkeit, Korrektheit etc.) geprüft.

- *Kapitel 8: Zusammenfassung und Ausblick* – Abschließend wird in Kapitel 8 ein Überblick über noch offene Forschungsbereiche und mögliche Erweiterungen dieser Arbeit gegeben.

2 Grundlagen

In diesem Kapitel werden zentrale, für die Arbeit relevante Grundlagen beschrieben. Zunächst werden die wichtigsten Begriffe definiert und gegenüber verwandten Technologie- und Forschungsgebieten abgegrenzt. Damit wird der Aktionsspielraum der Arbeit abgesteckt. Speziell für den Bereich Cloud Computing werden typische Einsatzszenarien aufgezeigt und in einem Marktüberblick bereits verfügbare Angebote verschiedener Hersteller kurz beschrieben.

2.1 Grundlegende Begriffsdefinitionen

In dieser Arbeit wird eine Reihe von grundlegenden Begriffen verwendet, die im Folgenden definiert werden. Zu den einzelnen Begriffen werden zudem die für die weiteren Ausführungen wichtigen Wesensmerkmale und der Bezug zum Themenbereich Cloud Computing hervorgehoben.

2.1.1 Softwarearchitektur

Zum Begriff der Softwarearchitektur gibt es eine Vielzahl an Definitionen. Im Kontext dieser Arbeit soll die Definition des Institute of Electrical and Electronics Engineers, Inc. (IEEE) aus dessen Standard IEEE 1471 zurückgegriffen werden.

> ***Softwarearchitektur*** *ist „die grundlegende Organisation eines Systems, dargestellt durch dessen Komponenten, deren Beziehungen zueinander und zur Umgebung sowie den Prinzipien, die den Entwurf und die Evolution des Systems bestimmen." [Iee07].*

Diese Definition gibt bereits Hinweise auf die zentralen Aspekte, die auch Strukturierungsgrundlage für diese Arbeit sind. Für den Aufbau einer Softwarearchitektur müssen unter anderem die folgenden Fragen beantwortet werden:

- Welche Anforderungen (fachlich, technisch, organisatorisch, …) motivieren eine bestimmte Ausprägung einer Softwarearchitektur?
- Welche Komponenten werden zur Abdeckung der Anforderungen benötigt?
- Wie sind die Komponenten verteilt und wie interagieren sie miteinander?

In dieser Arbeit werden die Fragen unter besonderer Berücksichtigung der Betriebsalternative Cloud Computing beleuchtet, d.h. es soll untersucht werden, welchen Einfluss Cloud Com-

puting auf Anforderungen, Auswahl und Interaktion der Komponenten eines Software-Systems haben kann, und welche Konsequenzen sich daraus auf verschiedene Entwurfsentscheidungen ergeben.

2.1.2 Services und Serviceorientierte Architekturen

Service-Orientierung ist ein zentrales Paradigma, das dem effizienten Einsatz des Cloud Computings zugrunde liegt. Cloud Computing bietet die technische Grundlage, IT Funktionen flexibel, für quasi beliebige Lastszenarien als Service bereitzustellen. Die Motivation für die Service-Orientierung wird in [ABB+09] wie folgt formuliert:

> *„Wir haben Serviceorientierung angewendet, um Organisationen zu helfen, kontinuierlich nachhaltigen Geschäftswert zu liefern mit höherer Agilität und Kosteneffizienz und im Einklang mit den sich ändernden fachlichen Bedürfnissen.“[ABB+09]*

Service-Orientierung hat demnach eine Geschäftswert- und eine technologische Komponente. Die technische Umsetzung von Services erfolgt mit dem Ziel dauerhaft, effizient und effektiv den Geschäftsbetrieb des Anwenders zu unterstützen. [CDG+10], [DEF08] und [Erl06] identifizieren in diesem Zusammenhang folgende Eigenschaften von Services:

- *Lose Kopplung* – Abhängigkeiten zwischen Services sind minimiert. Dies kann auf verschiedenen Ebenen (asynchrone Aufrufbeziehungen, dynamische Bindung etc.) erfolgen.
- *Abstraktion* – Die Funktionalität eines Service wird ausschließlich durch die Service-Schnittstelle beschrieben. Weitere Details zur Implementierung und inneren Funktionsweise sind nach außen nicht bekannt.
- *Service-Komposition* – Services können ihrerseits aus Services aufgebaut sein, unabhängig von Größe und Komplexität dieser Services-Kompositionen.
- *Service-Contract* – Services erfüllen innerhalb einer Service-Domäne die gleichen formalen Standards hinsichtlich der Service-Schnittstelle. Hierzu gehören einheitliche Design-Standards und Konventionen für die technischen Aufrufschnittstellen, konsistente Datenmodelle (um unnötige Datentransformationen innerhalb der Domäne zu vermeiden) etc.
- *Wiederverwendbarkeit* – Die Implementierung eines Service ist unabhängig von bestimmten Service-Konsumenten.
- *Autonomie* – Ein Dienst ist in sich abgeschlossen. Dies schließt in der Regel die erforderliche Laufzeitumgebung ein. Damit kann der Service eigenständig genutzt werden.

- *Zustandslosigkeit* – Services minimieren ihren Ressourcenkonsum, indem sie das Zustandsmanagement – falls benötigt – delegieren. Dies erhöht die Skalierbarkeit der Services, da eine parallele Ausführung mehrerer Service Instanzen erleichtert wird. Darüber hinaus wird hierdurch der Ansatz der Service-Abstraktion unterstützt, was wiederum die Wiederverwendbarkeit erhöht.

- *Auffindbarkeit* – Services sind mit Metadaten ausgestattet, anhand derer sie gefunden und interpretiert werden können.

All diese Eigenschaften kommen im Zusammenhang mit Cloud Computing zum Tragen, wo eine Service-Orientierung erzwungen wird, wenn die angebotenen IT Leistungen effizient genutzt werden sollen. Beim Aufbau von Softwarearchitekturen, in denen Cloud Services zum Einsatz kommen sollen, muss dies deshalb berücksichtigt werden. Service-orientierte Architekturen (SOA) werden hierbei in [BKN+10] wie folgt definiert:

> *„Unter einer **SOA [(Service-orientierten Architektur)]** versteht man eine Systemarchitektur, die vielfältige, verschiedene und eventuell inkompatible Methoden oder Applikationen als wieder verwendbare und offen zugreifbare Dienste repräsentiert und dadurch eine plattform- und sprachenunabhängige Nutzung und Wiederverwendung ermöglicht." [BKN+10, S.18]*

Beim Cloud Computing werden virtualisierte IT-Ressourcen als Dienste bereitgestellt und stehen über ihre Schnittstellen leicht zur Nutzung im Rahmen einer SOA zur Verfügung. Im Falle einer Public Cloud werden diese Dienste auf Basis standardisierter Web-Protokolle und Schnittstellen angeboten.

2.1.3 Verteilte Anwendungen

Nachdem bei Software, bei der im Betrieb Cloud-Technologien zum Einsatz kommen, inhärent eine Verteilung von Softwarekomponenten auf die bereitgestellten IT-Ressourcen stattfindet, kommt der Klasse der Verteilten Anwendungen im Rahmen der weiteren Untersuchungen große Bedeutung zu. Diese wird in [Sch10] wie folgt definiert:

> *„Ein **Verteiltes System** ist ein System, in dem Hardware- und Softwarekomponenten, die auf Netzwerkrechner verteilt sind, primär über den Austausch von Nachrichten miteinander kommunizieren und ihre Aktionen koordinieren." [Sch10]*

Für den Begriff der **Verteilten Anwendung** werden ebendort folgende drei Aspekte identifiziert:

- Die Anwendung A, deren Funktionalität auf eine Menge von kooperierenden, interagierenden Komponenten $A_1, \ldots, A_n, n \in \mathbb{N}, n > 1$ aufgeteilt ist; von denen jede Komponente einen internen Zustand (Data) und Operationen zur Manipulation des Zustands besitzt.

- Die Komponenten A_i sind autonome Einheiten, die verschiedenen Maschinen F_i zugeordnet werden können.

- Die Komponenten A_i tauschen über das Netzwerk Informationen aus.

Aus Sicht der Anwendungsarchitektur stellen sich demzufolge folgende Fragen

- Welches ist die beste Aufteilung der Anwendungsfunktionalität?
- Wie sollten die Komponenten A_i Maschinen, d.h. Ausführungsumgebungen zugeordnet werden? Welche Implikationen ergeben sich aus der gewählten Zuordnung?
- Wie können Informationen zwischen den Komponenten möglichst effizient ausgetauscht werden?
- Wie lassen sich die Entscheidungsalternativen bewerten, um letztlich eine möglichst „gute" Gesamtanwendung zu erhalten?

In jeder der genannten Fragestellungen ergeben sich durch den Einsatz von Cloud Computing neue Möglichkeiten bei Verteilung und Interaktion von Anwendungskomponenten. Und auch hier stellt sich die Frage nach einer Bewertung der Alternativen. Auch Dunkel u.a. heben in [DEF+08] im Zusammenhang mit dem Entwurf verteilter Anwendungen hervor, *„dass man eine gegebene Architekturform nicht per se als gut oder schlecht bezeichnen kann, sondern nur als gut oder schlecht geeignet für ein bestimmtes gegebenes Anwendungsproblem mit seinen Nebenbedingungen"* [DEF+08]. In den folgenden Abschnitten soll die Frage beantwortet werden, wie man aus dem gegebenen Anwendungsproblem die beste Architekturform und Verteilung der Komponenten ableiten kann.

2.1.4 Unternehmensanwendungen

Grundsätzlich gibt es zahllose Klassen von Anwendungssystemen, die jeweils spezifische Anforderungen haben (Echtzeitsysteme stellen hohe Anforderungen an die Kommunikationsinfrastruktur, Embedded Software unterliegt bestimmten Restriktionen der ausführenden Geräte etc.). Dies erschwert die Herleitung einer einheitlichen Vorgehensweise zur Bestimmung einer Anwendungsarchitektur. In dieser Arbeit soll deshalb der Fokus zunächst auf Unternehmensanwendungen liegen, wenngleich die Anwendbarkeit der hergeleiteten Vorgehensweise nicht ausschließlich auf diesen Anwendungstyp beschränkt bleiben soll.

Martin Fowler skizziert in [Fow03] ein paar Wesensmerkmale von Unternehmensanwendungen, um diese mangels einer präzisen Definition von anderen Anwendungstypen abzugrenzen:

- Dauerhafte Speicherung von Daten (oft länger als die Lebensdauer des ursprünglichen Speichermediums).
- Speicherung großer Datenmengen (> 1GB).
- Gleichzeitiger Zugriff auf die Daten.
- Starke Varianz in der Zahl der Anwender (zwischen 100 und 100 Millionen).
- Unterstützung verschiedener Benutzeroberflächen und Formfaktoren.
- Integration mit anderen Systemen in einer oft heterogenen Systemlandschaft mitunter über Unternehmensgrenzen hinweg.
- Konzeptionelle Dissonanz in Syntax und Semantik gespeicherter Datenobjekte.
- Komplexe Geschäftslogik, deren Anforderungen sich jederzeit ändern können.

2.1.5 Cloud-basierte verteilte Unternehmensanwendungen

Aus der Definition bzw. den Wesensmerkmalen Verteilter Anwendungen und Unternehmensanwendungen kann die Definition des Anwendungstyps hergeleitet werden, der Gegenstand der weiteren Betrachtungen sein soll: Cloud-basierte verteilte Unternehmensanwendungen.

> Eine **Cloud-basierte verteilte Unternehmensanwendung** ist eine Anwendung A, die folgende drei Kriterien erfüllt
>
> - A ist eine verteilte Anwendung.
> - A ist eine Unternehmensanwendung.
> - Mindestens eine der Anwendungskomponenten A_i wird (zeitweise) in der Cloud ausgeführt.

2.2 Abgrenzung zu verwandten Begriffen und Konzepten

Cloud Computing steht in enger Beziehung zu verwandten Begriffen und Konzepten, die mit vielen Ideen und Technologieansätzen wertvolle Voraussetzungen für die Cloud geschaffen haben.

2.2.1 Cluster und Grid Computing

Tabelle 2-1 zeigt eine aus [Fos02], [Pol09], [Mye09] und [ZCB10] abgeleitete Gegenüberstellung von Cluster, Grid und Cloud Computing.

Eigenschaft	Cluster	Grid	Cloud
Ausstattung	Commodity Computer	High-end Computer (Server, Cluster)	Commodity Computer und High-end Server- und Speichersysteme
Größe / Skalierung	100	1.000	1.000 bis 10.000
Kapazität	Stabil und garantiert	Variiert; aber hoch	Bereitstellung auf Bedarf
Failure Management ("self healing")	Beschränkt (oft werden fehlgeschlagene Tasks/Anwendungen nur neu gestartet)	Beschränkt (oft werden fehlgeschlagene Tasks/Anwendungen nur neu gestartet)	Umfangreiche Unterstützung für Failover und Replikation. VMs können leicht umgezogen werden.
Pricing	Beschränkt; kein offener Markt	Dominiert durch öffentliche Bereitstellung	Nutzungsabhängige Abrechnung
Anwendungs-szenarien	Wissenschaft, Unternehmen, Rechenzentren	Anwendungen für wissenschaftliche Berechnungen mit hohen Durchsatzanforderungen	Dynamisch provisionierte Legacy und Web Anwendungen, Content Delivery
Potenzial für 3rd Party oder Value Added Solutions	Beschränkt durch die klar vorgegebene Architektur	Beschränkt durch den starken Fokus auf wissenschaftliche Berechnungen	Hohes Potenzial durch standardisierte Schnittstellen und Möglichkeit der dynamischen Provisionierung

Tabelle 2-1: Gegenüberstellung von Cluster, Grid und Cloud Computing

Wie Jim Gray in [Gra03] hervorhebt, sind die stärksten Treiber für diese Technologien oft ökonomischer Natur. Im Bereich Cloud Computing ergeben sich beispielsweise durch Möglichkeiten der Finanzierung über Werbung gänzlich neue Geschäftsmodelle und Monetarisierungsmöglichkeiten, die bei der Berechnung des Return-on-Investment berücksichtigt werden müssen.

2.2.2 Virtualisierung

In [BKN+10] wird Virtualisierung wie folgt umschrieben: Bei Virtualisierung wird auf einem Pool von IT-Ressourcen eine abstrakte, logische Schicht gezogen, um eine Unabhängigkeit von der konkreten Ausgestaltung der Ressourcen zu erhalten. Der Pool kann auf Hardware- (Speicher, Prozessor, Netzwerk etc.), Betriebssystem- oder Anwendungsebene gebildet werden. Bei Bedarf können dann logische Elemente aus dem Pool bereitgestellt werden.

Als Nachteil ergibt sich ein gewisser Overhead durch die Virtualisierungsschicht. Vorteile sind allerdings eine bessere Ressourcenauslastung, höhere Verfügbarkeit und Flexibilität.

Im Cloud-Kontext wird Virtualisierung (aufgrund der genannten Vorteile) sehr stark eingesetzt. Dort wird Virtualisierung allerdings noch ergänzt durch automatische Provisionierung logischer Ressourcen, geeignete Abrechnungssysteme (die die nutzungsabhängige Abrechnung ermöglichen) und automatisierte Überwachung und Failover (um ausgefallene Ressourcen schnell ersetzen zu können).

2.2.3 High Performance Computing

High Performance Computing (HPC) führt verschiedene Disziplinen der Informatik zusammen: System Administration, Parallel Computing, System- und Softwarearchitektur, Algorithmik etc. Gemeinsam werden Systeme entwickelt, die im Rahmen wissenschaftlicher Forschung hoch-komplexe Berechnungen effizient und effektiv durchführen können.

Durch zunehmende Verbreitung großer Infrastrukturen, die im Wesentlichen aus preisgünstiger „commodity hardware“ bestehen und im Cloud Computing zum Einsatz kommen, verlagern sich Einsatzszenarien von rein wissenschaftlichen Berechnungen zu Geschäftsanwendungen (z.B. für Monte Carlo Simulationen) und werden HPC-Ansätze breiten Anwendergruppen zugänglich. HPC ist also ein wichtiger Treiber für Cloud Computing.

2.3 Marktüberblick

Im Public Cloud Umfeld gibt es eine Reihe großer PaaS-Anbieter, die Entwicklern leistungsfähige Dienste für eigene Cloud Services bereitstellen. Einfachere Vergleiche zwischen diesen Angeboten sind zwar möglich, wie dies von Kossmann u.a. in [KKL10] für Kennzahlen wie Performanz und Skalierbarkeit gezeigt wird, eine Entscheidung darüber, welches der bessere Dienst ist, hängt von vielen Faktoren ab. Wenngleich die Funktionalitäten zum Teil sehr ähnlich sind, unterscheiden sie sich doch in den Programmiermodellen, weshalb ein Wechsel des Anbieters mitunter schwierig ist, Entwickler also eine gewisse Bindung zu dem betreffenden Anbieter eingehen müssen. Bei der Auswahl des Cloud Anbieters muss demnach ein Abwägen zwischen Anbieterunabhängigkeit und Entwicklerproduktivität erfolgen. Im Folgenden sollen die Angebote der derzeit größten Anbieter Amazon, Microsoft und Google kurz beschrieben werden.

2.3.1 Amazon Web Services (AWS)

Amazon bietet mit AWS eine Sammlung von Cloud Services an, mit deren Hilfe Infrastrukturdienste (CPU, Storage, Netzwerk) genutzt, eigene Umgebungen aufgebaut und darauf eigene Anwendungen ausgeführt werden können [Aws15]. Eigene Ablaufumgebungen können in Form von Amazon Machine Images (AMI) erstellt werden. Dabei können Entwickler flexibel eigene Virtuelle Maschinen erstellen und diese dann in Amazons Elastic Compute Cloud (EC2) Umgebung ausführen. Damit wird es möglich, quasi beliebige Programmiermodelle zu verwenden, solange die entsprechend vorausgesetzten Frameworks, Bibliotheken etc. in das eingesetzte AMI installiert werden.

AWS umfasst darüber hinaus eine Reihe von Cloud Services, die in eigenen Anwendungen genutzt werden können. Unter anderem sind dies:

- *Rechenleistung* – Elastic Compute Cloud (EC2, virtuelle Maschinen), Elastic MapReduce (Big Data Berechnungen auf Basis des Hadoop Frameworks)
- *Netzwerkdienste* – Virtual Private Cloud (virtuelles Netzwerk von EC2-Instanzen), Route 53 (Domain Name System)
- *Speicherdienste* – S3 (Skalierbarer Speicher für große Binärdaten), Glacier (kostengünstiger Archivspeicher), Elastic Block Store (Speicherlaufwerke für EC2-Instanzen)
- *Datenbankdienste* – DynamoDB (NoSQL-Datenbank), ElastiCache (In-Memory Cache), Relational Database Service (RDS, relationale Datenbank), Redshift (Warehouse Datenbank), SimpleDB (einfache Datenbank für strukturierte Daten)
- *Anwendungsdienste* – CloudSearch (Volltextsuche), Simple Email Service, Simple Notification Service (Service für den Versand von Push Nachrichten)

2.3.2 Google AppEngine

Unter der Marke AppEngine werden Googles Public Cloud Dienste zusammengefasst [Goo14]. Herzstück ist eine Laufzeitumgebung für eigene Web-basierte Cloud Services, die in Python, Java, PHP und Go implementiert werden und dabei auch die anderen Dienste der Plattform nutzen können. Die Dienste der Plattform können über zwei Abstraktionsebenen angesprochen werden: eine low-level API erlaubt einen direkten Zugriff auf die Google Infrastruktur. Eine high-level API lehnt sich im Falle von Java an etablierte Java Packages an, so dass erfahrene Java-Programmierer mit aus der Java Standard Edition (SE) bekannten Klassen (z.B. Java Data Objects für den Zugriff auf den Datastore, javax.mail (JavaMail für E-Mail-Funktionalität) arbeiten können.

Die AppEngine bietet Entwicklern einige Plattformdienste, die auf die Google Infrastruktur zurückgreifen. Unter anderem:

- *Rechenleistung* – AppEngine (PaaS-Runtime), Compute Engine (virtuelle Maschinen)

- *Netzwerkdienste* – Loadbalancing, VPN
- *Speicherdienste* – Storage (Objektspeicherdienst), Datastore (NoSQL-Datenbank)
- *Datenbankdienste* – SQL (MySQL-Datenbank)
- *Anwendungsdienste* – Mail, Translate

Das Programmiermodell der AppEngine erfordert es, dass Anwendungen spezielle Anforderungen erfüllen. So müssen Requests kurzlebig und synchron gestellt werden. Bei Zeitüberschreitungen bricht die Umgebung die Verarbeitung der Anweisungen ab. Langlebige Verarbeitungsschritte müssen über Task Queues und Cron Jobs abgewickelt werden.

2.3.3 Microsoft Azure

Microsoft Azure (ehemals „Windows Azure“) ist Microsofts Public Cloud Plattform für Entwickler und Administratoren, die eigene Softwareanwendungen ganz oder in Teilen in der Cloud ausführen möchten [Mic15]. Die Plattform umfasst eine Reihe von Cloud Services, die einzeln oder in Kombination genutzt werden können. Zu diesen Services gehören unter anderem solche, mit denen Anwendungen in der Cloud ausgeführt, andere, mit denen unterschiedliche Daten in der Cloud gespeichert und verarbeitet und wieder andere, mit denen Cloud-Elemente sicher mit Ressourcen aus einem eigenen Rechenzentrum bzw. lokal ausgeführten Client-Anwendungen verknüpft werden können. Tatsächlich sind derartige Hybrid-Lösungen, bei denen einzelne Anwendungsteile in der Cloud, andere Teile in einer lokalen Ausführungsumgebung zusammen betrieben werden, eine große Stärke von Microsoft Azure.

Zur Erstellung und Migration Azure-basierter Anwendungen stehen verschiedene Entwicklungs- und Management-Umgebungen (Visual Studio, Eclipse, System Center, …) und Programmiertechnologien (.NET, Java, PHP, Python, …) mit entsprechenden APIs zur Verfügung.

Zur Ausführung von Anwendungen in der Cloud stellt Microsoft Azure drei Modelle zur Verfügung: Web Apps, Cloud Services und Virtual Machines. Web Apps bieten dabei eine einfache Hosting-Möglichkeit für Web-Anwendungen, Cloud Services eignen sich für Aufbau und Betrieb hoch-skalierbarer, hoch-verfügbarer Anwendungen bei minimalem Administrationsaufwand, und Virtual Machines erlauben den Betrieb eigener oder vorgefertigter virtueller Maschinen in der Cloud, wobei hier sowohl Windows als auch Linux als Betriebssystem unterstützt werden. Damit wird es beispielsweise auch möglich eine Java Virtual Machine oder Interpreter für Ruby oder Python auf Azure zu betreiben. Einige Azure Services sind:

- *Rechenleistung* – Web Apps (Hosting von Web-Anwendungen), Cloud Services (skalierbare, komponentenbasierte Anwendungen), Virtual Machines (virtuelle Maschinen), HDInsight (Big Data Berechnungen auf Basis des Hadoop Frameworks)

- *Netzwerkdienste* – Virtual Network (virtuelles Netzwerk von VM-Instanzen), Traffic Manager (globaler Loadbalancer), Service Bus
- *Speicherdienste* – Blob Storage (Speicher für große Binärdaten), Table Storage (Key-Value-Speicher), Redis Cache
- *Datenbankdienste* – DocumentDB (Datenbank für JSON-Objekte), SQL Database (relationaler Datenspeicher)
- *Anwendungsdienste* – Azure Search (Volltextsuche), Notification Hub (Service für den Versand von Push Nachrichten), Active Directory (Authentifizierungs- und Verzeichnisdienst), Media Services (Medienverwaltung)

3 Anforderungen an eine integrierte Entwurfs- und Bewertungsmethodik

Mit der Berücksichtigung der Cloud als Betriebsalternative wächst das Spektrum der Möglichkeiten für die Ausführung von Softwarekomponenten. Beim Entwurf einer Softwarearchitektur ergibt sich hieraus zusätzlicher Handlungsspielraum. Für quasi jede Architekturkomponente kann das Für und Wider der Cloud abgewogen und die Auswirkungen auf die Gesamtarchitektur ermittelt werden. Ziel ist es, für ein zu entwickelndes Softwaresystem die bestmögliche Verteilung der Komponenten auf die zur Verfügung stehenden Ausführungsumgebungen zu ermitteln. Bestmöglich ist in diesem Zusammenhang die Alternative, bei der der Quotient aus dem Nutzen des Softwaresystems und dem zu Implementierung und Betrieb erforderlichen Aufwand maximal ist. Der Nutzen wiederum lässt sich aus dem Grad der Erfüllung funktionaler, qualitativer und ökonomischer Anforderungen ableiten. In [Boo07] betont Booch den Wert nutzenmaximierender Architekturen für die Lebensdauer von Softwaresystemen. Der erwartete Nutzen muss beim Entwurf des Softwaresystems und der Auswahl von Entwurfsalternativen permanent überwacht werden. Benötigt wird ein integrierter Entwurfsprozess, der die Anforderungen an Cloud-basierte verteilte Unternehmensanwendungen berücksichtigen, daraus einen entsprechenden Architekturentwurf ableiten, diesen geeignet formal beschreiben und bewerten bzw. bei einer Bewertung unterstützen kann.

3.1 Anforderungen an Cloud-basierte verteilte Unternehmensanwendungen

Ausgangspunkt für den Entwurf einer Softwarearchitektur ist in der Regel eine Reihe von Anforderungen verschiedener Stakeholder. Die im weiteren Verlauf der Arbeit verwendeten Entwurfsmethoden unterscheiden nach [BCK03] drei Kategorien von Anforderungen:

- *Funktionale Anforderungen* – beschreiben (z.B. in Form von Use Cases) die Funktionalität des Softwaresystems, d.h. „was“ das System später leisten soll. Hierzu gehören auch ökonomische Anforderungen.

- *Qualitative Anforderungen* – legen fest, „wie gut“ das System die gewünschte Funktionalität erbringen soll.

- *Design Constraints* (Entwurfseinschränkungen) – machen Vorgaben für bestimmte Eigenschaften für das System (z.B. Auswahl eines bestimmten Datenbanksystems) ohne Betrachtung der Auswirkungen auf die anderen Anforderungen.

Die in dieser Arbeit betrachteten Softwaresysteme sind aus verteilten Komponenten aufgebaut. Dabei können einzelne oder auch alle Komponenten auf einer Cloud-Plattform ausgeführt werden, und es muss die Frage nach den jeweiligen Konsequenzen für die gegebenen Anforderungen beantwortet werden. Diese Fragestellung ist insbesondere dann relevant, wenn zwischen dem Erfüllungsgrad verschiedener Anforderungen abgewogen werden muss, wie Barbacci u.a. in [BKL+95] bereits treffend beschrieben haben. Dabei hat die Disziplin der Softwarearchitektur inzwischen nach Booch ([Boo06]) sowie Shaw und Clements ([SC06]) einen Reifegrad erreicht, in dem Methodiken entstehen, die ein strukturiertes Vorgehen für einen anforderungsoptimierenden Architekturentwurf vorgeben.

3.2 Entwurf Cloud-basierter verteilter Unternehmensanwendungen

Der zu entwickelnde Entwurfsprozess soll bei der Entscheidung für eine Ausführungsumgebung den oder die verantwortlichen Software-Architekten dabei unterstützen, die Konsequenzen für die einzelnen Anforderungen zu dokumentieren und zu bewerten. Insbesondere soll er bei der Beantwortung der Frage helfen, ob ein Einsatz der Cloud sinnvoll ist bzw. positiven Einfluss auf die Erfüllung von Anforderungen hat. Ohne Hilfestellung in Form einer Entwurfsmethodik wird die Entscheidung für oder gegen den Einsatz der Cloud häufig subjektiv aus Sicht einzelner Stakeholder getroffen. Oft erfolgt diese „aus dem Bauch heraus“ und ist im Nachhinein schwer nachvollziehbar. So wird Cloud Computing häufig aufgrund von Sicherheitsbedenken abgelehnt, ohne dass das geforderte Sicherheitsniveau zuvor in Form einer formalen Anforderung dokumentiert worden wäre. Dies nimmt dem Architekten die Möglichkeit durch Einsatz zusätzlicher Sicherheitskomponenten (z.B. Verschlüsselung von Daten) eine solche Anforderung auch beim Einsatz der Cloud zu erfüllen und für diese Alternative die Auswirkungen auf andere Anforderungen wie Wartbarkeit formal zu beurteilen.

Der Entwurfsprozess soll Gründe für Entscheidungen transparent machen und dabei helfen, diese entsprechend zu dokumentieren. Am Ende soll er dann Alternativen für verschiedene Komponenten bzw. deren Verteilung – insbesondere auch die Ausführung in der Cloud - beschreiben und den unter den gegebenen Anforderungen „bestmöglichen“ Architekturentwurf liefern. Bereits zum Entwurfszeitpunkt (und nicht erst bei der Implementierung) soll so eine fundierte Entscheidung pro oder contra Cloud ermöglicht werden.

3.3 Formale Beschreibung von Softwarearchitekturen

Im Rahmen des Entwurfsprozesses soll die zu erstellende Softwarearchitektur formal beschrieben werden. Auch die Prozessschritte sollen soweit formalisiert ablaufen, dass der Entwurf der Architektur aus den Anforderungen, die Beurteilung der Zwischenergebnisse und

die Herleitung der finalen Architekturbeschreibung nachvollziehbar und dokumentierbar werden. Hierzu gibt es beispielsweise mit dem Zachman Framework [Zac87], dem Department of Defense Architecture Framework (DoDAF) [DoD11] dem The Open Group Architecture Framework (TOGAF) [OG09] und einigen weiteren bereits interessante Ansätze, denen jedoch mitunter konkrete Vorgaben für die Dokumentation der Architektur fehlen, oder deren Komplexität sie für eine pragmatische Anwendung in kleineren Entwicklungsvorhaben ausschließt. Buckl u.a. bestätigen dies in ihren Ausführungen zum Management von Enterprise Architekturen: *„management frameworks, like Zachman […], TOGAF […], etc., are usually either too abstract and therefore not implementable, or too extensive to be used in practice, as they have to be utilized as a whole.“* [BEL+08]. Farwick u.a. stellen im Rahmen ihrer Forschungen ebenfalls das Fehlen ausreichender Detaillierung bei der Beschreibung von Enterprise Architekturen fest (*„E.g. in the most-widely applied framework, The Open Group Architecture Framework (TOGAF), the documentation is only briefly covered“* [FBH+13]).

Die formale Beschreibung soll perspektivisch auch Werkzeugunterstützung ermöglichen, um dem verantwortlichen Architekten bei der Entwicklung des Architekturentwurfs zu helfen (z.B. durch automatisierte Auswertung von Architekturalternativen, Dokumentation von Entscheidungen, …). Durch die Unterstützung können Planungsaufwände reduziert und dadurch der Scope der Architekturbewertung erweitert werden. Diese muss häufig unter Zeitdruck stattfinden, wie in [BCK03] beschrieben.

Darüber hinaus soll die Granularität der Architekturbeschreibung flexibel gewählt werden können, um bei einfacheren Softwaresystemen nicht unnötig Overhead zu erzeugen und Beschreibungs- und Prozesstechnik auch in kleineren Softwareprojekten einsetzen zu können.

3.4 Bewertung von Softwarearchitekturen

Alle Schritte zur Bereitstellung eines Softwaresystems, von der Planung bis hin zu Implementierung, Test und Betrieb, finden unter begrenzter Ressourcenausstattung statt. Es ist selbstverständlich, dass bei allen Beteiligten der Wunsch besteht, mit den bereitgestellten Ressourcen, das „bestmögliche“ System bereitzustellen. Dabei sollen möglichst frühzeitig im Projektverlauf die Weichen für das Erreichen dieses Ziels gestellt werden, d.h. bereits beim Entwurf der Softwarearchitektur sollen Entscheidungen auf deren Auswirkungen auf den Wert des späteren Systems beurteilt werden. Fehlentscheidungen in dieser Projektphase können in späteren Phasen nur mit deutlichem Mehraufwand korrigiert werden. Knodel und Naab berichten in [KN14] und [KN14-1] von ihren Erfahrungen aus zahlreichen Evaluierungsprojekten. Sie erkennen an, dass es bereits viele Ansätze zur Modellierung und auch Evaluierung von Softwarearchitekturen gibt, jedoch scheiterten viele Vorhaben an der Komplexität der eingesetzten Methoden und fehlender Integration in den Entwurfsprozess. Pragmatismus und

Transparenz für alle Stakeholder sind deshalb entscheidend für den Erfolg und die Akzeptanz von Architekturevaluierungen.

Der Wunsch nach der Bewertung einer Softwarearchitektur wirft natürlich sofort die Frage auf, nach welchen Kriterien die Beurteilung des Wertes erfolgen soll bzw. was den Wert der Architektur beschreibt.

3.4.1 Einflussgrößen auf die Bewertung

Der Mehrwert eines Softwaresystems wird im Rahmen dieser Arbeit als der Quotient aus dem Nutzen des Systems und dem Aufwand zur Implementierung und Bereitstellung des Systems angesehen. D.h. je höher der Nutzen pro Aufwandseinheit ist, desto höher ist der Wert bzw. je höher der Aufwand bei gleichem Nutzen, desto geringer ist der Wert. Dieser Wert wird häufig als Return-on-Investment (kurz: ROI) bezeichnet [Inv15].

> Der **Return-on-Investment (ROI)** eines Systems ist das Verhältnis aus dem Nutzen des Systems (Benefit) und der Höhe der zu tätigenden Investition, d.h. den Kosten (Cost), kurz
>
> $$ROI_{System} = \frac{Benefit_{System}}{Cost_{System}}$$

Ziel des zu entwickelnden Entwurfsprozesses ist es nun, aus gegebenen Architekturalternativen diejenige mit dem höchsten ROI-Wert zu ermitteln.

3.4.2 Bewertung des Nutzens einer Softwarearchitektur

Sofern bestehende Constraints erfüllt sind, ergibt sich Nutzen eines Softwaresystems wiederum aus dem Grad der Erfüllung der definierten Anforderungen, d.h. die funktionalen, qualitativen und ökonomischen Anforderungen müssen bestmöglich erfüllt sein, um einen maximalen ROI zu erzielen. Obige Formel für den ROI kann demnach wie folgt erweitert werden (wobei Voraussetzung ist, dass etwaige Constraints erfüllt sind).

$$ROI_{System} = \frac{Benefit_{System}^{functional} + Benefit_{System}^{Quality}}{Cost_{System}}$$

Steht im Rahmen des Architekturentwurfs eine Entscheidung zwischen mehreren Architekturalternativen an, muss diese auf etwaige Auswirkungen auf Nutzen- und Kostenfaktoren geprüft werden. Dabei ist zu berücksichtigen, dass die durch die Anforderungen formulierten Ziele nicht unabhängig voneinander sind und sogar gegenläufig sein können. Eine Alterna-

tive, die funktional besser ist, kann beim Qualitätsattribut der Wartbarkeit schlechter abschneiden. Der Entwurfsprozess muss dem Architekten dabei helfen, zwischen den Zielen abzuwägen. Hierzu müssen die Nutzenfaktoren vergleichbar gemacht werden, d.h. auch abstrakt formulierte Ziele wie Wartbarkeit, Bedienbarkeit etc. müssen auf vergleichbare Größen gebracht werden, um eine konsistente Interpretation der Ziele zu ermöglichen.

3.4.3 Bewertung der Aufwände für eine Softwarearchitektur

Die Bereitstellung eines Softwaresystems ist mit Kosten verbunden. Diese können unterschiedlicher Natur sein und fallen in allen Projektphasen an. Närman u.a. stellen in [NSS+09] ein Framework vor, mit dem Kosten für ein IT-Vorhaben abgeschätzt werden können. Sie berücksichtigen dabei unter anderem technische Kosten (Entwicklungskosten, Integrationskosten) und organisatorische Kosten im Rahmen der Einführung (temporär sinkende Produktivität der Anwender, Schulungskosten etc.).

Analog den Nutzenfaktoren müssen die Kosten für Architekturalternativen bewertet werden, da sie letztlich ebenfalls Einfluss auf den ROI des Systems haben. Neben konkret messbaren Betriebskosten (z.B. Kosten von Speicherdiensten in der Cloud), gibt es hier auch Kostenpositionen (z.B. Implementierungskosten), bei denen die Expertise der Beteiligten gefragt ist, diese zum Entwurfszeitpunkt abzuschätzen. Im Rahmen dieser Arbeit sollen die Kostenabschätzungen sehr einfach gehalten werden. Auf den Einsatz komplexerer Frameworks soll verzichtet werden.

4 Stand der Forschung

Für den Entwurfs- und Bewertungsprozess auf der einen Seite sowie für die Beschreibung einer Software-Architektur auf der anderen Seite gibt es jeweils bereits eine Reihe von Methoden. Diese haben sich grundsätzlich in der Praxis bewährt. Allerdings sind sie nur unzureichend auf Situationen ausgelegt, in denen unter den am Entwurf beteiligten Personen Einigkeit zum Grundaufbau der Architektur herrscht und lediglich abweichende Meinungen zur Verteilung einzelner Softwarekomponenten bestehen.

Darüber hinaus sind die Methoden nur zum Teil kombinierbar. In diesem Kapitel werden eine bestehende Beschreibungsmethode sowie etablierte Vorgehensmodelle vorgestellt, die in den folgenden Kapiteln zu einer integrierten Methode zusammengeführt werden.

Ziel dieser Methode ist es, für ein konkretes Softwareprojekt die Beschreibung einer nach subjektiven Kriterien „bestmöglichen" Architektur erstellen zu können, die dann Entwicklern als Grundlage für die Implementierung dient. Im Zentrum stehen dabei die Beschreibung der Architektur, der Entwurfskriterien und die Dokumentation der Entscheidungen für einzelne Architekturalternativen.

4.1 Beschreibung von Software-Architekturen

4.1.1 Methoden zur Beschreibung von Software-Architekturen

Zur Beschreibung von Software-Architekturen gibt es bereits eine Reihe von Methoden, die sich etabliert haben. Das in [Zac87] vorgestellte Zachman Framework stellt einen relativ allgemein gehaltenen Ordnungsrahmen für Softwaresysteme auf. Der Formalisierungsgrad ist verhältnismäßig gering. Die Integration in Vorgehensmodelle wird bewusst offen gehalten. Das Department of Defense Architecture Framework (DoDAF) in [DoD11] eignet sich zur Modellierung komplexer Systeme und legt dabei einen Fokus auf die Daten der Architektur. Das in [OG09] skizzierte, durch die Open Group entwickelte The Open Group Architecture Framework (TOGAF) hat sich als ein Industriestandard zur Beschreibung von Softwarearchitekturen etabliert, die zugehörige Modellierungssprache ArchiMate aus [JLQ+11] erlaubt die grafische Darstellung von Architekturen. Närmann u.a. führen das Konzept von ArchiMate in [NSJ+08] weiter, um entsprechende Architekturmodelle hinsichtlich verschiedener Qualitätsattribute zu analysieren. Sehr vielversprechend ist der von Heesch u.a. in [HAH12] skizzierte Ansatz, bei dem vier Sichten zur Dokumentation von Entscheidungen verwendet werden, eine Sicht auf Entscheidungsdetails, eine Sicht auf die Beziehungen zwischen Entscheidungen, eine Sicht auf die zeitliche Abfolge von Entscheidungen und eine Sicht auf die Einbeziehung

von Stakeholdern. In [HAH12-1] erweitern die Autoren diesen Ansatz um eine Sicht auf Einflussfaktoren für Entscheidungen. Van Heesch u.a. beschreiben in [HP13] ebenfalls einen Ansatz zur Dokumentation von aus Architektursicht relevanten Anforderungen („architecturally significant requirements (ASRs)“ [HP13]) und zugehörigen Entscheidungen zum Architekturentwurf („architectural design decisions (ADDs)“ [HP13]). Aber auch hier entspricht die Dokumentation eher einem Protokoll der Entscheidungen, das einen eindeutigen Architekturentwurf ergänzt. Es fehlen die Möglichkeiten, im Architekturentwurf Alternativen zu behalten und mit den Ergebnissen der Bewertung zu verknüpfen. Iacobucci u.a. stellen in [IM11] eine Modellierungssprache, die Capability Focused Modeling Language (CFML) vor, mit deren Hilfe Architekturalternativen im Architekturentwurf systematisch erzeugt und bewertet werden können. Zwar können damit auch recht umfangreiche, komplexe Architekturen evaluiert werden, jedoch ist der Ansatz schnell zu kompliziert, um ihn unter Beteiligung aller Stakeholder durchzuführen. Für einige Evaluierungsschritte ist die Zuhilfenahme von Cloud Technologien angedacht. Schmidt u.a. sehen in diesem Kontext in [SMW+14] auch den Einsatz von Big Data Technologien an, um auf fein-granularer Ebene Architekturanalysen durchführen zu können. Bei ausreichender Detailtiefe der Architekturbeschreibungen sicher sinnvoll, auf grob-granularer Ebene, auf der sich häufig die Diskussionen über das Pro und Contra des Einsatzes von Cloud Technologien bewegen, erscheint dieser Ansatz aber viel zu komplex.

Die genannten Methoden eignen sich alle gut zur Beschreibung von Softwarearchitekturen und damit zur Dokumentation und Weitergabe von Informationen über Softwaresysteme. Was ihnen allen jedoch fehlt, sind passende Integrationen in Vorgehensmodelle sowie die Möglichkeit, Alternativen im Softwareentwurf dokumentieren und bewerten zu können. Es fehlt darüber hinaus eine Integration in formale Vorgehensmethoden, die dabei unterstützen könnten, Entscheidungen zu Alternativen herbeizuführen und prozessbegleitend zu erfassen. Diese Defizite werden in dieser Arbeit im Rahmen der integrierten Entwurfs- und Bewertungsmethode adressiert.

4.1.2 Das Composite Application Framework nach Mietzner

Ralph Mietzner stellt in [Mie10] mit dem Composite Application Framework (Café) eine Methode zur Definition und Provisionierung zusammengesetzter, verteilter Softwaresysteme vor. Diese eignet sich aus folgenden Gründen hervorragend für den in dieser Arbeit entwickelten Ansatz zu Entwurf und Bewertung entsprechender Softwarearchitekturen:

- Die Methode ermöglicht über ein Anwendungsmodell eine vollständige Beschreibung eines Softwaresystems inklusive dessen Architektur und für die Bewertung relevanter Implementierungsdetails.
- Innerhalb eines Anwendungsmodells können sogenannte Variabilitätspunkte (Variability Points) beschrieben werden, über die implementierungs- und betriebsrelevante

Varianten beschrieben werden können. Hierüber können bei der Bewertung Alternativen für die Verteilung, d.h. verschiedene Ausführungsumgebungen (lokaler Betrieb vs. Betrieb einer Anwendungskomponente in der Cloud) durchgespielt werden.

- Für die Beschreibung des Systems wird eine Sprache zur Serialisierung des Anwendungs- und des Variability-Metamodells vorgestellt. Diese wird in den folgenden Abschnitten um Möglichkeiten zur Bewertung des Systems erweitert.

Die in Café enthaltene Beschreibungsmethode ermöglicht es, für ein und dasselbe Anwendungsszenario Architekturalternativen zu beschreiben. Jede der Alternativen setzt also die gleiche Funktionalität um. Dies ist in Gegensatz zu Entwurfstechniken im Zusammenhang mit Produktlinien zu sehen. Zwar geht dort, wie Clements u.a. in [CN12] beschreiben, der Entwurf von einem gemeinsamen Anwendungskern aus, die Alternativen sollen allerdings unterschiedliche Funktionalitäten umsetzen. Im Folgenden sollen die wichtigsten, für diese Arbeit relevanten Definitionen aus Café aufgeführt werden und der Bezug zu den Betrachtungen dieser Arbeit hergestellt werden.

Das sogenannte Anwendungsmodell beschreibt eine Anwendung, deren Komponenten, sowie Implementierungs- und Deployment-Aspekte. Formal wird das Anwendungsmodell in [Mie10] wie folgt definiert:

> Gegeben sei die Menge aller Anwendungsmodelle $AM = \{..., a_i, ...\}$. Ein **Anwendungsmodell** $a \in AM$ ist ein Tupel
>
> $$a = (C, I, F, B, D, type, impl, implT, files, blockOf, p)$$
>
> wobei für die einzelnen Elemente folgendes gilt:
>
C	ist eine Menge von Komponenten,
I	ist eine Menge von Implementierungen,
F	ist eine Menge von Dateien,
B	ist eine Menge von Building Blocks,
D	ist eine Menge von Deployment-Beziehungen,
$type$	ist eine Abbildung $(Komponente \rightarrow Komponententyp)$,
$impl$	ist eine Abbildung $(Komponente \rightarrow Implementierung)$,
$implT$	ist eine Abbildung $(Implementierung \rightarrow Implementierungstyp)$,
$files$	ist eine Abbildung $(Implementierung \rightarrow Datei)$,
$blockOf$	ist eine Abbildung $(Building\ Block \rightarrow Datei)$,
p	ist eine Abbildung $(Komponente \rightarrow Multimandanten\ Pattern)$.

Komponenten werden dabei ganz allgemein als Bausteine betrachtet, aus denen das Softwaresystem aufgebaut ist. Dies sind sowohl eigenimplementierte Anwendungsteile als auch Elemente, die von Zulieferern (z.B. Cloud-Anbietern) bereitgestellt werden. Zwischen Softwarekomponenten können gerichtete Deployment-Beziehungen bestehen, d.h. das Vorhandensein einer Komponente bedingt in diesem Fall das Vorhandensein einer weiteren Komponente. Jede Komponente besitzt einen Komponententyp, der von der eigentlichen Implementierung abstrahiert. Mit dem Komponententyp werden bestimmte Funktionalitäten unabhängig von einer konkreten Implementierung assoziiert. Eine ebenfalls modellierbare Multimandantenfähigkeit einer Anwendung soll in der vorliegenden Arbeit nicht weiter betrachtet werden.

Die Implementierung einer Komponente beschreibt, auf welchem Weg die Funktionalität der Komponente bereitgestellt wird. Zum einen kann die Bereitstellung extern oder durch einen Provider erfolgen (in diesem Fall werden technische Details der Implementierung nicht weiter beschrieben), oder durch Einsatz eines bestimmten Implementierungstyps. Dieser legt fest, welche Technologie zur Implementierung verwendet werden soll, und gibt damit bereits Hinweise darauf, welche Implementierungsartefakte (Dateien, Laufzeitumgebungen etc.) zur Umsetzung benötigt werden. In diesem Fall werden entsprechende Dateien mit der Implementierung assoziiert, die Konfigurationsinformationen, Programmcode etc. enthalten können. Dateien können aus Building Blocks bestehen. Diese sind Teile von Dateien, die eindeutig referenziert werden können. Hierüber wird es möglich, Alternativen zu definieren. So kann ein Building Block beispielsweise eine Konfigurationseinstellung sein, die je nach Ausführungsumgebung anders gesetzt werden muss.

Abb. 4-1 skizziert, wie die einzelnen Elemente des Architekturmodells nach [Mie10] grafisch notiert werden können.

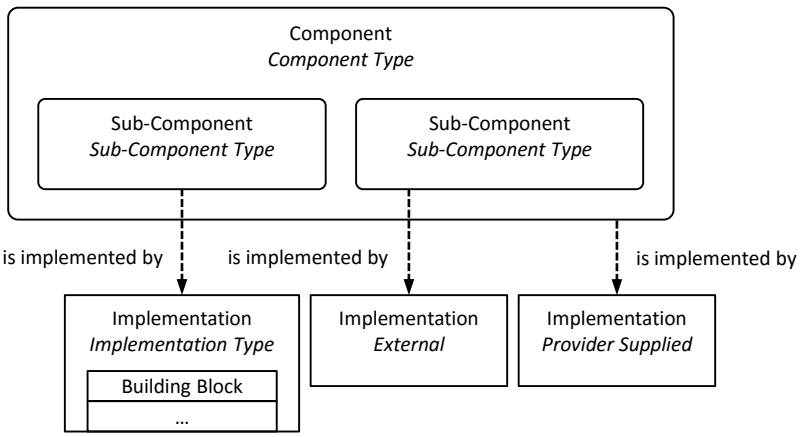

Abb. 4-1: Notation für ein Anwendungsmodell, nach [Mie10]

In Abb. 4-2 ist das Anwendungsmodell einer Beispielanwendung zu sehen. Die Anwendung besteht aus einer UI Komponente, die über zwei Dateien implementiert wird, und einer SQL Datenbank, die über ein SQL-Skript definiert wird. Die Ausführungsumgebungen – der Microsoft Azure Compute Service und das relationale Datenbanksystem – werden durch einen Provider, in diesem Fall Microsoft, bereitgestellt. Die UI Komponente ist von der SQL Datenbank abhängig.

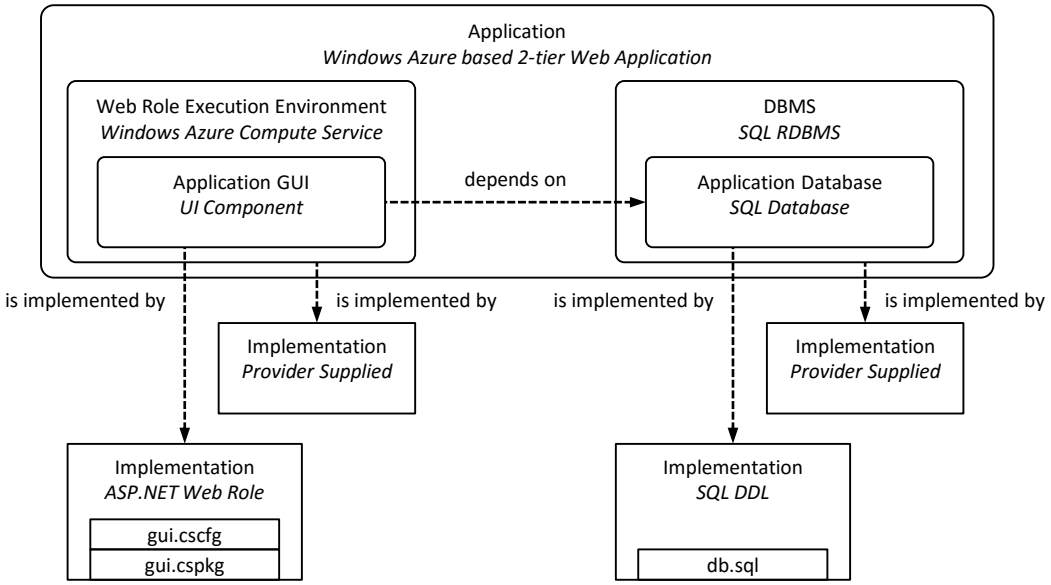

Abb. 4-2: Anwendungsmodell einer Beispielanwendung

Im Rahmen dieser Arbeit soll die Möglichkeit untersucht werden, Alternativen bei Entwurf und Implementierung zu bewerten. Hierzu ist es erforderlich, Alternativen im Anwendungsentwurf modellieren zu können. Im Beispiel in Abb. 4-2 ist es vorstellbar, die UI Komponente nicht in der Cloud sondern alternativ in einem lokalen Rechenzentrum zu betreiben. Dies muss bei der Modellierung der Anwendung berücksichtigt werden. Zu diesem Zweck können die von Mietzner in [Mie10] beschriebenen Variabilitätsmodelle genutzt werden. Diese werden wie folgt definiert:

> Gegeben sei die Menge aller Variabilitätsmodelle $VM = \{\dots, v_i, \dots\}$. Ein **Variabilitätsmodell** $V \in VM$ ist ein Tupel
>
> $$V = \begin{pmatrix} VP, VPR, Dp, L, Alt, EC, Co, E, ref, locators, alternatives, ec, cond, \\ enAlts, expression, buildingBlocks, roles, VPrefine, AltR, ecR \end{pmatrix}$$
>
> wobei für die einzelnen Elemente folgendes gilt:

VP	ist eine Menge von Variabilitätspunkten,
VPR	ist eine Menge von Variabilitätspunktverfeinerungen,
Dp	ist eine Menge von Abhängigkeiten zwischen Variabilitätspunkten,
L	ist eine Menge von Locators,
Alt	ist eine Menge von Alternativen,
EC	ist eine Menge von Aktivierungsbedingungen,
Co	ist eine Menge von Bedingungen,
E	ist eine Menge von Ausdrücken,
ref	ist eine Abbildung ($Variabilitätsmodell \rightarrow Variabilitätsmodell$),
$locators$	ist eine Abbildung ($locator \rightarrow vp$),
$alternatives$	ist eine Abbildung ($alt \rightarrow vp$),
ec	ist eine Abbildung ($ec \rightarrow vp$),
$cond$	ist eine Abbildung ($co \rightarrow ec$),
$enAlts$	ist eine Abbildung ($e \rightarrow ec$),
$expression$	ist eine Abbildung ($e \rightarrow alt$),
$buildingBlocks$	ist eine Abbildung ($Bulding\ Block\ Locator \rightarrow Building\ Block$),
$roles$	ist eine Abbildung ($vp \rightarrow Rolle$),
$VPrefine$	ist eine Abbildung ($vp \rightarrow vpr$),
$AltR$	ist eine Abbildung ($alt \rightarrow alt$),
ecR	ist eine Abbildung ($ec \rightarrow ec$).

Ein Variabilitätsmodell besteht aus Variabilitätspunkten, die Alternativen, d.h. mögliche Ausprägungen in Komponenten eines Anwendungsmodells beschreiben. Die Beschreibung der Alternativen erfolgt über Ausdrücke. So kann ein Variabilitätspunkt, der auf die Datenbankimplementierung zeigt, eine lokal betriebene SQL-Server-Datenbank oder alternativ eine in der Cloud betriebene Microsoft Azure SQL Database beschreiben. Über einen im Variabilitätspunkt enthaltenen Locator wird dasjenige Artefakt des Modells adressiert, auf den sich die Variabilität bezieht. Dies kann eine ganze Komponente oder ein Building Block der Implementierung einer Komponente sein. Für die Auswahl einer Alternative können Bedingungen definiert sein, die zur Auswahl einer Alternative erfüllt sein müssen. Zwischen Variabilitätspunkten können Abhängigkeiten definiert werden.

Neben einem konkreten Variabilitätsmodell, das sich auf ein bestimmtes Softwaresystem bezieht, sind auch abstrakte Variabilitätsmodelle möglich. Diese beschreiben Variabilitäten von Komponententypen und können einen allgemeinen Rahmen für den Entwurf eines Softwaresystems setzen. So können in einem abstrakten Variabilitätsmodell beispielsweise alle generell zulässigen Datenbanksysteme beschrieben werden. Abstrakte Modelle können in konkrete Modelle importiert und dort verfeinert werden. Dabei können importierte Variabilitätspunkte abgeleitet bzw. verfeinert ($VPrefine$) werden, d.h. Alternativen eingeschränkt, Aktivierungs-

bedingungen konkretisiert werden etc. So könnte im konkreten Variabilitätsmodell die Auswahl der Datenbanksysteme weiter eingeschränkt werden. Es ist denkbar, dass ein abstraktes Variabilitätsmodell von einem Enterprise Architekten erstellt wird, der unternehmensweite Vorgaben machen möchte, und das konkrete Modell von einem Software-Architekten, der eine konkrete Anwendung entwirft und dabei das abstrakte Modell importiert und verfeinert, d.h. für das Implementierungsszenario konkretisiert.

Abb. 4-3 veranschaulicht die Zusammenhänge zwischen Variabilitäts- und Anwendungsmodell sowie einer zugehörigen Implementierung.

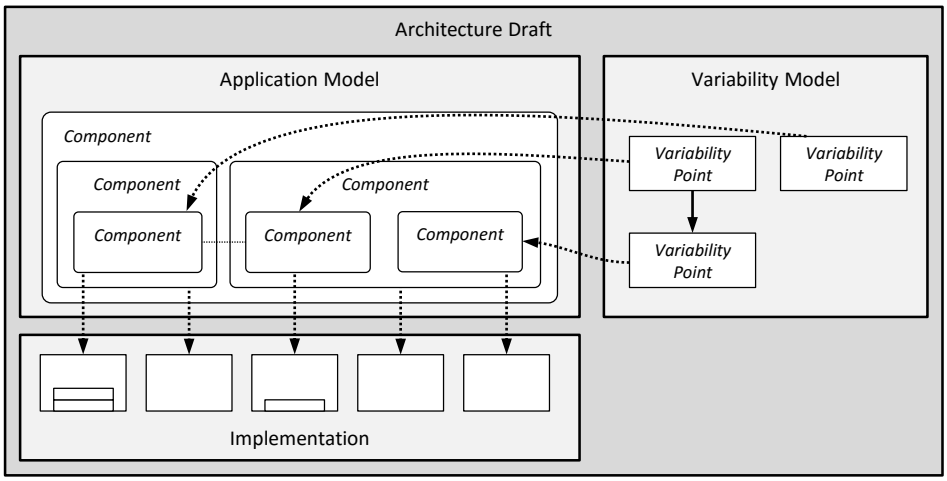

Abb. 4-3: Elemente eines Architekturentwurfs nach [Mie10]

Im Kontext dieser Arbeit erfüllt dieses Variabilitätsmodell einige wichtige Eigenschaften, aufgrund derer es sich sehr gut zur Beschreibung von Softwarearchitekturen eignet, in denen im Rahmen von Entwurf und Bewertung Alternativen für Implementierung und Bereitstellung beschrieben werden sollen:

- Der Ansatz ist technologieunabhängig. Somit können verschiedene Cloud Provider, deren Services sich ja zum Teil in Abstraktionsebene und Technologie deutlich unterscheiden, in einem Modell berücksichtigt werden.
- Das Variabilitätsmodell sieht im Zusammenspiel mit dem Anwendungsmodell eine Beschreibung eines Softwaresystems vor, die technisch genug ist, um systemseitig eine Bewertung verschiedener Alternativen zu unterstützen.
- Die Beschreibungsmöglichkeiten sind flexibel genug, um auf verschiedene Artefakte der Anwendungsarchitektur (von einzelnen Konfigurationseinstellungen bis hin zu

ganzen Komponenten) anwendbar zu sein. Damit können recht einfache Alternativen wie der Austausch einer lokalen durch eine Cloud Datenbank, bei der im Anwendungscode ggf. nur ein Connection String ausgetauscht werden muss bis hin zum Austausch einer lokalen relationalen durch einen Cloud-basierten Non-SQL-Datenspeicher durchgespielt werden, bei der größere Änderungen in der Persistenzlogik erforderlich sind.

- In Zwischenschritten der Entwicklung und Bewertung der Anwendungsarchitektur kann die Beschreibung ausreichend abstrakt gehalten werden, um bereits in frühen Entwurfsphasen ohne konkrete Implementierungsdetails Alternativen gegenüberstellen zu können.

- Die Beschreibung von Variabilitäten in einem eigenen orthogonalen Variabilitätsmodell hat, wie in [PM07] beschrieben, den Vorteil, dass damit Variabilitäten einfach und überschaubar modelliert werden können. Eine vollständige Integration in das Anwendungsmodell würde dieses unnötig komplex machen.

Die Beschreibung der Softwarearchitektur kann im Rahmen einiger bereits etablierter Entwurfsmethoden gewonnen werden. Da im Rahmen dieser Arbeit ein integriertes Vorgehensmodell erstellt werden soll, müssen die ausgewählten Entwurfsmethoden erweitert werden, um eine durchgängige Beschreibungsmöglichkeit für fachliche Anforderungen, Qualitätsattribute, Nutzenwerte etc. zu haben.

4.2 Entwurf und Bewertung von Software-Architekturen

Lassing u.a. beschreiben in [LRV99], was letztlich Ziele einer Bewertung von Softwarearchitekturen sind: Identifikation und Bemessung von Risiken, die ein Architekturentwurf enthält, sowie als Konsequenz die Schaffung von Vertrauen in einen Entwurf, so dass dieser als Grundlage für Folgeschritte in der Softwareerstellung (Implementierung, Test etc.) sein kann.

4.2.1 Methoden zur Bewertung von Entwurfsentscheidungen

Die methodische Unterstützung bei der Auswahl von Architekturalternativen ist Gegenstand zahlreicher Abhandlungen. Die in [HEA14] vorgestellte Decision Centric Architecture Review Methode (DCAR) unterstützt bei der Modellierung von Anforderungen, hier „forces“ genannt, beschränkt sich darüber hinaus aber auf eine Prozessbeschreibung und ermöglicht keine Modellierung von Abhängigkeiten zwischen Anforderungen und Elementen des Architekturentwurfs. Die Frage nach der Modellierung von Architekturalternativen wird nicht betrachtet. Österlind, Johnson, Karnati et. al. stellen in [OJK+13] eine Möglichkeit vor, über An-

wendung von Schritten der Entscheidungstheorie und Aggregation entsprechend aus priorisierten Anforderungen abgeleiteter Nutzenfunktionen eine Auswahl zwischen Architekturalternativen zu treffen. In größeren Projekten kann dies recht komplex werden und Plausibilitätsprüfungen stark erschweren. Prozessbeteiligte können von diesen mathematischen Modellen leicht überfordert sein. Darüber hinaus sieht diese Methode keine Integration mit der Architekturmodellierung vor. Einen ähnlichen Ansatz verfolgen Giesen und Völker in ihrer in [GV02] vorgestellten Methode, bei der für drei Klassen von Stakeholdern (Manager, Techniker, Anwender) jeweils Nutzenfunktionen aufgestellt werden. Auch dieser Ansatz erscheint aufwändig und zum Teil nur mit computerunterstützten Auswertungen durchführbar. Auch hier ist keine Integration mit dem Architekturmodell vorgesehen. Ähnliches gilt für einen Ansatz von Zayaraz und Thambidurai, der in [ZT05] beschrieben ist. Dieser ähnelt der Cost Benefit Analysis Method (CBAM), d.h. er betrachtet den Nutzen von Architekturalternativen und setzt diesen mit den jeweiligen Aufwänden in Beziehung. Er sieht jedoch keine Risikobetrachtung vor und lässt sich nicht ohne weiteres mit anderen Methoden kombinieren. Franke u.a. verfolgen in [FUS+09] einen Ansatz, bei dem Entscheidungen auf Basis größerer Architektur-Metamodelle mit Hilfe von Entscheidungsbäumen vereinfacht werden sollen. Auch dieser Ansatz ist vergleichsweise komplex. Darüber hinaus fehlt die Integration in den Softwareentwurfsprozess. Prem Jain stellt in [Jai11] mit ArchEE eine Use Case-basierten Ansatz vor, bei dem Architekturalternativen über ihr Abschneiden in Simulationsläufen miteinander bzw. mit einer bestehenden Architektur verglichen werden. Dieser aus dem militärischen Bereich stammender Ansatz ist ebenfalls vergleichsweise komplex und eignet sich dann besonders, wenn recht klar ein von allen Beteiligten getragenes Ziel (und keine widerstreitenden Anforderungen) besteht.

4.2.2 Methoden zur Bewertung von Softwarearchitekturen

Matthias Naab beleuchtet in [Naa12] eine zu dieser Arbeit ähnliche Fragestellung, d.h. wie bestehende Entwurfsmethoden für Softwarearchitekturen so angepasst werden können, dass die Qualität von Software erhöht wird. Während Naab dabei speziell auf die Attribute Flexibilität und Wartbarkeit für SOA-basierte Anwendungen fokussiert (er setzt hier auf Vorarbeiten von Arsanjani und Allam ([AA06]) zum methodenbasierten Entwurf von SOAs auf), soll die Sicht in dieser Arbeit etwas allgemeiner gehalten werden, d.h. quasi beliebige Qualitätsanforderungen sollen berücksichtigt werden.

Ionita u.a. beschreiben in [IAH+04] mit SQUASH eine Methode zur strukturierten, Szenariobasierten Analyse von Softwarearchitekturen. Im Vergleich zu der in dieser Arbeit verwendeten Methoden ATAM und CBAM wählen sie einen deutlich komplexeren Ansatz, um Anforderungen gegeneinander abzuwägen. Auch fehlt eine Beschreibungsmethodik für Architek-

turalternativen. Das gleiche Defizit besitzt der Ansatz von Riebisch u.a., der in [RW07] präsentiert wird. Dieser beschreibt ein Vorgehen, das sich an eine andere Methode, die Architecture-Level Modifiability Analysis (ALMA) anlehnt.

Johnson u.a. propagieren in [JJS+07] ein Werkzeug zur Analyse von Softwarearchitekturen. Dabei werden verschiedene, vollständige Architekturen (die jeweils unabhängig voneinander sein können) anhand verschiedener Kriterien untersucht. Die Analyse selbst wird von einem technischen Experten durchgeführt, d.h. nicht alle Stakeholder sind beteiligt. Auch ist es nicht möglich, Architekturen so zu modellieren, dass sie punktuell alternative Umsetzungen vorsehen (z.B. Datenhaltung einmal lokal, einmal in der Cloud).

Eine Reihe bestehender Methoden zur Bewertung von Softwarearchitekturen sehen vor, einen bestehenden Architekturentwurf in Bezug auf verschiedene Qualitätsanforderungen zu evaluieren und eine Einschätzung zur Tauglichkeit (Software Architecture Analysis Method, SAAM), Risiken, Sensitivity Points, Tradeoff Points (Architecture Tradeoff Analysis Method, ATAM), Entwurfstauglichkeit (Active Reviews for Intermediate Designs, ARID) usw. abgeben. Alle Methoden können dabei helfen, den im Rahmen gegebener Anforderungen bestmöglichen Architekturansatz, in [BBK02] auch Architekturtaktik („architectural tactic“) genannt, zu wählen. Unter anderem in [BN12], [RG08], [EHM07], [AZJ04], [BG04], [DN02] und [IHO02] finden sich umfangreiche Gegenüberstellungen verschiedener Bewertungsmethoden. Anhang B enthält hierzu eine Zusammenfassung.

4.2.3 Auswahl für das Forschungsziel geeigneter Methoden

Im Folgenden werden drei Beschreibungs- und Bewertungsmethoden vorgestellt, die sich aus folgenden Gründen als Bausteine für ein durchgängiges Beschreibungs- und Bewertungsmodell eignen:

- Sie ermöglichen die Analyse eines Architekturentwurfs im Hinblick auf beliebige Qualitätsanforderungen [AZJ04].
- In Kombination miteinander helfen sie, Zusammenhänge zwischen Qualitätsanforderungen [EHM07], Entwurfsrisiken sowie Nutzenwerte und Aufwände zu ermitteln.
- Die Beteiligten müssen – mit Ausnahme des Software-Architekten – nicht notwendigerweise über tiefes Wissen zur Implementierungstechnik und dem Evaluierungsprozess verfügen [BN12].
- Zu ihnen gibt es bereits erste Ansätze zur Integration, bei der Ergebnisartefakte einer Methode Eingangsartefakte einer nachfolgenden Methode sind.

Alle drei Methoden wurden am Software Engineering Institute der Carnegie Mellon University entwickelt. Im Rahmen dieser Arbeit sollen diese erweitert und im Sinne eines integrierten Gesamtmodells mit einer durchgängig verwendbaren Beschreibungsnotation für die einzelnen Eingabe- und Ausgabeartefakte versehen werden.

4.2.3.1 Die Attribute-Driven Design Method (ADD)

Die unter anderem in [WBB+06] und [BCK03] beschriebene ADD ist eine Entwurfsmethode für Software-Architekturen, in der der Entwurfsprozess auf Basis fachlicher und qualitativer Anforderungen durchgeführt wird. Dabei verfolgt die Methode einen rekursiven Prozess, bei dem die Architektur in jedem Durchlauf in kleinere Untermodule zerlegt wird und diese mit abgeleiteten Anforderungen versehen werden. In Abb. 4-4 werden die Schritte der ADD skizziert.

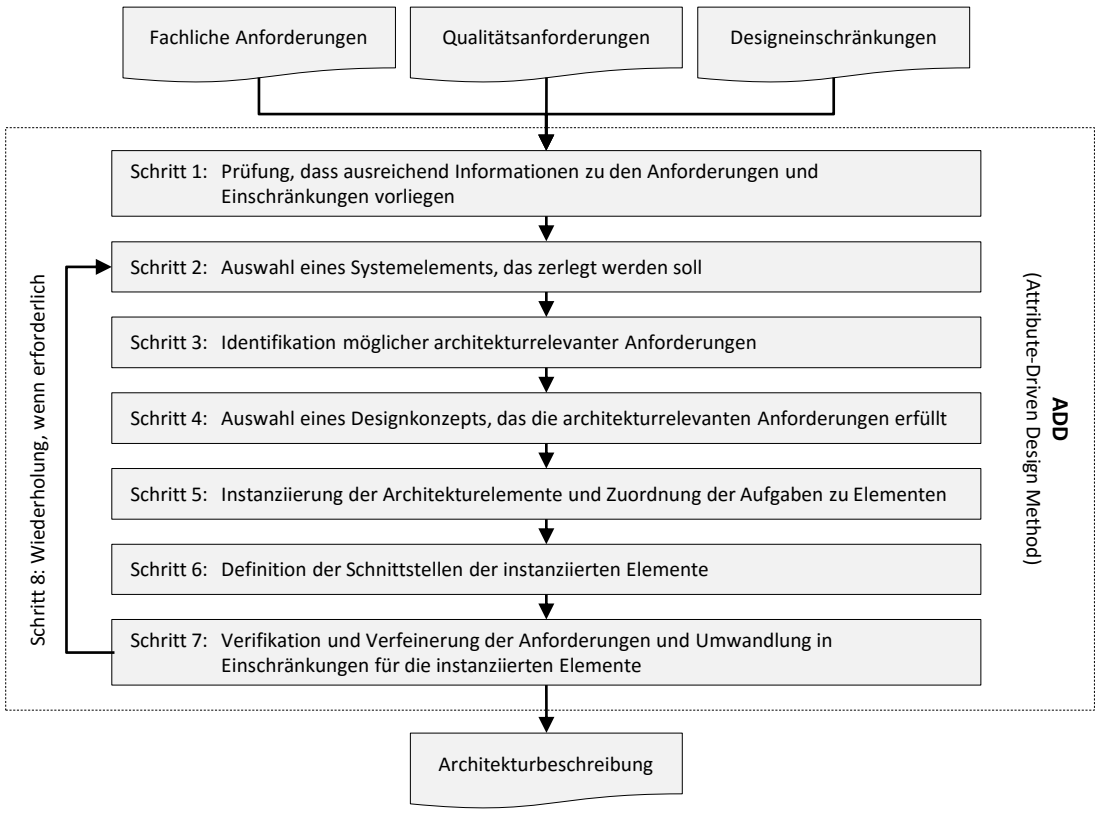

Abb. 4-4: Schritte der Attribute Driven Design Methode nach [WBB+06]

Eingangsartefakte sind Beschreibungen fachlicher Anforderungen, d.h. Aussagen darüber was das System leisten soll, qualitativer Anforderungen, d.h. Aussagen darüber, zu welchem

Grad bestimmte Eigenschaften des Systems erfüllt sein sollen, sowie Entwurfseinschränkungen, die bereits zu einem frühen Entwurfszeitpunkt bestimmte Entscheidungen für den Entwurf und die Implementierung vorweg nehmen (z.B. Festlegung auf eine bestimmte Implementierungstechnologie). Qualitative Anforderungen können von der späteren Softwarearchitektur besser oder schlechter erfüllt werden, Entwurfseinschränkungen müssen vollständig erfüllt werden. Mit letzteren lassen sich also auch K.o.-Kriterien für Architekturalternativen (z.B. bestimmte Verteilungen von Komponenten) beschreiben.

In mehreren iterativen Entwurfsschritten wird das Softwaresystem ausgehend vom Gesamtsystem nach und nach verfeinert bis ein Detailgrad bei der Architekturbeschreibung erreicht ist, der für weitere Bearbeitungsschritte (z.B. Implementierung) ausreichend ist. Ergebnis ist eine Architekturbeschreibung des zu implementierenden Systems. Die Methode selbst trifft keine Aussage darüber, wie die Beschreibung konkret beschaffen ist. Für die weiteren Ausführungen in dieser Arbeit wird unterstellt, dass eine Variante des in Abschnitt 4.1 genannten Beschreibungsmodells verwendet werden kann. Es wird in den folgenden Abschnitten untersucht, inwieweit das Ausgangsmodell in [Mie10] erweitert werden muss, um sich in der ADD bzw. den weiteren Entwurfs- und Bewertungsmethoden als Beschreibungsstandard zu eignen.

Die ADD selbst hat das Ziel, das Modell für genau eine eindeutige Softwarearchitektur in Form einer Architekturbeschreibung als Ergebnis zu liefern. Es ist nicht vorgesehen, dass Architekturalternativen mit entsprechenden Bewertungen behandelt und dokumentiert werden. Dies ist jedoch eine zentrale Anforderung an den in dieser Arbeit zu entwickelnden Gesamtprozess zum Entwurf eines nutzenmaximierenden Architekturmodells. Im weiteren Verlauf dieser Arbeit wird deshalb die ADD dahingehend erweitert, auch Alternativen im Entwurf zu dokumentieren, die in Folgeschritten dann bewertet werden können.

4.2.3.2 Die Architecture Tradeoff Analysis Method (ATAM)

Mit der Architecture Tradeoff Analysis Method (ATAM) gibt es eine Methode, die im Anschluss an die ADD durchgeführt werden kann. Sie ergänzt die ADD um Möglichkeiten zur Bewertung von Alternativen hinsichtlich fachlicher Anforderungen, Qualitätsattributen, Kosten etc. und deren Dokumentation.

Mit einem Softwaresystem sollen bestimmte funktionale und qualitative Ziele erreicht werden. Mit Hilfe der unter anderem in [BCK03], [KKC00] und [KKB+98] beschriebenen ATAM kann bereits zum Entwurfszeitpunkt auf Basis eines Architekturmodells bestimmt werden, ob diese Ziele erreichbar sind und wie sich diese Ziele in Relation zueinander verhalten. Im Falle von Zielkonflikten kann sie dabei helfen, diese aufzulösen, bzw. kann sie Stakeholder bei der Priorisierung einzelner Ziele unterstützen. Abb. 4-5 zeigt die einzelnen Ausführungsschritte der ATAM.

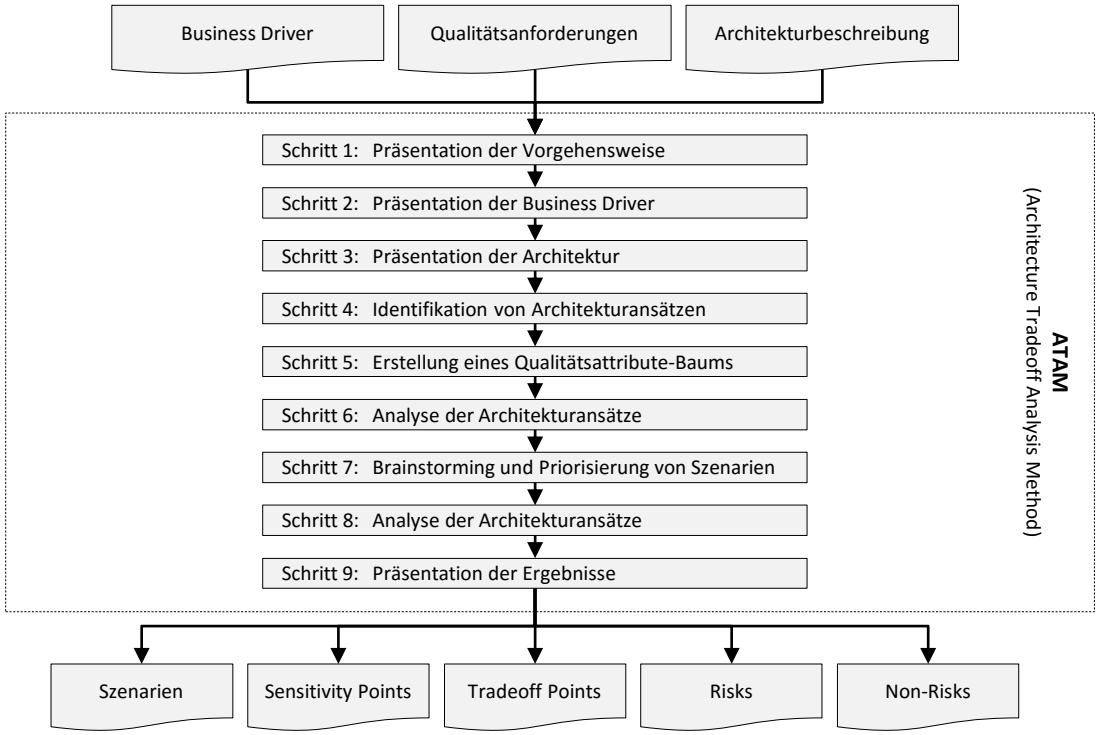

Abb. 4-5: Schritte der Architecture Tradeoff Analysis Method nach [KKB+98]

Die ATAM erhält als Eingangsartefakte eine Beschreibung der Business Driver, über die der Kontext des Anwendungssystems (Stakeholder, politischer Rahmen etc.) beschrieben wird, qualitative Anforderungen, die das System bestmöglich erfüllen muss, und eine Beschreibung der Architektur selbst. Nach Durchführung der Analyseschritte liegen folgende Ergebnisartefakte vor

- Business Goals, die den Nutzen des Systems beschreiben.
- Aus qualitativen Anforderungen abgeleitete Szenarien.
- Architekturansätze, die bestimmte Qualitätsattribute beeinflussen.
- Sensitivity Points, d.h. Architekturentscheidungen, die jeweils Einfluss auf ein Qualitätsattribut haben.
- Tradeoff Points, d.h. Architekturentscheidungen, die Sensitivity Points für mehrere Qualitätsattribute sind und dabei gegenläufige Auswirkungen auf den Erfüllungsgrad dieser Attribute haben.
- Risiken, d.h. Architekturentscheidungen, die ein Qualitätsattribut ungünstig beeinflussen und Nicht-Risiken.
- Risk Themes, die systeminheränte Risiken hinsichtlich der Erfüllung von Anforderungen und damit hinsichtlich der Business Goals beschreiben.

Die ATAM kann gute Dienste bei der Bewertung eines Architekturentwurfs hinsichtlich der Erfüllung gewünschter Qualitätsattribute sowie bei der anschließenden Risikobewertung liefern. Die Methode bleibt allerdings recht unpräzise in Bezug auf eine formale Beschreibung der betrachteten Artefakte (Architekturbeschreibung, Qualitätsattribute etc.). Zwar ist eine wiederholte Anwendbarkeit der Methode durch die Beschreibung des Bewertungsprozesses möglich, jedoch beschränkt sich die Anwendung auf das Befolgen von Handlungsanweisungen. Im Rahmen dieser Arbeit soll die Methode deshalb um eine formale Beschreibung der Methodenartefakte erweitert werden. Diese soll zum einen auf das in Kapitel 4.1 vorgestellte Architektur- und Variabilitätsmodell zur Beschreibung der Softwarearchitektur aufsetzen und zum anderen einen Artefakteaustausch mit den folgenden Entwurfs- und Bewertungsmethoden ermöglichen.

4.2.3.3 Die Cost Benefit Analysis Method (CBAM)

Darüber hinaus soll in der in dieser Arbeit entwickelten Methode neben der Qualitätsattributbewertung auch eine ökonomische Beurteilung von Design-Entscheidungen einfließen. Neben der ATAM wird deshalb eine weitere bestehende Methode benötigt: die Cost Benefit Analysis Method (CBAM).

Wie ATAM wurde die in [KAK02] beschriebene CBAM am Software Engineering Institute der Carnegie Mellon University entwickelt. Während die ATAM einen Softwareentwurf auf Basis gewünschter Qualitätsattribute beurteilt, stellt die CBAM prognostizierte Kosten, Nutzen, den Zeitplan und Risiken in den Fokus der Betrachtung. Sie stellt damit zum einen den Bezug zum Nutzen eines Softwaresystems her, der sich im Grad der Erfüllung verschiedener Qualitätsattribute äußert, und zum anderen zu den Kosten, die bei der Umsetzung einer gewählten Architekturalternative anfallen. Dies ermöglicht eine ganzheitliche Bewertung eines Architekturmodells mit dem Ziel, Stakeholder bei der Auswahl der nutzenmaximierenden Alternative zu unterstützen. Abb. 4-6 skizziert die Bearbeitungsschritte der CBAM.

In [NBC+03] wird bereits eine Integration von ATAM und CBAM beschrieben, bei der einzelne Ausgabeartefakte aus ATAM unmittelbar als Eingangsartefakte für CBAM verwendet werden können. Business Goals, die vorläufigen Szenarien und die Architekturbeschreibung werden während der ATAM hergeleitet bzw. bestehen bereits davor. Die in dieser Arbeit zu erstellende Formalisierung der Artefakte muss eine solche Integration ebenfalls unterstützen.

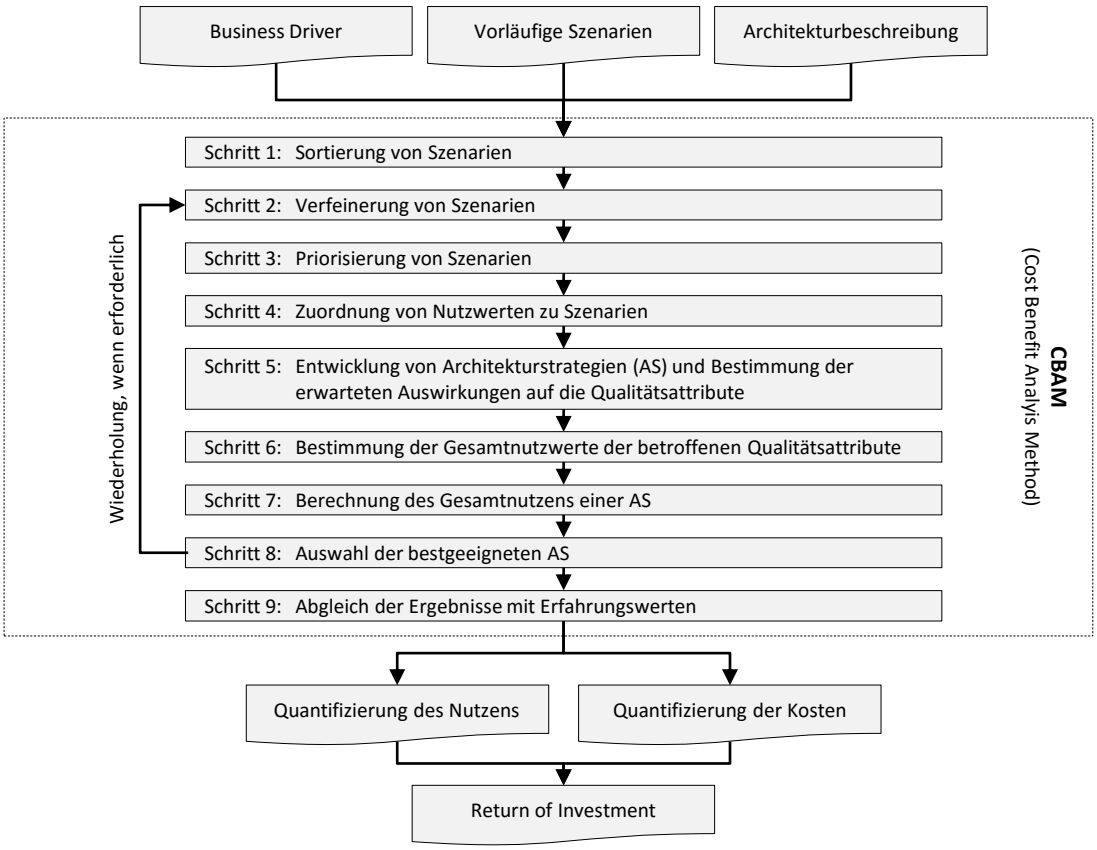

Abb. 4-6: Schritte der Cost Benefit Analysis Method (CBAM) nach [KAK02]

Ergebnis der CBAM sind folgende Artefakte:

- Architekturstrategien, die Alternativen für bestimmte Architekturentscheidungen beschreiben.
- Priorisierung der Strategien, die die Bedeutung der Strategien für einzelne Stakeholder erfasst.
- Return-of-Investment, der eine monetäre Gegenüberstellung von Aufwänden und Nutzen für die Alternativen beschreibt.
- Quantifizierung der Risiken, über die beurteilt wird, mit welchem Risiko der prognostizierte Nutzen verbunden ist.

4.2.4 Verkettung der Vorgehensmodelle

Die drei vorgestellten Vorgehensmodelle lassen sich zu einem durchgängigen Gesamtmodell verketten, in dem die Ausgabeartefakte eines Vorgehensmodells direkt als Eingabeartefakte des nächsten Vorgehensmodells genutzt werden können. Dieses wird in Abb. 4-7 skizziert.

Eine solche Verkettung wird bereits in Teilen in [NBC+03], [BCK03] und [KNK03] beschrieben, ohne dass jedoch durchgängig verwendbare formale Beschreibungen der Artefakte entwickelt werden. Diese Lücke soll mit dieser Arbeit geschlossen werden. Damit werden auch Voraussetzungen dafür geschaffen, den Gesamtprozess durch entsprechende Entwurfswerkzeuge zu unterstützen. Abb. 4-7 skizziert, wie die genannten Entwurfs- und Bewertungsmethoden verkettet durchgeführt werden können.

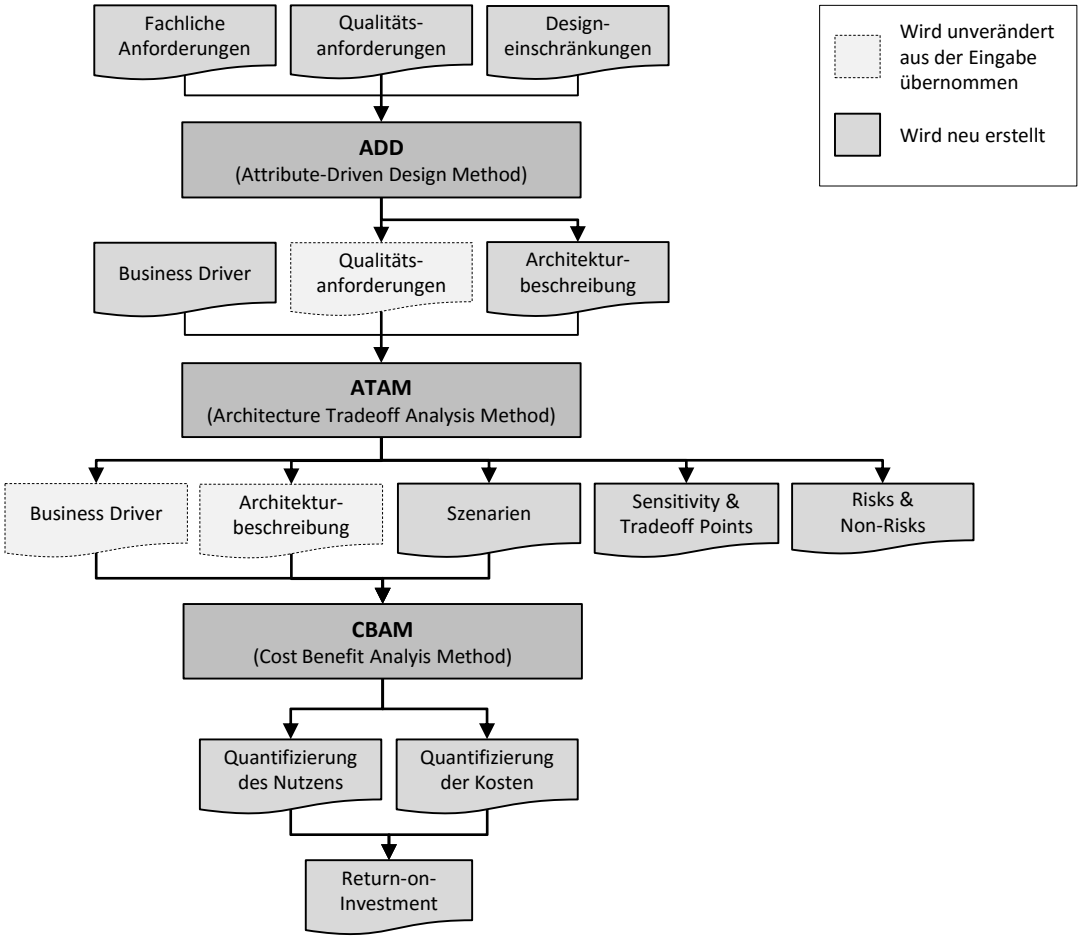

Abb. 4-7: Durchgehender Prozess zum Entwurf einer Architektur

Über das in Abschnitt 4.1.2 genannte Framework können Artefakte für die Übergabe zwischen den Methoden beschrieben werden. Der Architekturentwurf kann über Variabilitätspunkte Alternativen enthalten. ATAM und CBAM müssen deshalb um die Möglichkeit erweitert werden, auf alle Alternativen angewendet zu werden und damit entsprechend Bewertungen aller Architekturvarianten vorzunehmen. Diese können dann am Ende Grundlage für eine Entscheidung für die „bestgeeignete“ Alternative sein.

4.3 Forschungsbedarf

Sowohl das Beschreibungsmodell als auch die Entwurfs- und Bewertungsmethoden müssen also für den integrierten durch technische Werkzeuge unterstützbaren Gesamtprozess erweitert werden. Hierzu wird zum einen das Beschreibungsmodell so erweitert, dass es als Mittel zur Dokumentation von Entwurfs- und Bewertungsergebnissen genutzt werden kann. Zum anderen werden die Methoden so angepasst, dass sie nacheinander durchgeführt werden können und Variabilitäten im Architekturmodell unterstützen.

4.3.1 Erweiterungen bei der Modellierung der Software-Architektur

Das in [Mie10] beschriebene Verfahren zur Erstellung eines Architektur- und eines Variabilitätsmodells hat zum Ergebnis eine Anwendungsvorlage (Application Template), welches Basis für das Deployment einer bestimmten Anwendung in unterschiedlichen Konfigurationen und Ausführungsumgebungen sein kann. Dabei wird nur knapp auf die Frage eingegangen, nach welchen Bearbeitungsschritten die Anwendungsvorlage aus den Anforderungen abgeleitet und wie eine Bewertung von Alternativen für die Variabilitätspunkte formal durchgeführt werden kann.

Diese Lücken soll die vorliegende Arbeit schließen. Die hergeleitete Entwurfs-, Modellierungs- und Bewertungsmethode soll folgenden Anforderungen genügen:

> *Anforderung 1 (Formale Beschreibung der Anforderungen).* Funktionale und qualitative Anforderungen sowie Constraints können formal modelliert werden.
>
> *Anforderung 2 (Formale Beschreibung der Anforderungen einer Komponente).* Zwischen den Anforderungen an ein zu entwickelndes Softwaresystem und den Komponenten des Systems sollten Beziehungen modelliert werden können, so dass nachvollziehbar ist, welche Komponenten welche Anforderungen abdecken.
>
> *Anforderung 3 (Formale Beschreibung von Entwurfsalternativen).* Aus dem modellierten Anwendungsentwurf können für Entwurfsalternativen Bewertungen hinsichtlich der zugrundeliegenden Anforderungen durchgeführt und deren Ergebnisse gegenübergestellt werden.

4.3.2 Erweiterung bestehender Entwurfs- und Bewertungsmethoden

Die oben beschriebenen, bereits bestehenden Entwurfs- und Bewertungsmethoden haben ebenfalls Beschränkungen, die mit dieser Arbeit adressiert werden sollen. ADD, ATAM und CBAM fokussieren auf die Herleitung einer eindeutigen Softwarearchitektur und deren Bewertung hinsichtlich bestehender Anforderungen. Variabilitäten werden unmittelbar durch

Auswahl einer Alternative beseitigt. Zulieferungen durch Service Provider werden nur indirekt unterstützt.

Hinsichtlich des Entwurfs- und Bewertungsprozesses sollen im Rahmen dieser Arbeit Erweiterungen vorgenommen werden, die folgende Anforderungen erfüllen:

> *Anforderung 4 (Unterstützung hybrider Cloud-Architekturen).* Bei der Herleitung des Architekturentwurfs sollen explizit Szenarien unterstützt werden, bei denen Teile des Systems als Service eines Cloud Anbieters verwendet werden. Für diese Teile ist eine weitere Untergliederung nicht mehr erforderlich.

> *Anforderung 5 (Bewertbarkeit von Entwurfsalternativen).* Variabilitäten sollen so lange wie möglich im Architekturentwurf verbleiben, um auch auf Ebene der Gesamtarchitektur noch Auswirkungen des Einsatzes von Cloud Services auf die Anforderungen betrachten zu können.

> *Anforderung 6 (Vermeidung unnötigen Overheads).* Der Entwurfs- und Bewertungsprozess soll so einfach wie möglich gehalten werden, um diesen auch bei kleineren Projekten sinnvoll und ohne unnötigen Overhead einsetzen zu können.

4.3.3 Integration der Ansätze zu einem durchgehenden Leitfaden

Ergebnis dieser Arbeit ist ein integrierter Prozess zur Erstellung und Bewertung eines Softwarearchitekturentwurfs bei gegebenen Anforderungen. Ausgangspunkt sind formal beschriebene Anforderungen an das Softwaresystem. Entsprechend der ADD wird hieraus beginnend bei einem generischen Architekturartefakt dieses schrittweise verfeinert. Als Ergebnis steht dann eine formal beschriebene Architekturvorlage, bestehend aus Architektur- und Variabilitätsmodell, sowie eine mit dieser verknüpften Beschreibung der Anforderungen. Diese werden dann an ATAM und CBAM übergeben, um eine detaillierte Bewertung von Architekturalternativen hinsichtlich der gegebenen Anforderungen vorzunehmen.

Die einzelnen Prozessschritte werden hierbei formalisiert, d.h. die Transformationen von formal beschriebenen Eingangsartefakten in die jeweiligen Ausgangsartefakte werden detailliert beschrieben. Für den Gesamtprozess gilt dabei folgende Anforderung:

> *Anforderung 7 (Ein- und Ausgaben des Gesamtprozesses).* Aus gegebenen Anforderungen an ein Softwaresystem sollen ein Architekturentwurf sowie eine Bewertung darin enthaltener Alternativen nach Risiken und dem erwarteten Return-on-Investment abgeleitet werden.

4.3.4 Integration von Architekturmustern

Im Rahmen der Evaluierung des integrierten Prozesses sollen Best Practices, die im Kontext von Cloud Computing bestehen und die bereits Hinweise auf abstrakte Variabilitätsmodelle geben, nach Möglichkeit berücksichtigt werden. So kann im Bereich der Cloud Storage Services Windows Azure Table Storage recht einfach durch Amazon Simple Storage Service (S3) ersetzt werden. Eine solche Ersetzbarkeit soll in einem abstrakten Variabilitätsmodell formuliert und dann in konkreten Anwendungsszenarien verwendet werden können.

> *Anforderung 8 (Modellierbarkeit von Architekturmustern).* Das verwendete Beschreibungsmodell ermöglicht die Modellierung von Architekturmustern, so dass diese während der Architekturmodellierung verwendet werden können.

4.4 Scope

Für die oben beschriebenen Entwurfsmethodiken wurden von den jeweiligen Autoren bereits viele Details zur Organisation einzelner Bearbeitungsschritten beschrieben. So gibt es Vorschläge für die Zusammensetzung von Entwurfsteams, der Ausgestaltung von Meetings und zugehöriger Mitschriften. Sofern diese Details nicht für die Modellierung und formalen Bewertung des Architekturentwurfs notwendig sind, sollen sie für den hier beschriebenen integrierten Prozess ausgeklammert werden. Der Fokus soll auf der technischen und nicht der organisatorischen Umsetzung liegen.

5 Erweiterung einzelner bestehender Beschreibungs- und Vorgehensmethoden

Die bestehenden Beschreibungs- und Vorgehensmodelle und die in deren Verlauf beschriebenen Ein- und Ausgabeartefakte sind in [BCK03] bzw. detaillierter in [WBB+06], [KAK02] und [KKC00] definiert bzw. beschrieben. Die Modelle sind allerdings noch nicht ausreichend formalisiert, um Ein- und Ausgabeartefakte einzelner Bearbeitungsschritte einer maschinellen Verarbeitung zugänglich zu machen. Außerdem wird durch dieses Defizit die Integration der Einzelmethoden zu einem Gesamtmodell erschwert. In den folgenden Abschnitten sollen deshalb für die (informellen) Definitionen der Ein- und Ausgabeartefakte formale Notationen eingeführt werden und für die Entwurfs- und Bewertungsprozesse entsprechend formale Algorithmen erstellt werden. Diese orientieren sich an der Syntax der in [Mie10] beschriebenen Beschreibungsmethodik für Architekturmodelle. In späteren Abschnitten wird aus den Einzelmethoden ein Gesamtmodell abgeleitet.

5.1 Erweiterung der Attribute Driven Design Method

Die Attribute Driven Design Method leitet schrittweise, ausgehend vom Gesamtsystem, durch iterative Unterteilung in Subsysteme und Zuweisung entsprechend abgeleiteter Anforderungen an diese Subsysteme die Beschreibung einer Softwarearchitektur ab. Zentrale Handlungsgrundlage sind Anforderungen an bestimmte Qualitätsattribute. Diese können beispielsweise über einen Qualitätsattributworkshop, wie in [BEL+03] beschreiben, erarbeitet werden.

Der folgende Abschnitt betrachtet die einzelnen Arbeitsschritte, die in [WBB+06] beschrieben sind, und formuliert Erweiterungen für die Methode, die dann in einem späteren Abschnitt für die Integration von ADD in die Gesamtmethode benötigt werden. Dabei wird jeweils beschrieben, welche Vorgaben bereits durch ADD beschrieben werden und wo diese ergänzt werden.

5.1.1 Formalisierung der Eingabe

Die ADD erhält die drei folgenden Eingabeartefakte:

- *Fachliche Anforderungen*
  Diese beschreiben, welche Funktionalitäten vom System bzw. einzelnen Systemkomponenten erbracht werden sollen.

Beispiel: „Anwender können sich im Bestellsystem den Inhalt des Warenkorbs anzeigen lassen“.

- *Qualitative Anforderungen*
  Hierüber wird beschrieben, mit welcher messbaren Ausprägung (d.h. „wie gut“) bestimmte Eigenschaften erfüllt sein sollten.
  Beispiel: „Der Zugriff auf die Bestelldatenbank erfolgt mit einer Antwortzeit von weniger als zehn Millisekunden“.

- *Designeinschränkungen*
  Diese machen für den Architekturentwurf bestimmte Vorgaben, die auf jeden Fall erfüllt sein müssen, d.h. sie stellen Entwurfsentscheidungen dar, bei denen bereits zu Beginn feststeht, welche Alternative ausgewählt wird.
  Beispiel: „Die Benutzerdaten werden in einer lokalen SQL-Server-Datenbank gespeichert (und dürfen nicht in die Public Cloud ausgelagert werden)“.

Für diese Artefakte wird nun eine formale Beschreibung benötigt, um sie im weiteren Verlauf in den Gesamtprozess zu integrieren und einer Werkzeugunterstützung zugänglich zu machen. In Anlehnung an die Ausführungen in [Mie10] sollen die dort definierten Anwendungs- und Variabilitätsmodelle hier um ein Anforderungsmodell ergänzt werden, welches wie folgt definiert sei:

Definition 1 (Anforderungsmodell). Gegeben sei die Menge aller Anforderungsmodelle $RM = \{..., r_i, ...\}$ sowie eine Menge von Business Drivern $bd \in BD$. Ein **Anforderungsmodell** $rm \in RM$ ist ein Tupel

$$rm = (FR, QR, DC, freqPrio, freqBD, qreqPrio, qreqBD, dcBD)$$

wobei für die einzelnen Elemente folgendes gilt:

FR	ist eine Menge von fachlichen Anforderungen,
QR	ist eine Menge von qualitativen Anforderungen,
DC	ist eine Menge von Designeinschränkungen
$freqPrio$	ist eine Abbildung $(fr \rightarrow Priorität)$,
$freqBD$	ist eine Abbildung $(fr \rightarrow bd)$,
$qreqPrio$	ist eine Abbildung $(qr \rightarrow Priorität)$,
$qreqBD$	ist eine Abbildung $(qr \rightarrow bd)$,
$dcBD$	ist eine Abbildung $(dc \rightarrow bd)$.

Die einzelnen Anforderungen leiten sich aus Business Drivern ab. Diese sind wie folgt definiert.

Definition 2 (Business Driver). Ein **Business Driver** ist die in Prosa verfasste Beschreibung eines Ziels, das mit der Erstellung der Anwendungssoftware erreicht werden soll.

Häufig werden (insbesondere bei Softwaresysteme überschaubarer Komplexität) fachliche Anforderungen in einfacher Prosa verfasst. Dieses pragmatische Vorgehen hat, insbesondere dann seine Berechtigung, wenn die Projektbeteiligten einen guten Überblick über die Anforderungen und ein gemeinsames Verständnis zur Interpretation der Anforderungen haben. Für eine Werkzeugunterstützung, die auch eine Filterung von Anforderungen nach Systemkomponenten abbilden soll, ist allerdings eine formale Notation erforderlich. In [BCK03] wird hierzu für Systemanforderungen eine Notation der Form „Stimulus-Reaktion" beschrieben. Auf deren Basis lässt sich eine fachliche Anforderung wie folgt definieren:

Definition 3 (Fachliche Anforderung). Eine **fachliche Anforderung** $fr \in FR$ ist ein Tupel

$$fr = (stsrc, st, env, res, descr)$$

wobei für die einzelnen Elemente folgendes gilt:

$stsrc$	ist die Beschreibung einer Stimulusquelle
st	ist die Beschreibung eines Stimulus,
env	ist die Beschreibung der Umgebung, unter der das System läuft,
res	ist die Beschreibung der Reaktion des Systems,
$descr$	ist eine Beschreibung der fachlichen Anforderung in Prosa.

Qualitative Anforderungen enthalten etwas mehr Informationen, da sie nicht nur beschreiben „was" das System leisten soll, sondern auch „wie gut" diese Leistung erbracht und gemessen werden soll. Qualitative Anforderungen seien in Anlehnung an die Ausführungen in [BCK03] wie folgt formal definiert:

Definition 4 (Qualitative Anforderung). Eine **qualitative Anforderung** $qr \in QR$ ist ein Tupel

$$qr = (stsrc, st, env, res, resmsr, descr)$$

wobei für die einzelnen Elemente folgendes gilt:

$stsrc$	ist die Beschreibung einer Stimulusquelle,
st	ist die Beschreibung eines Stimulus,
env	ist die Beschreibung der Umgebung, unter der das System läuft,
res	ist die Beschreibung der Reaktion des Systems,
$resmsr$	ist die Beschreibung einer Maßeinheit für die Reaktion des Systems,
$descr$	ist eine Beschreibung der qualitativen Anforderung in Prosa.

Designeinschränkungen können vielfältiger Natur sein. Sie können z.B. bestimmte Implementierungsdetails vorschreiben, die Zulieferung durch Cloud Provider vorgeben bzw. auch ausschließen. Sie machen also Vorgaben dafür, welche Komponententypen für Komponenten und welche Implementierungstypen für Implementierungen zur Verfügung stehen. In einem werkzeugunterstützten Entwurfs- und Bewertungsprozess können daraus Checklisten generiert werden, die einen Software-Architekten dabei unterstützen, während des Entwurfs und der Ableitung von Entwurfs- und Implementierungsdetails die Einschränkungen auf deren Erfüllung im Architekturmodell hin zu prüfen. Designeinschränkungen seien also wie folgt formal definiert:

Definition 5 (Designeinschränkung). Eine **Designeinschränkung** $dc \in DC$ ist ein Tupel

$$dc = (typec, implTc, descr)$$

wobei für die einzelnen Elemente folgendes gilt:
$typec$ ist die Abbildung $(Komponente \rightarrow Komponententyp)$,
$implTc$ ist die Abbildung $(Implementierung \rightarrow Implementierungstyp)$,
$descr$ ist eine Beschreibung der Designeinschränkung in Prosa.

Die ADD selbst kennt neben den Anforderungen keine weiteren Eingabeartefakte. Um die Bearbeitungsschritte der ADD allgemeingültig und unabhängig davon zu machen, ob sie die erste Iteration oder nachfolgende Iterationen durchläuft, soll an dieser Stelle noch ein weiteres Eingabeartefakt eingeführt werden: der initiale Architekturentwurf. Dieser ist ein spezieller Architekturentwurf, bei der die einzige Komponente das Gesamtsystem selbst und nur die Variabilitäten des Gesamtsystems (sofern vorhanden) definiert sind.

Definition 6 (initialer Architekturentwurf). Ein **initialer Architekturentwurf** $ad_{init} \in AD$ ist ein Tupel

$$ad_{init} = (am_{init}, vm_{am_{init}}).$$

wobei für die einzelnen Elemente folgendes gilt:
am_{init} ist ein Anwendungsmodell, das nur eine einzige Komponente enthält,
$vm_{am_{init}}$ ist das Variabilitätsmodell des Komponententyps der Komponente aus am_{init}.

Dieser initiale Architekturentwurf dient bei der Durchführung der ADD als weitere Eingabe. In folgenden Iterationen wird aus diesem initialen Entwurf ein finaler Architekturentwurf abgeleitet, d.h. die als einzige enthaltene Komponente wird schrittweise immer weiter untergliedert, bis ein für folgende Entwicklungsschritte – insbesondere die Implementierung – ausreichend hoher Detaillierungsgrad erreicht ist.

5.1.2 Formalisierung der Ausgabe

Das Anforderungsmodell steckt den Rahmen für die Herleitung einer Softwarearchitektur ab. Es beschreibt, was das modellierte Softwaresystem leisten muss, in welcher qualitativen Ausprägung und wo Vorgaben für den Entwurf der Architektur bestehen. Im Rahmen dieser Arbeit soll der Fokus auf der Erstellung und Bewertung einer Softwarearchitektur und nicht auf der Erstellung eines Deployment-Paketes liegen. Für die weiteren Betrachtungen werden zwei der drei in [Mie10] beschriebenen Artefakte einer Anwendungsvorlage (Anwendungsmodell, Variabilitätsmodell und Komponentenbindungen) benötigt: Anwendungs- und Variabilitätsmodell. Nachdem für die spätere Bewertung eines Anwendungsentwurfs entscheidend ist, wie Architekturkomponenten und Alternativen mit Anforderungen in Beziehung stehen, soll an dieser Stelle das Anwendungsmodell zu einem anforderungsbehafteten Anwendungsmodell erweitert werden.

Definition 7 (anforderungsbehaftetes Anwendungsmodell). Gegeben sei ein Anforderungsmodell RM sowie die Menge aller anforderungsbehafteten Anwendungsmodelle $AM_{RM} = \{\dots, a_i, \dots\}$. Ein **anforderungsbehaftetes Anwendungsmodell** $am_{RM} \in AM_{RM}$ ist ein Tupel

$$am = (C, I, F, B, D, type, impl, implT, files, blockOf, p, freqC, qreqC, dcC)$$

$C, I, F, B, D, type, impl, implT, files, blockOf$ und p sind definiert wie im Anwendungsmodell (siehe 4.1). Für die weiteren Elemente gilt:
$freqC$ ist eine Abbildung $(fr \rightarrow Komponente)$,
$qreqC$ ist eine Abbildung $(qr \rightarrow Komponente)$,
dcC ist eine Abbildung $(dc \rightarrow Komponente)$,
wobei fr, qr und dc die entsprechenden Elemente aus RM sind.

Diese bilden zusammen einen Architekturentwurf, der wie folgt formal definiert sei:

Definition 8 (Architekturentwurf). Gegeben sei die Definition eines anforderungsbehafteten Anwendungsmodells $am_{RM} \in AM_{RM}$ und die eines zugehörigen Variabilitätsmodells $vm \in VM$. Die Menge der Architekturentwürfe AD ist definiert als eine Teilmenge des kartesischen Produkts der Menge der anforderungsbehafteten Anwendungsmodelle und der Menge der Variabilitätsmodelle:

$$AD \subseteq AM_{RM} \times VM$$

Ein **Architekturentwurf** $ad \in AD$ ist somit definiert als ein Tupel aus einem Anwendungsmodell $am_{RM} \in AM_{RM}$ und einem zu am_{RM} passenden Variabilitätsmodell $vm_{am} \in VM$

(am_{RM}, vm_{am}).

Ein Architekturentwurf $ad \in AD$ ist nur dann gültig, wenn das Variabilitätsmodell zum Anwendungsmodell passt.

Für spätere Prozessschritte ist es erforderlich, nicht nur die Menge aller fachlichen Anforderungen zu kennen, sondern auch diejenigen Anforderungen identifizieren zu können, die direkten Bezug zu einer bestimmten Systemkomponente besitzen. Entsprechend lässt sich die Menge aller fachlichen Anforderungen einer Komponente wie folgt definieren:

Definition 9 (Fachliche Anforderungen einer Komponente). Die Menge der **fachlichen Anforderungen FR_C einer Komponente** $c \in C$ eines Anwendungsmodells sei definiert als

$$FR_C = \{fr \in FR \mid freqC(fr) = c\}.$$

Analog zu den fachlichen Anforderungen ist auch für qualitative Anforderungen von Interesse, welche jeweils einzelnen Architekturkomponenten zugewiesen sind. Entsprechend lässt sich die Menge aller qualitativen Anforderungen einer Komponente wie folgt definieren:

Definition 10 (Qualitative Anforderungen einer Komponente). Die Menge der **qualitativen Anforderungen QR_C einer Komponente** $c \in C$ eines Anwendungsmodells sei definiert als

$$QR_C = \{qr \in QR \mid qreqC(qr) = c\}.$$

Auch hier ist die Menge aller Designeinschränkungen einer Komponente von Interesse. Diese lässt sich wie folgt definieren:

Definition 11 (Designeinschränkungen einer Komponente). Die **Designeinschränkungen dc_C einer Komponente** $c \in C$ eines Anwendungsmodells sei definiert als

$$DC_C = \{dc \in DC \mid dcC(dc) = c\}.$$

Ausgehend vom Anforderungsmodell kann mit Hilfe der ADD ein Architekturentwurf erstellt werden. Die hierzu erforderlichen Schritte werden informell in [WBB+06] und formell in folgenden Abschnitten beschrieben.

5.1.3 Erweiterung der Bearbeitungsschritte

Um die informell beschriebene Methodik der ADD auf ein Anforderungsmodell anwenden zu können, um daraus einen Architekturentwurf zu erstellen, muss diese um formelle Bearbeitungs- und Ableitungsregeln ergänzt werden, die im Folgenden für jeden Arbeitsschritt der ADD beschrieben werden sollen.

Die Bearbeitungsschritte der ADD überführen dabei in jeder Iteration den Anwendungsentwurf in einen etwas detaillierteren Anwendungsentwurf. Um den Ein- und Ausgabeentwurf der i-ten Iteration kennzeichnen zu können wird folgende festgelegt:

> ad_i^{in} ist der Entwurf, der in der i-ten Iteration als Eingabe übergeben wird,
> ad_i^{out} ist der Ergebnisentwurf der i-ten Iteration, wobei folgendes gilt:
>
> $ad_1^{in} = ad_{init}$, d.h. Eingabe der ersten Iteration ist der initiale Architekturentwurf, und $ad_{i+1}^{in} = ad_i^{out}$, d.h. der Ausgabeentwurf der i-ten Iteration ist Eingabeentwurf der $(i+1)$-ten Iteration (für $i > 0$).

5.1.3.1 Schritt 1: Prüfung, dass ausreichend Informationen zu den Anforderungen und Einschränkungen vorliegen

Der erste Schritt der ADD besteht aus einer Validierung der Eingabe. Wie oben beschrieben, ist die Eingabe in die ADD ein Anforderungsmodell sowie ein initialer Architekturentwurf, aus dem in mehreren Iterationen immer detailliertere Architekturentwürfe abgeleitet werden. In jeder Iteration gelten für Anforderungsmodell und Architekturentwurf folgende Regeln, die bestimmen, ob die Anforderungen ausreichend und vollständig beschrieben sind:

> *Regel 1* (Jede Komponente muss mindestens eine Anforderung haben). Es gibt keine Komponente, die nicht durch mindestens eine fachliche oder qualitative Anforderung beschrieben ist, d.h.
>
> $\nexists\, c \in C : fr_c = \emptyset \wedge qr_c = \emptyset.$
>
> *Regel 2* (Jede Anforderung ist vollständig beschrieben), d.h. kein Element des Tupels zur Beschreibung einer Anforderung ist leer.

Entwurfseinschränkungen sind in diesem ersten Schritt noch nicht relevant. Sofern vorhanden, muss deren Einhaltung in den folgenden Schritten sichergestellt werden.

5.1.3.2 Schritt 2: Auswahl eines Systemelements, das zerlegt werden soll

In der ersten Iteration der ADD ist dieser Schritt trivial, da das einzige Element des bis zu diesem Zeitpunkt entwickelten Architekturentwurfs eine einzige Komponente ist: das Gesamtsystem, welches über den initialen Architekturentwurf beschrieben wird. Ab der zweiten Iteration muss eine Komponente ausgewählt werden, die in dieser Iteration weiter ausgearbeitet wird. Die ADD macht hier keine Vorgaben, nach welchem Verfahren die Komponente ausgewählt werden soll. Für den in dieser Arbeit entwickelten Prozess soll nur aus solchen Komponenten ausgewählt werden können, für die noch keine Implementierung definiert wurde.

> *Regel 3* (Menge der für eine weitere Zerlegung auswählbaren Komponenten). Die Menge auswählbarer Komponenten C_{aus} ist definiert als
>
> $$C_{aus} = \{c \in C \mid \nexists i \in I : impl(c) = i\}$$
>
> Aus dieser Menge kann ein Element ausgewählt werden.

Für die tatsächliche Auswahl ist unerheblich, ob für diese oder eine andere der auswählbaren Komponenten eine Variabilität definiert ist. Es liegt am verantwortlichen Software-Architekten, ob er für die Auswahl grundsätzlich eher eine Depth-first- oder eine Breadth-first-Strategie verfolgt. Die ADD macht hier keine Vorgaben, ob zunächst eine Komponente immer weiter verfeinert (depth-first) oder die Architektur als Ganzes gleichmäßig ausgearbeitet werden soll (breadth-first).

5.1.3.3 Schritt 3: Identifikation möglicher architekturrelevanter Anforderungen

Aus der Vereinigungsmenge der fachlichen und qualitativen Anforderungen für die ausgewählte Komponente werden fünf bis sechs Anforderungen mit höchster Priorität sowie alle Designeinschränkungen ausgewählt. Die Menge der in diesem Sinne architekturrelevanten Anforderungen für die ausgewählte Komponente lässt sich also wie folgt definieren:

> *Definition 12* (Architekturrelevante Anforderungen einer Komponente). Die Menge der **architekturrelevanten Anforderungen einer Komponente** setzt sich zusammen aus den wichtigsten fachlichen Anforderungen, den wichtigsten qualitativen Anforderungen sowie allen Designeinschränkungen dieser Komponente.
>
> $$REQ_{arch,c} = \left(FR_{arch,c}, QR_{arch,c}, DC_c\right)$$
>
> Es gibt somit keine Anforderung an die Komponente, die wichtiger ist als die als architekturrelevant eingestuften Anforderungen. Es gilt also:

$$FR_{arch,c} = \{fr \in FR_c \mid \nexists fr_i \in FR_c \setminus FR_{arch,c} : Prio(fr_i) > Prio(fr)\},$$
$$QR_{arch,c} = \{qr \in QR_c \mid \nexists qr_i \in QR_c \setminus QR_{arch,c} : Prio(qr_i) > Prio(qr)\}$$

Da grundsätzlich alle Designeinschränkungen der Komponente erfüllt sein müssen, werden alle Einschränkungen für die Komponente betrachtet. Eine Filterung nach Priorität entfällt.

Die architekturrelevanten Anforderungen einer Komponente sind somit eine Untermenge des Anforderungsmodells des Gesamtsystems und bilden ihrerseits ein Anforderungsmodell.

5.1.3.4 Schritt 4: Auswahl eines Designkonzepts, das die architekturrelevanten Anforderungen erfüllt

Dieser Arbeitsschritt ist einer Formalisierung nur sehr eingeschränkt zugänglich. Hier ist die Expertise eines Software-Architekten gefordert, der sicherstellen muss, dass bei einer weiteren Ausarbeitung einer Komponente $c \in C_{aus}$ alle Anforderungen aus $REQ_{arch,c}$ erfüllt werden. Das ausgewählte Designkonzept bestimmt, in welche Architekturelemente die für die Zerlegung ausgewählte Komponente aufgeteilt wird und wo bereits die Implementierung von Teilkomponenten beschrieben werden kann. Kann kein Designkonzept gefunden werden, das die relevanten Anforderungen erfüllen kann, muss die ADD an dieser Stelle abgebrochen werden. Es kann in diesem Fall kein Entwurf erstellt werden, der die Anforderungen erfüllt. Möglicherweise können die Anforderungen nachgebessert bzw. aufgeweicht werden, so dass ein erneuter Durchlauf der ADD erfolgversprechender ist. Im Umkehrschluss bedeutet dies, dass – sofern hier kein Abbruch der ADD initiiert werden muss – das ein vollständiger Durchlauf der ADD zu einem Architekturentwurf führt, der grundsätzlich alle fachlichen Anforderungen und alle Designeinschränkungen erfüllt. Bei den qualitativen Anforderungen muss in nachgelagerten Phasen untersucht werden, zu welchem Grad sie erfüllt werden.

5.1.3.5 Schritt 5: Instanziierung der Architekturelemente und Zuordnung der Aufgaben zu Elementen

Mit der Auswahl des Designkonzepts erfolgt nun eine Detaillierung der Architekturkomponente c , welche in Schritt 2 ausgewählt wurde. Dabei wird die Komponente in eine oder mehrere Sub-Komponenten zerlegt. Für diese wiederum können Implementierungen mit und ohne Variabilität beschrieben werden. In diesem Schritt wird nun jede Anforderung aus $REQ_{arch,c}$ auch allen Sub-Komponenten zugewiesen für die diese Anforderung relevant ist. Die Anforderungen werden dabei zunächst unverändert übernommen. Auf die Beschreibungsmodelle hat dieser Entwurfsschritt somit folgende Auswirkungen:

- Das Anwendungsmodell wird um die neuen Komponenten (die Sub-Komponenten der für die Verfeinerung ausgewählten Komponente c) sowie die zugehörigen Modellelemente (Komponententyp, Implementierung, … und die entsprechenden Abbildungen) ergänzt.

- Das Variabilitätsmodell wird ggf. entsprechend ebenfalls um die zugehörigen Elemente zu den Sub-Komponenten ergänzt.

- Das Anforderungsmodell erhält neue Abbildungen, die die Zugehörigkeit von Anforderungen zu den neuen Komponenten beschreiben. Diese Abbildungen sind temporär. In Schritt 7 werden diese durch abgeleitete Anforderungen ersetzt, die auf die einzelnen Sub-Komponenten zugeschnitten sind.

Die Auswirkungen auf die Modelle sollen nun formal beschrieben werden. Zunächst wird mit der Verfeinerung einer Komponente eine Möglichkeit geschaffen, neu hinzugekommene Komponenten im Architekturentwurf zu identifizieren.

> *Definition 13* (Verfeinerung einer Komponente). Die Menge $CR = \{\dots, c_i, \dots\}$ mit $i \geq 1$ bezeichnet eine endliche Menge von Komponenten.
> Die Abbildung $Crefine : CR \rightarrow C$ beschreibt die **Verfeinerung von Komponenten**, d.h. die Menge der Sub-Komponenten $C_{sub,c}$ einer Komponente $c \in C$ eines Anwendungsmodells lässt sich demzufolge wie folgt notieren.
>
> $$C_{sub,c} = \{c_{sub} \in C : Crefine(c_{sub}) = c\}$$

Der Architekturentwurf erhält mit der Verfeinerung also zusätzliche Komponenten. Zu den Komponenten, die bereits im Eingabeentwurf ad^{in} enthalten sind, kommen die Komponenten hinzu, die die Verfeinerung von c darstellen, d.h.

$C^{out} = C^{in} \cup C_{sub,c}$, wobei $c \in C^{in}$.

Die Anforderungen an die Komponente c werden nun zunächst unverändert an die Komponenten aus $C_{sub,c}$ übertragen. Die Zuordnungen von Anforderungen an Sub-Komponenten erfolgt dabei dergestalt, dass nur solche Anforderungen an Komponenten zugeordnet werden, die für die betreffende Sub-Komponente grundsätzlich relevant ist. Hierzu wird für die Komponenten aus $C_{sub,c}$ ein vorläufiges Anforderungsmodell $rm_{sub,c}$ erstellt, dessen Elemente wie folgt belegt werden.

$FR_{sub,c}$	enthält für alle $c_i \in C_{sub,c}$ alle $fr \in FR_{arch,c}$,
$QR_{sub,c}$	enthält für alle $c_i \in C_{sub,c}$ alle $qr \in QR_{arch,c}$,
$DC_{sub.c}$	enthält für alle $c_i \in C_{sub,c}$ alle $dc \in DC_c$,
$freqC_{sub,c}$	weist die fachlichen Anforderungen den neuen Komponenten zu,

$freqPrio_{sub,c}$	übernimmt die Priorisierungen der fachlichen Anforderungen unverändert,
$qreqC_{sub,c}$	weist die qualitativen Anforderungen den neuen Komponenten zu,
$qreqPrio_{sub,c}$	übernimmt die Priorisierungen der qualitativen Anforderungen,
$dcC_{sub,c}$	weist die Designeinschränkungen den neuen Komponenten zu.

In Schritt 7 wird dieses vorläufige Anforderungsmodell verfeinert und in das Anforderungsmodell des Architekturentwurfs aufgenommen.

5.1.3.6 Schritt 6: Definition der Schnittstellen der instanziierten Elemente

Der Architekturentwurf wird in einer ADD-Iteration nicht nur um neue Komponenten erweitert. Die neuen Komponenten ziehen auch entsprechende Erweiterungen bei den anderen Elementen des Architekturentwurfs, d.h. dem Anwendungs- und dem Variabilitätsmodell nach sich. Für jede hinzugefügte Komponente $c_i \in C_{sub,c}$ müssen die Elemente des Anwendungsentwurfs wie Implementierungen, Dateien, Building Blocks etc. in das Modell aufgenommen werden, sofern dies durch das in Schritt 4 ausgewählte Designkonzept erforderlich ist. Gleiches gilt für das Variabilitätsmodell: Sofern für die verfeinerte Komponente zu bewertende Implementierungsalternativen ins Modell aufgenommen werden sollen, muss das Variabilitätsmodell entsprechend ergänzt werden. Diese Erweiterungen hängen stark vom gewählten Designkonzept bzw. von der Festlegung ab, bestimmte Architekturentscheidungen noch nicht zu treffen sondern als Variabilitäten zu modellieren. Die Erweiterungen der Modelle werden deshalb von einem Software-Architekten durchgeführt. Eine formale Beschreibung ist nicht erforderlich.

5.1.3.7 Schritt 7: Verifikation und Verfeinerung der Anforderungen und Umwandlung in Einschränkungen für die instanziierten Elemente

In Schritt 5 wurden die Anforderungen an die verfeinerte Komponente c unverändert als Anforderungen an die Sub-Komponenten $c_i \in C_{sub,c}$ übernommen. Als Zwischenschritt ist dies in Ordnung, allerdings für eine weitergehende Ausarbeitung des Architekturentwurfs nicht ausreichend. In der Regel übernehmen die einzelne Sub-Komponenten c_i als Aufgabe nur Teilfunktionalitäten von c . Entsprechend können die Anforderungen an c_i bei Bedarf auf die jeweilige Funktionalität reduziert werden.

Abb. 5-1 skizziert diese Ableitung von Anforderungen anhand eines (sehr einfachen) Beispiels. Gesucht ist die Architektur einer Anwendung, die zu einer gegebenen Zahl die Wurzel berechnet und das Ergebnis formatiert ausgibt. Das initiale Anforderungsmodell enthält eine einzige fachliche Anforderung fr_1, das initiale Anwendungsmodell als einzige Komponente das Gesamtsystem, dem alle Anforderungen (alsofr_1) zugeordnet sind. Schritte 1 bis 5 der

ADD zerlegen nun das Gesamtsystem in zwei weitere Komponenten, eine Berechnungskomponente und einen Ausgabeformatierer. Diesen beiden Komponenten wurden in Schritt 5 die aus fr_1 abgeleiteten Anforderungen fr_2^* bzw. fr_3^* des vorläufigen Anforderungsmodells $rm_{sub,c}$ zugeordnet. Für die Berechnungskomponente ist allerdings der Aspekt der Formatierung irrelevant, umgekehrt ist für den Formatierer die Berechnung unbedeutend. Aus fr_2^* und fr_3^* können deshalb nun in Schritt 7 die für den Berechner und den Formatierer relevanten Anforderungen fr_2 und fr_2 abgeleitet und in das Anforderungsmodell der Ausgabe der ersten ADD-Iteration rm_1 aufgenommen werden.

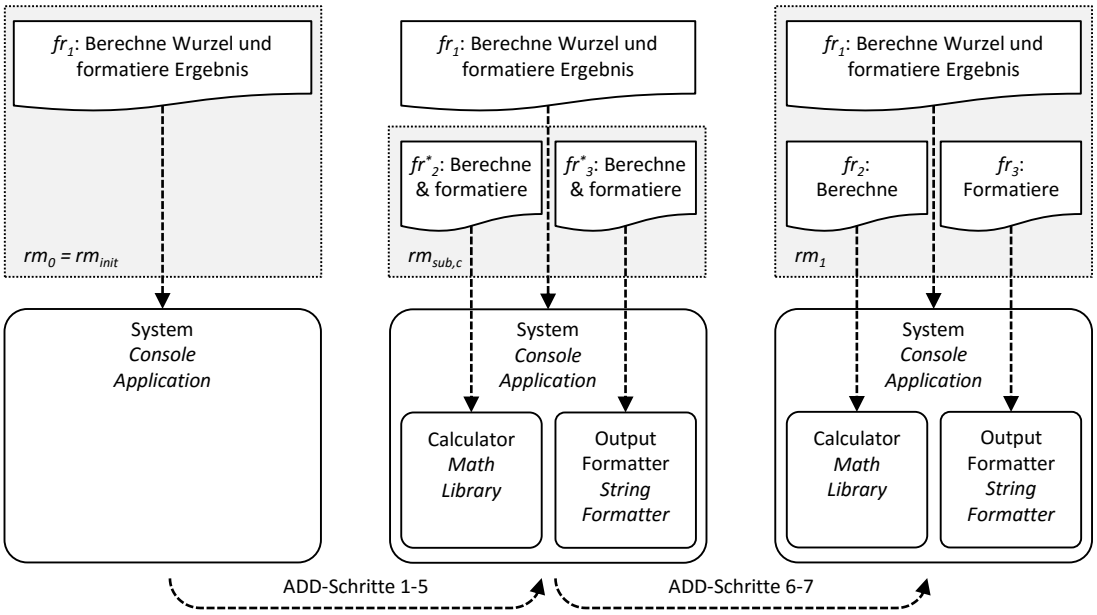

Abb. 5-1: Ableitung von Anforderungen im Rahmen der ADD

Mit Abschluss der ADD erhält ein Software-Architekt einen Architekturentwurf. Dieser enthält ein Anwendungsmodell, das alle Komponenten der Softwarearchitektur enthält, sowie ein zugehöriges Variabilitätsmodell, dass Variabilitäten im Anwendungsmodell über Variabilitätspunkte kennzeichnet. Bei der Durchführung der ADD wird sichergestellt, dass alle Anforderungen des Anforderungsmodells von der Architektur (in allen modellierten, alternativen Ausprägungen) erfüllt werden. Insbesondere kann angenommen werden, dass alle fachlichen Anforderungen erfüllbar sind (eine Unerfüllbarkeit würde bei fortgeschrittener Detaillierung des Entwurfs erkannt) und auch tatsächlich erfüllt werden. Beispielsweise kann ein Anwendungsmodell in einer Alternative die Speicherung von Daten in einer lokal betriebenen Datenbank in einer anderen Alternative in einer Cloud Datenbank vorsehen. Beide Alternativen können die an die Anwendung gestellten funktionalen und qualitativen Anforderungen erfüllen und werden auch nicht durch eine Designeinschränkung verboten.

5.2 Erweiterung der Architecture Tradeoff Analysis Method

Es stellt sich nun die Frage, welche der Alternativen die unter den gegebenen (priorisierten) Anforderungen die „beste“ ist bzw. welche Konsequenzen die Entscheidung für eine und gegen die anderen Alternativen nach sich zieht. Einen Teil der Antworten erhält man durch die Anwendung der Architecture Tradeoff Analysis Method (ATAM) auf den zuvor erstellten Architekturentwurf: nämlich Aussagen zu Abhängigkeiten zwischen Alternativen und Qualitätsattributen, d.h. wo und wie die Auswahl von Alternativen Konsequenzen auf Qualitätsattribute hat.

5.2.1 Formalisierung der Eingabe

Die ATAM erhält folgende Eingabeartefakte:

- *Business Driver*
  Diese beschreiben die Motivation und die Interessensgruppen, die hinter dem Entwicklungsprojekt stecken. Die Business Driver helfen, die Qualitätsattribute des Anwendungssystems zu gewichten und damit letztlich dabei, eine Auswahl aus Architekturalternativen zu treffen. Ihnen kann beispielsweise entnommen werden, ob hohe Verfügbarkeit, schnelle Bereitstellung oder hohe Sicherheit primär angestrebt werden sollten.

- *Qualitative Anforderungen*
  Aus dem für die ADD eingeführten Anforderungsmodell sind in der ATAM nur die qualitativen Anforderungen von Interesse. Es kann vorausgesetzt werden, dass der aus der ADD resultierende Architekturentwurf bereits in allen Alternativen alle fachlichen Anforderungen und Designeinschränkung erfüllt. In der ATAM geht es nun darum, die Alternativen auf den Erfüllungsgrad einzelner Alternativen zu prüfen.

- *Architekturbeschreibung*
  Hier kann das Ergebnis der ADD direkt übernommen werden. Der Architekturentwurf beschreibt die zu erstellende Architektur in dem für die Analyse erforderlichen Umfang.

Nachdem für die qualitativen Anforderungen und die Architekturbeschreibung bereits in Abschnitt 5.1.1 entwickelt bzw. aus [Mie10] entnommen und abgeleitet wurden, wird von den Eingabeartefakten nur für die Business Driver eine formale Beschreibung benötigt.

> *Definition 14* (Business Modell). Gegeben sei die Menge aller Business Modelle $BM = \{\dots, bm_i, \dots\}$. Ein **Business Modell** $bm \in BM$ ist ein Tupel

$$bm = (BD, ST, stakeBD)$$

wobei für die einzelnen Elemente folgendes gilt:

BD	ist eine Menge von Business Drivern,
ST	ist eine Menge von Stakeholdern
$stakeBD$	ist eine Abbildung $(bd \rightarrow st)$.

Ein Business Driver ist eine in Prosa verfasste Beschreibung eines Ziels, das mit der Erstellung der Anwendungssoftware erreicht werden soll. Ein Stakeholder wiederum ist die Beschreibung einer von der Erstellung des Systems betroffener Interessensgruppen (z.B. Anwender, Systembetrieb, Entwickler, …).

> *Regel 4* (Jeder Business Driver hat einen Stakeholder), d.h. jedem Business Driver ist über $stakeBD$ genau ein Stakeholder zugewiesen.

Das Business Modell bringt die Sicht verschiedener Stakeholder in die Evaluierung eines Architekturentwurfs ein. Die Analyse des Entwurfs soll nicht ausschließlich aus technischer Perspektive erfolgen sondern explizit Auswirkungen von Architekturentscheidungen auf qualitative Anforderungen und Geschäftsziele betrachten. Die Business Driver motivieren die Erstellung des Anwendungssystems. Sie sind Grundlage für die Erstellung und Priorisierung der qualitativen Anforderungen. In [BCK03, S.277] werden folgende Beispiele für Business Driver gegeben:

- Die wichtigsten Funktionalitäten des Systems
- Mögliche relevante technische, ökonomische und politischen Einschränkungen
- Geschäftsziele und deren Kontext, sofern sie in Beziehung zum System bestehen

Aus den Business Drivern werden qualitative Anforderungen abgeleitet, d.h. eine solche Anforderung ist nur dann valide, wenn sie von einem Business Driver abstammt. Im Verlauf der ATAM werden für qualitative Anforderungen Szenarien entwickelt, die die Anforderungen konkretisieren und messbar machen. Damit wird die Qualität bzw. der Nutzen des Gesamtsystems greifbar.

In der ebenfalls als Eingabe in die ATAM verwendeten Architekturbeschreibung können vorhandene Alternativen gegeneinander abgewogen werden. Für die einzelnen Alternativen werden hierzu Entscheidungen durchgespielt, d.h. für die im Architekturentwurf enthaltenen Variabilitätspunkte werden alle zulässigen Kombinationen von Alternativen betrachtet und entsprechend den Anforderungen bewertet. Abb. 5-2 zeigt die Ein- und Ausgabeartefakte der ATAM nach [KKC00].

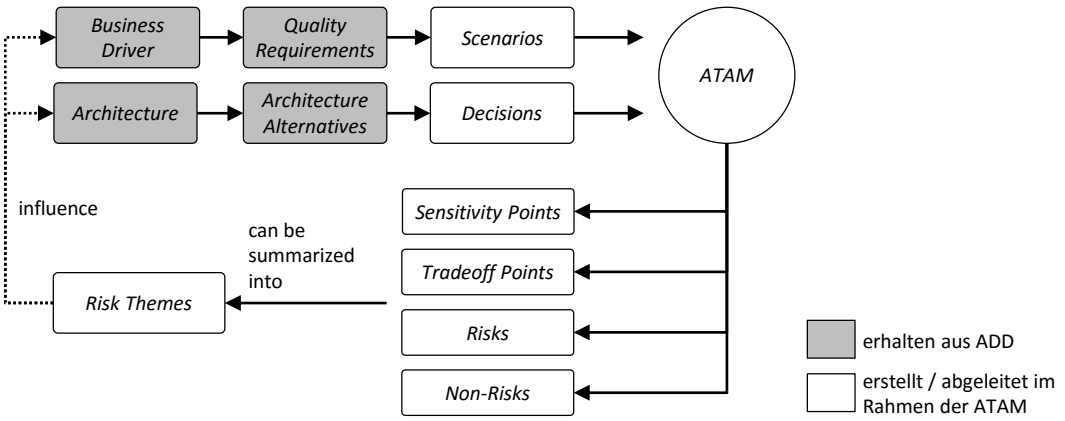

Abb. 5-2: Ein- und Ausgabeartefakte der ATAM (Quelle: [KKC00])

Die ATAM erhält hierzu aus vorgelagerten Arbeitsschritten – in dieser Arbeit sind dies die Schritte bzw. Eingabedaten der ADD – zum einen Business Driver und daraus abgeleitete qualitative Anforderungen. Zum anderen erhält sie den Entwurf einer Softwarearchitektur, welcher alle fachlichen Anforderungen und alles Entwurfseinschränkungen erfüllt. Im Architekturentwurf sind noch Architekturalternativen enthalten.

Im Rahmen der ATAM werden die qualitativen Anforderungen nun durch Szenarien konkretisiert und diese priorisiert. Ebenso werden mögliche, d.h. zulässige, Architekturentscheidungen herausgearbeitet. Diese Entscheidungen werden bezüglich der qualitativen Anforderungen bzw. Szenarien bewertet. Ergebnis ist dann eine Kategorisierung der Entscheidungen. Mögliche Kategorien sind:

- *Sensitivity Point* – die Entscheidung beeinflusst den Grad, zu dem ein bestimmtes Qualitätsattribut erfüllt wird.
- *Tradeoff Point* – die Entscheidung beeinflusst den Erfüllungsgrad mehrerer qualitativer Anforderungen.
- *Risk* – es hängt von der konkreten Entscheidung ab, ob bestimmte qualitative Anforderungen erfüllt werden können.
- *Non-Risk* – die Entscheidung ist unkritisch, d.h. die qualitativen Anforderungen werden unabhängig davon erfüllt für welche Alternative sich die am Entwurf Beteiligten entscheiden.

Jede Entscheidung ist entweder ein Risk oder ein Non-Risk. Eine Entscheidung kann darüber hinaus Sensitivity Point oder Tradeoff Point sein.

Eine Architekturentscheidung ist formal betrachtet die Auswahl einer zulässigen Alternative eines Variabilitätspunktes mit entsprechender Festlegung eines Wertes für die gewählte Alternative. Dies entspricht der Auswahl eines Wertes für einen Variabilitätspunkt im Rahmen der Bindung dieses Variabilitätspunktes in [Mie10]. Als formale Definition für eine Architekturentscheidung kann also festgehalten werden:

> *Definition 15* (Architekturentscheidung). Eine **Architekturentscheidung** ist bezogen auf einen Variabilitätspunkt eine Abbildung
>
> $$AE(vp) \mapsto \begin{cases} explicitValue(a), a \in Alt^{Explicit} \\ buildingBlocks(a), a \in Alt^{Expression} \\ \bot, a \in Alt^{Empty} \end{cases}$$
>
> wobei a eine zulässige Alternative für den Variabilitätspunkt vp ist.
>
> $AE(vp)$ sei definiert als die Menge aller möglichen Architekturentscheidungen eines Variabilitätspunktes vp.

Bei einer Architekturentscheidung wird also aus den zur Verfügung stehenden Alternativen eine ausgewählt. Die Menge der möglichen Alternativen ist die Vereinigungsmenge aus folgenden Arten von Alternativen:

- *explizite Alternativen*
  Die Alternativen sind bereits im Architekturentwurf enthalten. Im Rahmen der ADD wurden für die Implementierung einer Komponente hierzu mehrere Varianten formuliert.

- *Ausdrucksalternativen*
  Werte der Alternativen ergeben sich durch Ableitung aus den gewählten Alternativen anderer Variabilitätspunkte.

- *leere Alternative*
  Der durch vp referenzierte Building Block entfällt.

Eine Architekturentscheidung besteht aus zwei Schritten: Aus den möglichen Alternativen für den Variabilitätspunkt muss in einem ersten Schritt eine Alternative ausgewählt werden. Dies kann beispielsweise eine Alternative aus Alt^{Free} sein, d.h. eine Alternative, bei der ein frei wählbarer Wert gesetzt werden kann. Je nach ausgewählter Alternative kann dann im zweiten Schritt ein Wert für die ausgewählte Alternative (z.B. ein konkreter Wert für „free Value“) festgesetzt werden.

Häufig bestehen Abhängigkeiten zwischen den Alternativen mehrerer Variabilitätspunkte. D.h. wird bezüglich eines Variabiliätspunkts eine Entscheidung getroffen, schränkt dies die möglichen Alternativen eines anderen Variabilitätspunkts ein. Es liegt in der Verantwortung des Software-Architekten, bei der Bewertung des Architekturentwurfs diese Abhängigkeiten zu berücksichtigen und für ein Analysefazit nur solche Architekturalternativen zu betrachten, die auch gültige Kombinationen von Architekturentscheidungen darstellen. Formal seien Architekturalternativen wie folgt definiert.

> *Definition 16* (Architekturalternative). Gegeben sei die Menge aller Architekturentscheidungen AE_{VP} aller Variabilitätspunkte $vp \in VP$ eines gültigen Architekturentwurfs ad. Die Menge aller Architekturalternativen des Architekturentwurfs $AAlt_{ad}$ ist die Teilmenge des kartesischen Produkts $AE(vp_1) \times AE(vp_2) \times \ldots \times AE(vp_n)$ aller Architekturentscheidungen die nur gültige, d.h. miteinander kombinierbare Architekturentscheidungen enthält, d.h.
>
> $$AAlt_{ad} \subseteq AE(vp_1) \times AE(vp_2) \times \ldots \times AE(vp_n)$$
>
> Eine **Architekturalternative** $aalt \in AAlt$ ist ein Tupel $(ae(vp_1), \ldots, ae(vp_n))$ von Architekturentscheidungen, die jedem Variabilitätspunkt genau eine konkrete Architekturentscheidung zuweisen.

Eine Architekturalternative kann somit als Grundlage für eine Implementierung verwendet werden. Sie repräsentiert einen Architekturentwurf, bei dem alle Architekturentscheidungen getroffen sind und der somit keine Variabilitätspunkte mehr enthält. Die Analyse- und Bewertungsmethoden, die nach dem Architekturentwurf angewendet werden, führen zu einer Empfehlung für eine Architekturalternative, deren Architekturentscheidungen in Summe zum höchsten Return-on-Investment führen.

5.2.2 Formalisierung der Ausgabe

Ergebnis der Architekturbewertung im Rahmen der ATAM sind Risiken, Tradeoff Points, Sensitivity Points und Non-Risks. Diese werden im Folgenden mit einer formalen Definition versehen. Die Ergebnisse können letztlich wieder auf die Eingabeartefakte zurückgeführt werden und dort Anpassungen auslösen (Business Driver können hinterfragt, Architekturelemente geändert werden). Ein- und Ausgabeartefakte werden in Abb. 5-2 zusammengefasst dargestellt.

Für die Einstufung von Variabilitätspunkten als Sensitivity Point etc. wird zunächst eine Abbildung von Variabilitätspunkten auf Komponenten benötigt. Dies ist erforderlich, um die von einer Architekturentscheidung betroffene Komponente zu ermitteln.

Definition 17 (Komponente eines Variabilitätspunkts). Die **Komponente eines Variabilitätspunkts** ist die von einem Variabilitätspunkt $vp \in VP$ eines Variabilitätsmodells referenzierte Komponente $c \in C$ eines Anwendungsmodells. Sie kann über folgende Abbildung ermittelt werden

$$compVP(vp) = \{c \in C \mid buildingBlock(locator(vp) \in B(c))\}.$$

Die von einem Variabilitätspunkt referenzierte Komponente ist also die Komponente, aus der ein beliebiger Building Block durch den Locator des Variabilitätspunkts adressiert wird. Da der Variabilitätspunkt auf genau eine Komponente zeigt, kann diese eindeutig bestimmt werden.

Etwas komplexer ist die Bestimmung der Variabilitätspunkte, die eine bestimmte Komponente betreffen. Hier muss nicht nur die Komponente selbst sondern auch alle Sub-Komponenten (die ebenfalls Variabilitäten besitzen können) berücksichtigt werden.

Definition 18 (Variabilitätspunkte einer Komponente). Die **Variabilitätspunkte** $vp \in VP$ **einer Komponente** $c \in C$ eines Anwendungsmodells sind alle Variabilitätspunkte, die über einen Building Block Locator einen Building Block der Komponente selbst oder eines der Sub-Komponenten von c referenzieren.

$$vpComp(c) = \{vp \in VP \mid \exists b \in B^{loc}(vp), \exists c_1 \in c \cup C_{sub,c} : b \in B(c_1)\}$$

Um beurteilen zu können, ob und wie gut eine qualitative Anforderung von einer Architekturentscheidung erfüllt wird, werden zwei Abbildungen benötigt:

Definition 19 (Normalisierter Reaktionsgrad einer qualitativen Anforderung). Gegeben sei eine qualitative Anforderung $qr \in QR_c$ an eine Komponente $c \in C$ sowie eine Architekturentscheidung ae_{vp} bezüglich eines c betreffenden Variabilitätspunkts vp. Der **normalisierte Reaktionsgrad** rg **der qualitativen Anforderung** bezüglich ae sei über folgende Abbildung definiert:

$$rg(qr, ae_{vp}) \mapsto \{r \in \mathcal{R} | 0 \leq r \leq 100\} \cup \text{"unbekannt"}$$

Für zwei Reaktionsgrade $rg_1, rg_2 \neq "unbekannt"$ gelten folgende Restriktionen:
$rg_1(qr, ae_{1,vp}) > rg_2(qr, ae_{2,vp}) \Leftrightarrow ae_1$ erfüllt qr besser als ae_2
$rg_1(qr, ae_{1,vp}) = rg_2(qr, ae_{2vp}) \Leftrightarrow ae_1$ erfüllt qr genauso gut wie ae_2
$rg_1(qr, ae_{1,vp}) < rg_2(qr, ae_{2,vp}) \Leftrightarrow ae_1$ erfüllt qr schlechter als ae_2

Der Reaktionsgrad beschreibt also, wie gut ein bestimmtes Qualitätsattribut erfüllt wird. Wenn das Qualitätsattribut beispielsweise „kurze Antwortzeit" heißt, dann könnte der Reaktionsgrad einer Alternative ae_1 den Wert 50 haben (weil die Antwortzeit beispielsweise 5ms beträgt), der Reaktionsgrad einer Alternative ae_2 den Wert 80 (weil die Antwortzeit mit 3ms besser als bei ae_1 ist) und der Reaktion einer Alternative ae_3 könnte unbekannt sein (weil die Antwortzeit zum Zeitpunkt des Architekturentwurfs nicht abgeschätzt werden kann).

Um eine Aussage darüber treffen zu können, ob der Reaktionsgrad den konkreten Anforderungen entspricht, wird eine Größe benötigt, die festlegt, welcher Reaktionsgrad von einer Alternative mindestens erreicht werden muss. Dies ist der Zielreaktionsgrad.

> *Definition 20* (Zielreaktionsgrad einer qualitativen Anforderung). Der **Zielreaktionsgrad einer qualitativen Anforderung** $qr \in QR$ sei über folgende Abbildung definiert:
>
> $$passRG(qr) \mapsto \{r \in \mathcal{R} | 0 \leq r \leq 100\}$$
>
> Für die Abbildung gelten folgende Restriktionen in Bezug auf eine Architekturentscheidung ae:
> $rg(qr, ae) \geq passRG(qr) \Leftrightarrow ae$ erfüllt die Anforderungen hinsichtlich qr.
> $rg(qr, ae) < passRG(qr) \Leftrightarrow ae$ erfüllt die Anforderungen hinsichtlich qr nicht.

In obigem Beispiel könnte der Zielreaktionsgrad auf 70 festgelegt worden sein (weil eine Antwortzeit von maximal 4ms erforderlich ist). Nun stellt sich die Frage, wie bestimmt werden kann, ob eine Architekturentscheidung eine qualitative Anforderung erfüllt. Diese wird durch folgende Definition beantwortet:

> *Definition 21* (Erfüllung einer qualitativen Anforderung). Gegeben sei der Reaktionsgrad rg einer qualitativen Anforderung $qr \in QR_c$ an eine Komponente $c \in C$ sowie eine Architekturentscheidung $ae(vp)$ bezüglich eines c betreffenden Variabilitätspunkts vp. Die **Erfüllung einer qualitativen Anforderung** sei über folgende Abbildung definiert:
>
> $$pass(qr, ae(vp)) \mapsto \begin{cases} true & \begin{cases} rg(qr, ae(vp) \neq \text{"unbekannt"} \wedge \\ rg(qr, ae(vp) \geq passRG(qr) \end{cases} \\ false & \text{sonst} \end{cases}$$

Demnach erfüllt in obigem Beispiel die Alternative ae_1 die Anforderung, ae_2 erfüllt sie nicht. Auch ae_3 kann die Anforderung nicht erfüllen, da der Reaktionsgrad dieser Alternative unbekannt ist.

Mit diesen Definitionen können nun die Ergebnisse der ATAM ebenfalls formal definiert werden:

Definition 22 (Sensitivität) Zwischen einem Variabilitätspunkt vp und einer Qualitätsanforderung einer Komponente besteht eine **Sensitivität**, wenn das Reaktionsgrad der Qualitätsanforderung abhängig von der Architekturentscheidung bezüglich dieses vp ist. Daraus lässt sich folgende Abbildung für einen Variabilitätspunkt $vp \in VP$ und eine Qualitätsanforderung $qr \in QR$ ableiten:

$$sensitivity(vp, qr) \mapsto \begin{cases} true & \exists\, ae_1, ae_2 \in AE(vp): rg\big(qr, ae_1(vp)\big) \neq rg\big(qr, ae_2(vp)\big) \\ false & \text{sonst} \end{cases}$$

Zwischen dem Variabilitätspunkt $vp_{response}$, zu dem die obigen Alternativen ae_1, ae_2 und ae_3 gehören, besteht eine Sensitivität zur Qualitätsanforderung „kurze Antwortzeit", da der Reaktionsgrad von der jeweiligen Architekturentscheidung abhängt. Es besteht demnach bereits dann eine Sensitivität wenn es in der Menge der möglichen Architekturentscheidungen auch nur eine Entscheidung gibt, für die ein anderer Reaktionsgrad gilt als für eine beliebige andere Entscheidung.

Definition 23 (Sensitivity Point). Ein Variabilitätspunkt vp ist ein **Sensitivity Point**, wenn es zu der von vp referenzierten Architekturkomponente $c = compVP(vp)$ eine Qualitätsanforderung $qr \in QR_c$ gibt, für die eine Sensitivität zwischen qr und vp besteht, d.h.

$$sensP(vp) \mapsto \begin{cases} true & \exists\, qr \in QR_c, c \in compVP(vp): sensitivity(vp, qr) = true \\ false & \text{sonst} \end{cases}$$

Nachdem zwischen $vp_{response}$ und der Qualitätsanforderung „kurze Antwortzeit" eine Sensitivität besteht, ist der Variabilitätspunkt ein Sensitivity Point. Gäbe es eine weitere Qualitätsanforderung, zu der $vp_{response}$ eine Sensitivität besitzt, wäre der Variabilitätspunkt zugleich auch Tradeoff Point.

Definition 24 (Tradeoff Point) Ein Variabilitätspunkt vp ist ein **Tradeoff Point**, wenn es zur von vp referenzierten Architekturkomponente $c = compVP(vp)$ mehr als ein Qualitätsattribut $qr_i \in QR_c$ gibt, für das eine Sensitivität zwischen qr_i und vp besteht, d.h.

$$tradeoffP(vp) \mapsto \begin{cases} true & \begin{cases} \exists\, qr_1, qr_2 \in QR_c, qr_1 \neq qr_2, c \in compVP(vp): \\ sensitivity(vp, qr) = true \end{cases} \\ false & \text{sonst} \end{cases}$$

Mit diesen beiden Definitionen kann eine Antwort auf die Frage gegeben werden, ob zwischen einem Variabilitätspunkt und einer (Sensitivity) oder mehrerer (Tradeoff) qualitativer Anforderungen Abhängigkeiten bestehen. Es stellt sich darüber hinaus die Frage, wie diese Antworten zu bewerten sind. Interessant sind Fälle, in denen mindestens eine Alternative die

Anforderung verletzt. Dann muss der Sensitivity bzw. Tradeoff Point als Risiko angesehen werden.

> *Definition 25* (Risk) Ein Variabilitätspunkt vp ist ein **Risk**, wenn vp ein Sensitivitätspunkt ist und die Auswirkung einer der Alternativen $ae(vp)$ entweder nicht abgeschätzt werden können oder mindestens eine Alternative die Anforderungen mindestens einer Qualitätsanforderung der betroffenen Architekturkomponente verletzt, d.h.

$$riskP(vp) \mapsto \begin{cases} true & \exists ae \in AE(vp), \exists qr \in QR_{compVP(vp)} : pass\big(qr, ae(vp)\big) = \text{"false"} \\ false & \text{sonst} \end{cases}$$

In obigem Beispiel ist $vp_{response}$ ein Risk, da die Alternativen ae_2 und ae_3 die Anforderung verletzen. Bei der Auswahl der Alternativen muss besonderes Augenmerk auf die Variabilitätspunkte gelegt werden, die als Risk eingestuft sind.

Bei Non-Risks ist die Situation deutlich besser. Bei diesen erfüllen alle Alternativen die qualitativen Anforderungen, es kann also frei eine der Alternativen ausgewählt werden.

> *Definition 26* (Non-Risk) Ein Variabilitätspunkt vp ist ein **Non-Risk**, wenn vp ein Sensitivitätspunkt ist, die Auswirkungen aller Alternativen abgeschätzt und sicher angenommen werden kann, dass die Auswirkung aller Alternativen $ae(vp)$ die Anforderungen aller Qualitätsattribute der betroffenen Architekturkomponente erfüllen, d.h.

$$nonriskP(vp) \mapsto \begin{cases} true & \forall ae \in AE(vp), \forall qr \in QR_{compVP(vp)} : pass\big(qr, ae(vp)\big) = \text{"true"} \\ false & \text{sonst} \end{cases}$$

Die Mengen der Sensitivity Points, Tradeoff Points, Risks und Non-Risks können zu einem Risikomodell zusammengefasst werden.

> *Definition 27* (Risikomodell) Gegeben sei die Menge aller Risikomodelle $RiskM = \{\dots, riskm_i, \dots\}$. Ein auf einen bestimmten Architekturentwurf AD und ein bestimmtes Anforderungsmodell RM bezogenes **Risikomodell** $riskm_{AD,RM} \in RiskM$ ist ein Tupel
>
> $$riskm_{AD,RM} = (SENSP, TOFFP, RISKS, NONRISKS, sensitivities)$$
>
> wobei für die einzelnen Elemente folgendes gilt:
>
> | | |
> |---|---|
> | $SENSP$ | ist eine Menge von Sensitivity Points, d.h. $SENSP = \{vp \in VP \mid sensP(vp) = true\}$ |
> | $TOFFP$ | ist eine Menge von Tradeoff Points, d.h. $TOFFP = \{vp \in VP \mid tradeoffP(vp) = true\}$ |
> | $RISKS$ | ist eine Menge von Risks, d.h. |

	$RISKS = \{vp \in VP \mid riskP(vp) = true\}$
$NONRISKS$	ist eine Menge von Non-Risks, d.h.
	$NONRISKS = \{vp \in VP \mid nonriskP(vp) = true\}$
$sensitivities$	ist eine Abbildung ($VP \rightarrow QR \cup \perp$).

Das Risikomodell gibt für einen gegebenen Architekturentwurf und ein gegebenes Anforderungsmodell einen Überblick darüber, welche Zusammenhänge zwischen Entscheidungen und Qualitätsattributen bestehen (angegeben über $SENSP$ und $sensitivities$) auf welche Architekturentscheidungen die am Prozess beteiligten Personen besonderes Augenmerk richten müssen ($TOFFP$ und $RISKS$) bzw. wo unkritische Entscheidungen anstehen ($NONRISKS$).

Zu beachten ist, dass nicht jedem Variabilitätspunkt notwendigerweise auch mindestens ein Qualitätsattribut zugewiesen sein muss. Ein solcher Fall (ein Variabilitätspunkt ist formal $\perp$ zugewiesen) bedeutet allerdings, dass die den Variabilitätspunkt betreffende Architekturentscheidung nicht auf Basis einer qualitativen Abwägung getroffen werden kann. In der Regel kann diese Entscheidung dann rein aufwandsbezogen erfolgen (die kostengünstigste Alternative wird gewählt).

5.2.3 Erweiterung der Bearbeitungsschritte

Im Folgenden sollen die in [KCC00] nur informell beschriebenen Bearbeitungsschritte in formelle Bearbeitungs- und Ableitungsregeln überführt werden. Dies eröffnet später die Möglichkeit, die Methodik der ATAM mit den beiden anderen Methoden zum Anwendungsentwurf (ADD) und zur ökonomischen Bewertung des Entwurfs (CBAM) zusammenzuführen. Ein- und Ausgabeartefakte der ATAM sind in Abb. 5-2 skizziert.

5.2.3.1 Schritt 1: Präsentation der Vorgehensweise

Dieser Schritt ist rein organisatorischer Natur und bedarf keiner Formalisierung. Das Team, welches die ATAM durchführt, wird in die Vorgehensweise eingeführt. Bedeutung von Ein- und Ausgabeartefakten sowie die Bearbeitungsschritte werden erläutert.

5.2.3.2 Schritt 2: Präsentation der Business Driver

Auch dieser Schritt ist eher organisatorischer Natur. Hier werden die Business Driver bzw. das Business Modell vorgestellt. Dieses ist wichtig, um in weiteren Bearbeitungsschritten die Bedeutung, d.h. Priorität von Qualitätsattributen bzw. abgeleiteten Szenarien einschätzen zu können. Das Business Modell sollte deshalb darauf geprüft werden, ob jede Qualitätsanforderung mit einem entsprechenden Business Driver hinterlegt ist bzw. ob folgende Regel durch das Business Modell erfüllt ist.

Regel 5 (Vollständigkeit eines Business Modells). Ein Business Modell ist vollständig, wenn jede Qualitätsanforderung $qr \in QR$ durch mindestens einen Business Driver motiviert ist, d.h.

$$\forall\, qr \in QR, \exists bd \in BD : bdriverQR(bd) = qr$$

Mit dieser Regel wird sichergestellt, dass die Motivation für die verschiedenen qualitativen Anforderungen geklärt ist. Das Analyseteam, das die ATAM durchführt, erhält damit den Rahmen für die Einschätzung und Bewertung von Konsequenzen einzelner Architekturentscheidungen, d.h. die Bedeutung von qualitativen Anforderungen wird geklärt.

5.2.3.3 Schritt 3: Präsentation der Architektur

In diesem Schritt wird der zu analysierende Architekturentwurf vorgestellt. Es wird das Architekturmodell präsentiert und das Variabiliätsmodell erläutert. Das Analyseteam, das die ATAM durchführt, erfährt damit, welche Architekturentscheidungen bewertet und kategorisiert werden müssen. Eine Formalisierung dieses Bearbeitungsschrittes ist nicht erforderlich. Es kann angenommen werden, dass Vollständigkeit und Korrektheit des Architekturentwurfs bereits im Rahmen des Entwurfsprozesses sichergestellt werden.

5.2.3.4 Schritt 4: Identifikation von Architekturansätzen

Architekturansätze sind alternative Wege für eine Architekturimplementierung, mit denen die gewünschten Funktionalitäten in gewünschter Qualität erreicht werden sollen. Im vorliegenden Fall entsprechen die Architekturansätze Architekturalternativen. Eine weitergehende Formalisierung dieses Schrittes ist nicht erforderlich. Die Architekturansätze werden durch die Variabilitätspunkte des Variabilitätsmodells bereits ausreichend beschrieben.

5.2.3.5 Schritt 5: Erstellung eines Qualitätsattribute-Baums

Ein zentrales Artefakt, das im Rahmen der ATAM und einiger anderer Methoden für die Entscheidungsfindung verwendet wird, ist das Szenario. In [IAH05] unterstreichen Ionita u.a. den Nutzen von Szenarien in der Unterstützung von Entscheidungen zwischen Architekturalternativen. Ein Szenario konkretisiert, was durch qualitative Anforderungen nur abstrakt formuliert wird. Es beschreibt dabei in einem kurzen – häufig in Prosa verfassten – Statement eine Interaktion mit dem System. Die Beschreibung enthält dabei Informationen zu einem Stimulus, d.h. einem Impuls, der von außen auf das System ausgeübt wird, zur Umgebung, in der der Impuls stattfindet und zur Reaktion, die der Stimulus im System auslöst. Diese Reaktion sollte messbar sein. Damit wird die Qualität des Systems überprüfbar.

Ein Beispiel: eine Qualitätsanforderung an ein System könnte wie folgt formuliert sein: „das System muss performant sein“. Ein Szenario, das sich auf dieses Attribut bezieht, könnte wie folgt beschrieben werden: „Anwender können sich über die Web-Oberfläche bei 1000 gleichzeitig angemeldeten Usern (Umgebung) Daten anfragen (Stimulus) mit einer Latenz von maximal 2 Sekunden anzeigen (Response) lassen.“

Definition 28 (Szenarienmodell). Gegeben sei die Menge aller Szenarienmodelle $SM = \{\dots, sm_i, \dots\}$. Ein **Szenarienmodell** $sm \in SM$ ist ein Tupel

$$sm = (S, scenQR, scenPrio)$$

wobei für die einzelnen Elemente folgendes gilt:

S	ist eine Menge von Szenarien,
$scenQR$	ist eine Abbildung $(s \rightarrow qr)$,
$scenPrio$	ist eine Abbildung $(s \rightarrow Priorität)$.

Das Szenarienmodell muss vollständig und konsistent sein, d.h. jede Qualitätsanforderung muss durch mindestens ein Szenario umschrieben sein und zu jedem Szenario muss es genau eine Qualitätsanforderung geben, die durch dieses Szenario beschrieben wird. Folgende Regeln müssen also erfüllt sein:

Regel 6 (Vollständigkeit eines Szenarienmodells). Ein Szenarienmodell $sm \in SM$ ist vollständig, wenn jede Qualitätsanforderung $qr \in QR$ durch mindestens ein Szenario konkret beschrieben wird, d.h.

$$\forall\, qr \in QR, \exists s \in S : scenQR(s) = qr$$

Regel 7 (Konsistenz eines Szenarienmodells). Ein Szenarienmodell $sm \in SM$ ist konsistent, wenn jedes Szenario $s \in S$ genau eine Qualitätsanforderung $qr \in QR$ konkret beschreibt, d.h.

$$\forall s \in S, qr_1 \in QR, scenQR(s) = qr_1 : \nexists qr_2 \in QR, qr_1 \neq qr_2 : scenQR(s) = qr_2$$

Szenarien konkretisieren also qualitative Anforderungen. Diese wiederum machen für einzelne Architekturkomponente Vorgaben dafür, „wie gut“ bestimmte Kriterien erfüllt sein müssen. Diese Güte ist bei Komponenten, bei denen über Variabilitätspunkte alternative Umsetzungen beschrieben sind, abhängig von der Auswahl der konkreten Alternative. Diese Zusammenhänge werden in Abb. 5-3 skizziert.

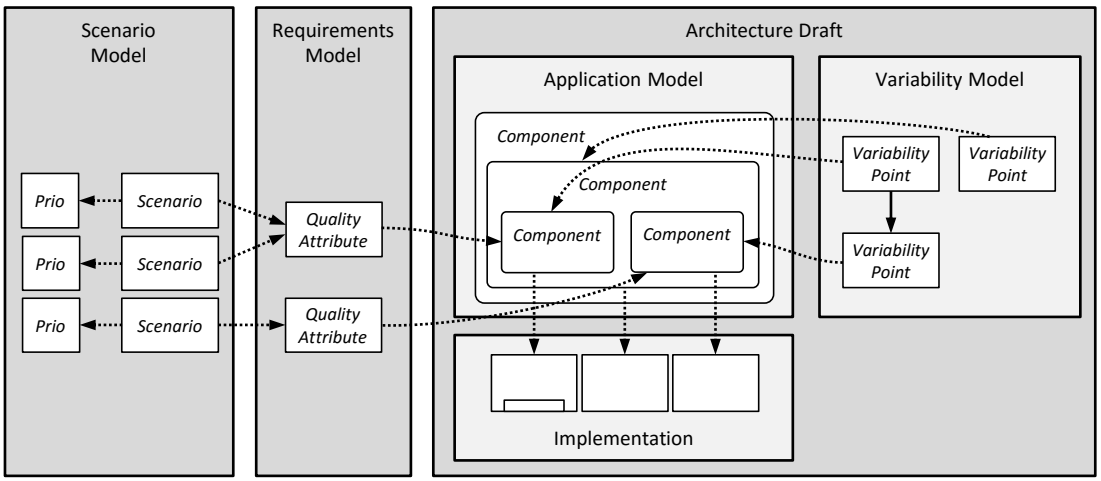

Abb. 5-3 Beziehung zwischen Szenarien-, Anforderungsmodell und Architekturentwurf

Das Szenarienmodell wird über Ableitungsschritte aus der Menge der qualitativen Anforderungen gewonnen, wobei hier vielfach der Erfahrungsschatz beteiligter Software-Architekten zum Einsatz kommt. Ergebnis der Ableitung ist ein Qualitätsattribute-Baum, an dessen Blättern die Szenarien stehen.

```
// initialisiere eine leere Liste für Szenarien
var scenarios = new List();

// initialisiere eine Queue mit vorläufigen Szenarien
// mit den qualitativen Anforderungen des Anforderungsmodells,
// wobei qr.Parent = null für alle qr aus QR
var preScenarios = new Queue( QR );

// solange es noch vorläufige Szenarien in der Queue gibt
while( preScenarios.Count > 0 )
{
   // nimm das nächste Szenario aus der Queue heraus
   var nextPreScenario = preScenarios.Dequeue();

   // wenn dieses Szenario bereits ausreichend detailliert ist
   if( isDetailedEnough( nextPreScenario ) )
   {
      // füge das Szenario dem Szenarienmodell hinzu
      sm.S.Add( nextPreScenario );
      sm.scenQR.Add( new Branch( nextPreScenario.Parent, nextPreScenario ) );
      // füge das Szenario der Liste der Szenarien hinzu
      scenarios.Add( nextPreScenario );
   }
   // wenn das Szenario noch nicht ausreichend detailliert ist
   else
   {
      // erstelle eine Liste detaillierender Szenarien s, bei denen
      // jeweils s.Parent = nextPreScenario gesetzt ist
```

```
            var listOfMoreDetailedScenarios =
               createListOfMoreDetailedScenarios( nextPreScenario );

            // füge diese Szenarien zur ggf. weiteren Detaillierung an
            // die Queue an
            preScenarios.Enqueue( listOfMoreDetailedQr );
         }
      }
```

Listing 5-1: Ableitung eines Szenarienmodells aus gegebenen qualitativen Anforderungen

Dieser Prozess führt zu einer Liste von Szenarien. Für jedes darin hinterlegte Szenario ist angegeben, auf welche Qualitätsanforderung es sich bezieht. Ergebnis ist ein Qualitätsattribute-Baum, wie er in Abb. 5-4 skizziert ist.

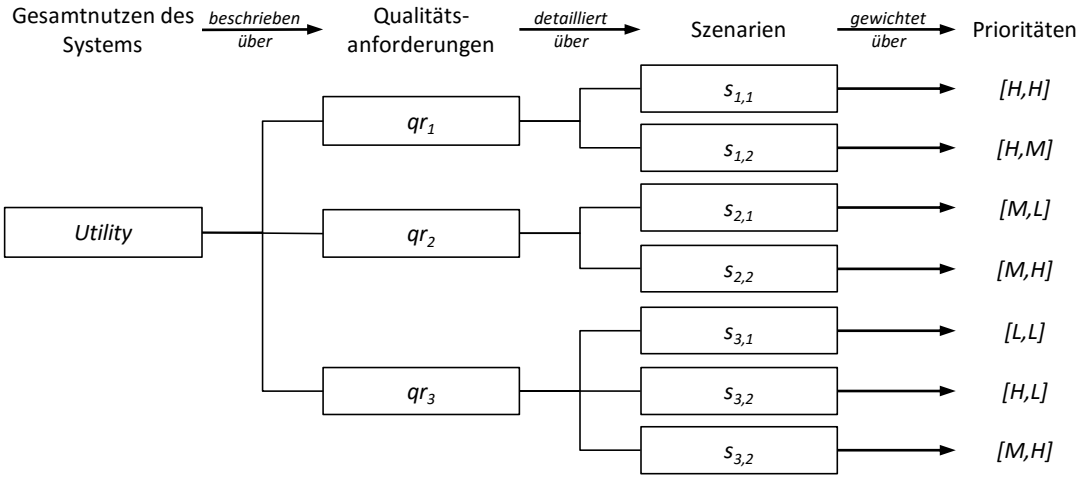

Abb. 5-4: Erstellung eines Utility-Trees aus Szenarien und deren Prioritäten

Für die abgeleiteten Szenarien wird in den folgenden Schritten ermittelt, wie gut sie über einzelne Architekturentscheidungen erfüllt werden. Besonderes Augenmerk soll dabei auf Szenarien liegen, die zum einen aus fachlicher Sicht als wichtig eingestuft werden und zum anderen für die Implementierung Herausforderungen darstellen. Die Szenarien werden dazu priorisiert. Im Prinzip kann hierzu eine beliebige Priorisierungsstrategie herangezogen werden. An dieser Stelle soll die in [KKC00] vorgeschlagene Strategie übernommen werden. Szenarien werden Tupel von zwei Werten „high“, „medium“ und „low“ zugewiesen, wobei der erste Wert die Bedeutung des Szenarios für den Gesamterfolg des Systems, der zweite Wert die Komplexität bei der Umsetzung zur Erreichung dieses Szenarios beschreibt.

Diese Art der Priorisierung ist für die weiteren Schritte absolut ausreichend. Zur Reduzierung des Gesamtprozesses kann festgelegt werden, dass nicht alle Szenarien analysiert werden,

sondern der Fokus auf die höher priorisierten gelegt wird (z.B. nur auf Szenarien, die als [H,H], [H,M] und [M,H] eingestuft wurden; in Abb. 5-4 also nur die Szenarien $s_{1,1}$, $s_{1,2}$, $s_{2,2}$ und $s_{3,2}$).

5.2.3.6 Schritt 6: Analyse der Architekturansätze (1. Iteration)

Dieser Schritt ist der wohl wichtigste im Rahmen der ATAM. Hier werden für die hoch priorisierten Szenarien zunächst die betroffenen qualitativen Anforderungen ermittelt, für diese wiederum die betroffenen Architekturkomponenten und zugehörige Variabilitätspunkte. Für letztere wird dann einzeln analysiert, ob es sich um einen Sensitivity Point, einen Tradeoff Point, ein Risk oder ein Non-Risk handelt. Der Algorithmus zur Analyse ist in Listing 5-2 in Pseudo-C#-Code wiedergegeben.

```
// Algorithmus zur Bestimmung von Sensitivitäten, Tradeoff Points,
// Risks und Non-Risks (jeweils Listen von Variabilitätspunkten)
var sensitivityPoints = new List<VP>();
var tradeoffPoints    = new List<VP>();
var risks             = new List<VP>();
var nonRisks          = new List<VP>();
var sensitivities     = new Dictionary<VP, List<QR>>();

// initialisiere die sensitivitys für alle Variabilitätspunkte
foreach( var vp in VP )
{
   sensitivities.Add( VP, new List<QR>() );
}

// analysiere alle Szenarien
foreach( var scenario in scenarios )
{
   // wenn das Szenario hoch priorisiert wurde, also z.B. Priorität ist
   // [H,H], [H,M], [M,H], dann analysiere das Szenario
   if( isHighPrio( scenario ) )
   {
      // ermittle die Qualitätsanforderung, die durch das Szenario
      // konkretisiert wird
      var qr = scenQR( scenario );

      // ermittle die Architekturkomponente, die durch die Qualitäts-
      // anforderung referenziert wird
      var c = qreqC( qr );

      // ermittle alle Variabilitätspunkte, die Alternativen für die
      // Komponente definieren
      var vpComp = vpComp( c );

      foreach( var vp in vpComp )
      {
         // analysiere, ob es sich bei dem Variabilitätspunkt um eine
         // Sensitivität, einen Tradeoff Point, ein Risk oder ein
         // Non-Risk handelt.
```

```
            if ( sensP( vp ) )
            {
               sensitivityPoints.Add( vp );
               sensitivities[ vp ].Add( qr );
            }
            if ( tradeoffP( vp ) ) tradeoffPoints.Add( vp );
            if ( riskP( vp ) )     risks.Add( vp );
            if ( nonriskP( vp ) )  nonRisks.Add( vp );
         }
      }
   }
```

Listing 5-2: Analyse von Architekturansätzen im Rahmen der ATAM (1. Iteration)

Das Herzstück des Algorithmus, in dem die eigentliche Bewertungsarbeit steckt, wird durch die vier Analyse-Statements am Ende gebildet. Durch diese wird der betreffende Variabilitätspunkt daraufhin analysiert, ob er ein Sensitivity Point, ein Tradeoff Point, ein Risk oder ein Non-Risk ist. Hinter diesen Funktionen verbergen sich die entsprechenden Definitionen aus Abschnitt 5.2.2. Ergebnis dieses Algorithmus sind vier Listen, die für die einzelnen Szenarien die relevanten Variabilitätspunkte mit entsprechender Klassifizierung enthalten.

5.2.3.7 Schritt 7: Brainstorming und Priorisierung von Szenarien

In diesem Schritt werden zusätzliche Stakeholder (z.B. zukünftige Nutzer des Systems, Vertreter des Systembetriebs, …) hinzugezogen. Diese entwickeln – zunächst unabhängig von den Ergebnissen der Erstellung des Qualitätsattribute-Baums in 5.2.3.5 – weitere Szenarien aus ihrer Erwartungshaltung an das System. Grundsätzlich gelten für diese Szenarien die gleichen Anforderungen und Formalismen wie für die oben bereits beschriebenen Szenarien. Es obliegt dem Software-Architekten, etwaige Inkonsistenzen bezüglich Inhalt und Priorität der neuen und der bestehenden Szenarien zu identifizieren und zu diskutieren. Eine weitere Formalisierung dieses Arbeitsschrittes ist nicht erforderlich.

5.2.3.8 Schritt 8: Analyse der Architekturansätze (2. Iteration)

Mit der so gewonnenen neuen Liste an Szenarien kann ein analoger Analyseprozess wie unter 5.2.3.6 durchlaufen werden. Dieser wird in folgendem Listing beschrieben:

```
// analysiere alle Szenarien
foreach( var scenario in scenarios )
{
   // wenn das Szenario hoch priorisiert wurde, also z.B. Priorität ist
   // [H,H], [H,M], [M,H], dann analysiere das Szenario
   if( isHighPrio( scenario ) )
   {
      // ermittle die durch das Szenario betroffenen Architekturkomponenten
      // die Methode scenarioC ist ein Platzhalter für einen manuellen, durch
```

```
      // einen Software-Architekten durchgeführten Schritt
      var Components = scenarioC ( scenario );

      // analysiere jede dieser Komponenten
      foreach( var c in Components )
      {
         // ermittle alle Variabilitätspunkte, die Alternativen für die
         // Komponente definieren
         var VP = VP( c );

         // analysiere, ob es sich bei dem Variabilitätspunkt um eine
         // Sensitivität, einen Tradeoff Point, ein Risk oder ein
         // Non-Risk handelt.
         if ( sensP( VP ) )     sensitivities.Add( scenario, VP );
         if ( tradeoffP( VP ) ) tradeoffPoints.Add( scenario, VP );
         if ( riskP( VP ) )     risks.Add( scenario, VP );
         if ( nonriskP( VP ) )  nonRisks.Add( scenario, VP );
      }
   }
}
```

Listing 5-3: Analyse von Architekturansätzen im Rahmen der ATAM (2. Iteration)

Der Algorithmus ist weitestgehend identisch zum Algorithmus in 5.2.3.6. Wichtigster Unterschied ist, dass die Ermittlung der vom Szenario betroffenen Architekturkomponenten nicht über die Qualitätsattribute erfolgen kann. Hier ist der Erfahrungsschatz des Architekten gefragt, die für die Analyse heranzuziehenden Komponenten zu identifizieren.

5.2.3.9 Schritt 9: Präsentation der Ergebnisse

Dieser Schritt ist wieder organisatorischer Natur. Hier werden die ermittelten Szenarien und deren zugehörigen Sensitivity Points, Tradeoff Points, Risks und Non-Risks zusammengestellt und präsentiert. Eine weitergehende Formalisierung ist nicht erforderlich.

5.3 Erweiterung der Cost Benefit Analysis Method

Nach Abschluss der ATAM liegt neben dem Architekturentwurf auch eine Übersicht über die Qualität und die Risiken von Entwurfsentscheidungen vor. Damit sind Aussagen darüber möglich, welche Alternativen zu welchem Qualitätsgrad führen. Ohne weitere Restriktionen könnte der Gesamtentwurfsprozess mit der Maßgabe beendet werden, in den Variabilitätspunkten einfach die Architekturalternative zu wählen, die zu einer Qualitätsmaximierung führt. Dieser Ansatz besitzt jedoch entscheidende Nachteile:

- Höhere Qualität wird automatisch mit einem höheren Return-on-Investment assoziiert.

- Der Preis der Qualität bleibt unberücksichtigt, d.h. der Erreichung bestimmter Qualitätsattribute werden keine entsprechenden Aufwände gegenübergestellt.
- Die einzelnen Qualitätsattribute werden als jeweils gleich wichtig betrachtet.[2]

Um beispielsweise das Ziel einer möglichst hohen Ausfallsicherheit zu erreichen könnte festgelegt werden, grundsätzlich alle Systemkomponenten redundant auszulegen. Die ATAM ignoriert, dass dies zweifellos zu (möglicherweise unnötig) hohen Kosten bei der Umsetzung führen würde.

Die Cost Benefit Analysis Method adressiert diese Nachteile insofern, als dass sie eine Begrenztheit in Zeit und Umsetzungsaufwand in die Bewertung einbezieht. Um dabei nicht ins andere Extrem – es wird grundsätzlich die Alternative mit dem geringsten Aufwand gewählt, die die Anforderungen gerade noch erfüllt – zu verfallen, wird dem Reaktionsgrad einer qualitativen Anforderung Nutzenwert zugewiesen. Dieser ist normalisiert, wodurch eine Vergleichbarkeit der Reaktionsgrade erreicht wird. Zusammen mit einer Priorisierung der Anforderungsszenarien können Architekturalternativen hinsichtlich ihres ROI-Beitrags bewertet werden, d.h. die Beantwortung folgender Fragen wird möglich:

- Welcher Reaktionsgrad einer qualitativen Anforderung kann durch eine Architekturentscheidung erzielt werden?
- Wie hoch ist der Nutzwert, der durch diesen Reaktionsgrad bei Umsetzung der Alternative erzielt werden kann?
- Wie hoch ist der Aufwand (Zeit, Kosten etc.) zur Umsetzung?

Somit liegen die Informationen zur Ermittlung des Return-on-Investment vor: der Nutzen und der Aufwand. Diese beiden Werte zueinander ins Verhältnis gesetzt, ergeben den Return-on-Investment.

5.3.1 Formalisierung der Eingabe

Nachdem ATAM und CBAM mit der Carnegie Mellon Universität den gleichen Ursprung haben, ist deren Kompatibilität keine Überraschung: im Laufe der CBAM werden die Artefakte wiederverwendet, die bereits Eingabe in die ATAM bzw. dort gebildet wurden. Insbesondere die Modelle des Architekturentwurfs (einschließlich dessen Variabilitäten), das Anforderungsmodell sowie das Szenarienmodells finden in der CBAM wieder Verwendung. Es werden keine weiteren Eingaben benötigt. Alle weiteren Artefakte werden im Rahmen der CBAM entwickelt.

[2] Die Priorisierung im Rahmen der ATAM diente nur dazu, eine Reduzierung der Analysekomplexität durch Beschränkung auf die „wichtigen" Szenarien zu erreichen, nicht, um die einzelnen Qualitätsattribute gegeneinander abzuwägen.

5.3.2 Formalisierung der Ausgabe

CBAM leitet aus den bisher gewonnenen Artefakten das finale Ziel der gesamten Architekturbewertung ab: den Return-on-Investment für jede einzelne Architekturentscheidung $ae \in AE$. Gewonnen wird eine Abbildung

$$roi(ae) \longmapsto \mathbb{R}$$

Mit dieser Abbildung werden nun alle anstehenden Architekturentscheidungen direkt vergleichbar. Die Entscheidung, die mutmaßlich zu einem höheren ROI führt kann einer solchen vorgezogen werden, deren ROI niedriger ist. In der Gesamtbetrachtung können die Werte aller in einer Architekturalternative enthaltenen Architekturentscheidungen zusammengeführt werden.

$$roi(aalt) = \sum_{vp \in VP} roi(ae_{aalt}(vp))$$

5.3.3 Erweiterung der Bearbeitungsschritte

Ausgehend von den Eingaben, die zuvor aus den vorgelagerten Aktivitäten von ADD und ATAM gewonnen wurden, werden nun neun Arbeitsschritte durchgeführt, aus denen schließlich die Aussagen zum ROI der einzelnen Architekturalternativen gewonnen werden.

5.3.3.1 Schritt 1: Sortierung von Szenarien

Wie bei der ATAM stehen auch bei der CBAM Szenarien im Zentrum der Betrachtungen. Sie repräsentieren die qualitativen Anforderungen an die zu erstellende Software. In diesem ersten Schritt haben die beteiligten Stakeholder erneut die Möglichkeit, neue Szenarien in die Betrachtungen einzubringen. Eine darüber hinausgehende Erweiterung, die Auswirkungen auf eines der betrachteten Modelle hätte, erfolgt nicht.

5.3.3.2 Schritt 2: Verfeinerung von Szenarien

Für die Szenarien wird nun eine Verfeinerung durchgeführt, die darin besteht, für jedes einzelne Szenario konkrete Zielerreichungsgrade festzulegen. Dies sind – häufig in Prosa – formulierte Werte, die beschreiben, wie gut ein bestimmtes Szenario vom System erfüllt wird. Folgende Zielerreichungsgrade werden dabei unterschieden:

- „Worst“ – der schlechtest mögliche (noch akzeptable) Erreichungsgrad
- „Current“ – der Erreichungsgrad, der mit der aktuellen Lösung erzielt wird
- „Desired“ – der mit der neuen Lösung angestrebte Erreichungsgrad

- „Best“ – der beste, überhaupt mögliche Erreichungsgrad

Ein Beispiel: Die Qualitätsanforderung an ein System zur Bereitstellung von Ergebnissen einer Datenbankabfrage könnte „hohe Performanz“ lauten. Ein Szenario hierzu wäre „gute Antwortzeiten für Benutzer“ lauten. Die Erreichungsgrade könnten nun in Millisekunden ausgedrückt sein, d.h. „>100ms“ (worst), „50ms“ (current), „10ms“ (desired) und „<5ms“ (best).

Formal wird also eine Abbildung benötigt, die einem Szenario diese Erreichungsgrade zuordnet:

> *Definition 29* (Level-bezogener Zielerreichungsgrad eines Szenarios). Die **Level-bezogenen Zielerreichungsgrade** eines Szenarios $s \in S$ seien über folgende Abbildung definiert:
>
> $$respGoal(s, level) \mapsto responseGoal \cup \perp$$
>
> Wobei gilt: $level \in \{\text{worst, current, desired, best}\}$. $responseGoal$ ist eine – ggf. in Prosa verfasste – Beschreibung eines Zielerreichungsgrads.

Diese Beschreibungen der Zielerreichungsgrade spannen den Rahmen auf, in dem dann die Bewertung des tatsächlichen, durch eine Architekturentscheidung erzielten Nutzens erfolgen kann. Sofern im Rahmen des Softwareprojekts gänzlich neue Komponenten entwickelt werden, also noch keine bestehende Software geändert wird und demzufolge noch keine Komponenten existieren, mit denen ein entsprechender aktueller Zielerreichungsgrad verbunden werden kann, ist der Wert für das Level „current“ leer, also $\perp$.

5.3.3.3 Schritt 3: Priorisierung von Szenarien

In Schritt der 3 der CBAM wird erneut eine Priorisierung der Szenarien durchgeführt. Diese wird benötigt, um die Bedeutung des Nutzens einer einzelnen Architekturentscheidung im Rahmen des Gesamtnutzens aller Entscheidungen zu beurteilen. Ein vergleichsweiser geringer Zuwachs des Zielerreichungsgrads einer sehr wichtigen Komponente kann damit mit einem höheren Nutzenzuwachs bewertet werden als eine starke Verbesserung des Zielerreichungsgrads einer unbedeutenden Komponente.

Im Rahmen der Priorisierung der Szenarien können alle an der Architekturplanung Beteiligten Stimmen für Szenarien abgeben, die in Summe jeweils die Bedeutung des Szenarios für den Gesamtnutzen des Systems wiederspiegeln.

> *Definition 30* (Bedeutung eines Szenarios). Die **Bedeutung eines Szenarios** $s \in S$ sei über folgende Abbildung definiert:

$votesScen(s) \mapsto votes$

Wobei gilt: $votes \in \mathbb{N}$.

Für die Bedeutung zweier Szenarien s_1, s_2 gilt dabei folgendes:
$votesScen(s_1) > votesScen(s_2) \Leftrightarrow s_1$ist wichtiger für den Gesamtnutzen als s_2,
$votesScen(s_1) = votesScen(s_2) \Leftrightarrow s_1$ist für den Gesamtnutzen genauso wichtig wie s_2,
$votesScen(s_1) < votesScen(s_2) \Leftrightarrow s_1$ist weniger wichtig für den Gesamtnutzen als s_2.

Die eigentliche Abbildung kann dabei über eine Stimmabgabe der am Prozess Beteiligten erfolgen. Eine Möglichkeit ist, jedem Beteiligten eine bestimmte Anzahl an Stimmen zu geben, die dieser dann nach eigener Einschätzung (auch gehäufelt) auf Szenarien verteilen kann.

5.3.3.4 Schritt 4: Zuordnung von Nutzwerten zu Szenarien

In diesem Schritt wird nun der Nutzenbegriff eingeführt. Dieser ist zunächst auf den Zielerreichungsgrad eines Szenarios bezogen. Er ist eine Kennzahl, die beschreibt, welcher Nutzen jeweils bei den in Schritt 2 eingeführten Leveln erreicht wird. Ein Wert von 100 markiert den maximalen Nutzen, ein Wert von 0 bedeutet, dass aus einer gewählten Alternative kein Nutzen bei einem Szenario erzielt werden kann.

Definition 31 (Level-bezogener Nutzengrad eines Szenarios). Der **Nutzengrad eines Szenarios** $s \in S$ sei über folgende Abbildung definiert:

$utilityScore(s, level) \mapsto score$

Wobei gilt: $level \in$ {worst, current, desired, best}.
Weiterhin gilt: $score \in \mathbb{N} : 0 \leq score \leq 100$.

Der Nutzengrad eines Szenarios ist eine Aussage darüber, welchen Einfluss eine bestimmte Architekturentscheidung auf ein einzelnes Szenario nehmen kann. Die vier Werte für die Level spannen den Rahmen auf, in dem sich eine konkrete Alternative mit ihrem Nutzeneinfluss bewegt. Die Differenz zwischen dem Nutzen nach Anwendung einer Architekturentscheidung und dem aktuellen Nutzen $utilityScore(s,$current$)$ kann positiv (d.h. die Alternative bewirkt eine Erhöhung des Nutzens) oder negativ (d.h. die Alternative führt zu einer Verschlechterung des Nutzens) sein. Die Bestimmung des Nutzengrads einer Architekturentscheidung erfolgt in den nächsten beiden Schritten.

5.3.3.5 Schritt 5: Entwicklung von Architekturstrategien und Bestimmung der erwarteten Auswirkungen auf die Qualitätsattribute

Die in der CBAM als Architekturstrategien bezeichneten Elemente entsprechen Architekturalternativen. Für jede zugehörige Architekturentscheidung wird nun zunächst bestimmt, bei welche Szenarien sie Einfluss auf den Zielerreichungsgrad ausübt, d.h. welches Szenario von der Umsetzung einer Alternative betroffen ist. Anschließend wird abgeschätzt, wie hoch der Zielerreichungsgrad für diese Alternative ausfallen wird.

> *Definition 32* (Auswirkungen einer Architekturentscheidung). Die **Auswirkungen einer Architekturentscheidung** $ae \in AE$ seien über folgende Abbildung definiert:
>
> $impactOn(ae) \mapsto S_{ae}$
>
> Wobei gilt: $S_{ae} \subset S$, d.h. S_{ae} ist die Menge all derjenigen Szenarien, deren Zielerreichungsgrad von der Umsetzung der Architekturentscheidung ae betroffen ist.

Für jedes Szenario aus S_{ae} muss nun abgeschätzt werden, welcher konkrete Zielerreichungsgrad sich bei einer Implementierung von ae ergibt.

> *Definition 33* (Architekturentscheidung-bezogener Zielerreichungsgrad eines Szenarios). Der **auf eine Architekturentscheidung** $ae \in AE$ **bezogene Zielerreichungsgrad eines Szenarios** $s \in S$ sei über folgende Abbildung definiert:
>
> $respGoal(s, ae) \mapsto responseGoal$
>
> $responseGoal$ ist wie bei den Level-bezogenen Zielerreichungsgraden eine – ggf. in Prosa verfasste – Beschreibung eines Zielerreichungsgrads.

Bei der Bestimmung dieses Zielerreichungsgrads kommt hochgradig die Expertise eines Software-Architekten zum Einsatz. Der Erreichungsgrad wird auf Basis von Erfahrung und Abschätzung auf Basis des Architekturentwurfs getroffen.

5.3.3.6 Schritt 6: Bestimmung der Nutzwertänderungen einzelner Architekturalternativen

Für die im vorangegangenen Schritt bestimmten Zielerreichungsgrade müssen nun entsprechende Nutzengrade bestimmt werden.

> *Definition 34* (Architekturentscheidung-bezogener Nutzengrad eines Szenarios). Der **auf eine Architekturentscheidung** $ae \in AE$ **bezogene Nutzengrad eines Szenarios** $s \in S$ sei über folgende Abbildung definiert:

$utilityScore(s, ae) \mapsto score$

Wobei folgendes gilt: $score \in \mathbb{N} : 0 \leq score \leq 100$.

Die Abbildung muss durch Interpolation aus den Level-bezogenen Zielerreichungsgraden, den Level-bezogenen Nutzengraden und den *ae* betreffenden Zielerreichungsgrad ermittelt werden. Dies wird in Abb. 5-5 veranschaulicht.

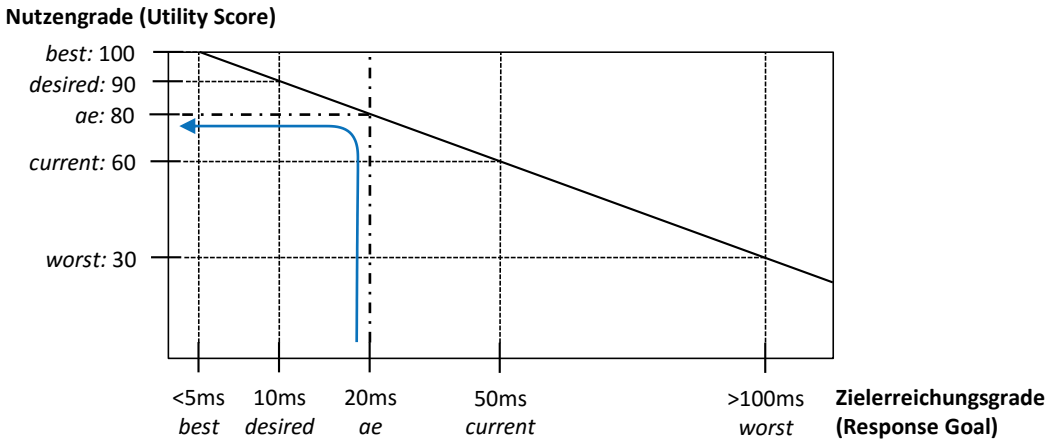

Abb. 5-5 Interpolation des Architekturentscheidung-bezogenen Nutzengrads

Die Abbildung zeigt ein Raster, an dessen horizontaler Achse die Zeilerreichungsgrade aus obigem Beispiel (siehe 5.3.3.2) aufgetragen sind. Hinzu kommt der für *ae* angenommene Wert für die Antwortzeit von 20ms. Auf der vertikalen Achse sind die den Level-bezogenen Zielerreichungsgraden zugehörigen Nutzengrade eingefügt. Es ist hier ein linearer Zusammenhang zwischen Zielerreichungsgrad und Nutzengrad, wiedergegeben durch die diagonal verlaufene Linie, erkennbar. Damit ist es nun möglich bereits aus diesem Raster einen sinnvollen Nutzengrad für *ae* abzulesen: Vom Schnittpunkt einer durch den Achsenwert 20ms verlaufenden senkrechten Geraden mit der Diagonalen aus kann das Lot auf die vertikale Achse gefällt werden. Dem Schnittpunkt mit der Achse entspricht in etwa der Nutzwert 80.

5.3.3.7 Schritt 7: Berechnung des Gesamtnutzens einer Architekturstrategie

Mit den nun vorliegenden Daten kann nun der Gesamtnutzen einer Architekturentscheidung ermittelt werden. Dieser ergibt sich aus der gewichteten Summe der pro betroffenen Szenario erzielten Nutzengradänderung. Die Gewichtung ergibt sich aus der Priorisierung der einzel-

nen Szenarien. Die Nutzengradänderung aus der Differenz zwischen dem entscheidungsbezogenen Nutzengrad und dem aktuellen Nutzengrad. Zur Berechnung kann somit folgende Formel angesetzt werden:

Definition 35 (Gesamtnutzen einer Architekturentscheidung). Der **Gesamtnutzen einer Architekturentscheidung** $ae \in AE$ kann über folgende Formel berechnet werden:

$$Utility(ae) = \sum_{s \in S_{ae}} votesScen(s) \cdot (respGoal(s, ae) - respGoal(s,\text{current}))$$

Damit kann man bereits ein erstes Ranking von Architekturentscheidungen aus einem Architekturentwurf aufstellen. Der prognostizierte Nutzengewinn ist nun für jede einzelne Alternative bekannt. Entsprechend lässt sich der Gesamtnutzen einer Architekturalternative als Summe der Nutzenwerte der enthaltenen Architekturentscheidungen berechnen.

$$Utility(aalt) = \sum_{vp \in VP} Utility(ae_{aalt}(vp))$$

5.3.3.8 Schritt 8: Auswahl der bestgeeigneten Architekturstrategie

Wie bereits erwähnt, wäre es ein naiver Ansatz, aus der Liste der Architekturentscheidungen ohne weitere Betrachtung diejenigen Alternativen auszuwählen, die im Nutzenranking oben stehen. Häufig ist ein hoher Nutzengewinn aber mit entsprechend hohen Aufwänden verbunden. Damit stellt sich die Frage, wie es bei den einzelnen Alternativen um das Aufwand-Nutzen-Verhältnis bestellt ist. Es wird also eine Return-on-Investment-Betrachtung benötigt. Hierzu muss für jede Architekturentscheidung ein Wert für den bei der Umsetzung zu erwartenden Aufwand abgeschätzt werden.

Definition 36 (Gesamtaufwand einer Architekturentscheidung). Der **Gesamtaufwand einer Architekturentscheidung** $ae \in AE$ sei über folgende Abbildung definiert:

$Cost(ae) \mapsto \mathbb{N}$.

Auch hier können die Gesamtaufwände für eine Architekturalternative über Aufsummieren der Aufwände der betreffenden Architekturentscheidungen berechnet werden.

$$Cost(aalt) = \sum_{vp \in VP} Cost(ae_{aalt}(vp))$$

Die Abbildungen zu Nutzen und Kosten können im Nutzenmodell zusammengefasst werden:

Definition 37 (Nutzenmodell) Gegeben sei die Menge aller Nutzenmodelle $UM = \{..., um_i, ...\}$. Ein auf einen bestimmten Architekturentwurf AD und ein bestimmtes Anforderungsmodell RM bezogenes **Nutzenmodell** $um_{AD,RM} \in UM$ ist ein Tupel

$$um_{AD,RM} = (Utility, Cost)$$

wobei für die einzelnen Elemente folgendes gilt:
$Utility$ ist eine Abbildung $(ae \rightarrow Utility(ae))$
$Cost$ ist eine Abbildung $(ae \rightarrow Cost(ae))$.

Die Werte für die Architekturentscheidungen basieren auf der Erfahrung der am Bewertungsprozess beteiligten Personen. Prinzipiell werden hier Werte benötigt, die eine Vergleichbarkeit von Architekturalternativen hinsichtlich der zu erwartenden Kosten herstellen. Mit diesen Aufwandswerten können nun die Werte für den Return-on-Investment für alle Architekturalternativen berechnet werden.

Definition 38 (Return-on-Investment einer Architekturentscheidung). Der **Return-on-Investment einer Architekturentscheidung** $ae \in AE$ sei über folgende Formel definiert:

$$ROI(ae) = \frac{Utility(ae)}{Cost(ae)}$$

Mit dieser Formel ist nun das Ziel erreicht, eine Vergleichbarkeit von Architekturalternativen nach ihrem Aufwand-Nutzen-Verhältnis herzustellen. Damit erhalten nicht notwendigerweise die Alternativen mit dem höchsten Nutzenzuwachs den Vorzug, sondern diejenigen, bei denen das Verhältnis zu den anfallenden Aufwänden am günstigsten ist. Dabei kann eine Alternative ae_1, die einen höheren Nutzwertgewinn als eine Alternative ae_2 nach sich zieht, beim Return-on-Investment durchaus schlechter abschneiden als ae_2, wenn die Realisierung von ae_2 zu einem Bruchteil der Kosten von ae_1 möglich ist.

Definition 39 (Return-on-Investment einer Architekturalternative). Der **Return-on-Investment einer Architekturalternative** $aalt \in Aalt$ sei über folgende Formel definiert:

$$ROI(aalt) = \frac{Utility(aalt)}{Cost(aalt)}$$

5.3.3.9 Schritt 9: Abgleich der Ergebnisse mit Erfahrungswerten

In einem letzten Bewertungsschritt werden die Ergebnisse einem „Reality Check“ unterzogen, d.h. die erzielten Werte werden von erfahrenen Prozessbeteiligten auf Plausibilität überprüft. Sofern sich in der intuitiven Einschätzung der Beteiligten grobe Abweichungen gegenüber den berechneten Werten ergeben (die zum Teil ja auf Annahmen und subjektiven Bewertungen basieren), können mehrere Maßnahmen ergriffen werden:

- Analyse der Hintergründe der Abweichungen, um Aussagen darüber abzuleiten, wo getroffene Annahmen und Einschätzungen im Widerspruch zu der Gesamteinschätzung bestehen.
- Rückkehr zu vorgelagerten Prozessschritten, um Annahmen und Bewertungen zu korrigieren.

Auch wenn sich die Beteiligten bei Ergreifen der zweiten Maßnahme dem Vorwurf aussetzen, den Prozess bzw. Werte solange anzupassen, bis das „gewünschte Ergebnis“ erzielt wird, können damit zumindest Annahmen und Bewertungen transparent gemacht werden.

Im Kontext von Cloud Computing ist dies häufig zu beobachten: Prozessbeteiligte sprechen sich zum Teil aus unbegründeten Motiven (z.B. „Bauchgefühl“) gegen den Einsatz von Speichertechnologien der Public Cloud aus. Nun sind solche Technologien im Vergleich zu vergleichbaren lokalen Lösungen extrem kostengünstig, was ihnen bei der Return-on-Investment zunächst einen großen Vorteil verschafft. Dies steht aber wiederum im Widerspruch zum „Wunschergebnis“ der genannten Stakeholder. Nun ist es durchaus legitim, zu einem früheren Bewertungsschritt zurückzukehren, und dort Korrekturen bei qualitativen Anforderungen (z.B. Zugriffslatenz aus dem lokalen Netzwerk), Priorisierungen (Bedeutung der eigenen Kontrolle über die Systeme) etc. vorzunehmen. Dies kann im Ergebnis dazu führen, dass der ROI-Wert der gewünschten Lösung (lokaler Betrieb der Speichersysteme) verbessert wird. Dieses Vorgehen ist durchaus vertretbar, sofern hierüber Konsens unter den Beteiligten herrscht und diese Entscheidungen und „Manipulationen“ dokumentiert werden. Denn damit wird dann beispielsweise im skizzierten Szenario erkennbar, dass eine teurere Lösung (lokaler Betrieb der Speichersysteme) bevorzugt wird, weil diese in als sehr wichtig priorisierten Kriterien deutlich besser abschneidet.

6 Zusammenführung der erweiterten Methoden zu einem Gesamtprozess

In den bisherigen Abschnitten wurden die Vorgehensmodelle zu Entwurf und Bewertung eines Architekturentwurfs einzeln betrachtet. Parallel wurden Beschreibungsmodelle definiert, mit denen Eingabe- und Ausgabeartefakte der einzelnen Methoden beschrieben werden können. Wo notwendig, wurde dabei darauf geachtet, dass Modelle bzw. Modellelemente bei Bedarf in mehreren Methoden verwendet werden können und sich die Definitionen in diesem Fall von Modell zu Modell nicht widersprechen. So kam beispielsweise der Architekturentwurf sowohl in der ADD als auch in der ATAM und CBAM zum Einsatz.

Darüber hinaus wurden Konsistenzregeln definiert, die eine Prüfung zulassen, ob Ausgabeartefakte einer Methode als Eingabeartefakte einer nachgelagerten Methode verwendet werden können.

6.1 Ableitung eines integrierten Gesamtprozesses

Grundsätzlich ist es nun möglich, die erwähnten Methoden ADD, ATAM und CBAM nacheinander durchzuführen und die dabei verwendeten Artefakte entsprechend formal zu beschreiben.

Abb. 6-1 zeigt, wie durch die Aneinanderreihung der drei Ausgangsmethoden ein integrierter Gesamtprozess gewinnen lässt. Aus dem Anforderungsmodell (RM) wird mittels der ADD ein Architekturentwurf (AD) gewonnen. Anschließend werden durch die ATAM ein Szenarienmodell (SM) und ein Risikomodell (RiskM) abgeleitet. Die ATAM erhält hierzu den aus der ADD gewonnenen Architekturentwurf sowie die zu Beginn formulierten qualitativen Anforderungen des Anforderungsmodells. Weitere Eingabe für die ATAM ist das Businessmodell (BM). Architekturbeschreibung, Business Driver und das Szenarienmodell wiederum sind Eingabe für die CBAM, die ein Nutzenmodell (UM) erzeugt.

Mit dieser Aneinanderreihung wird ein zentrales Ziel erreicht: Aus den formal definierten Anforderungen an ein Softwaresystem wird zunächst ein entsprechender Architekturentwurf (mit Variabilitäten) abgeleitet und dieser dann hinsichtlich bestehender Risiken und Kosten-Nutzen-Relationen analysiert. Damit kann also der bezüglich Risiken und Nutzen beste Architekturentwurf bestimmt werden.

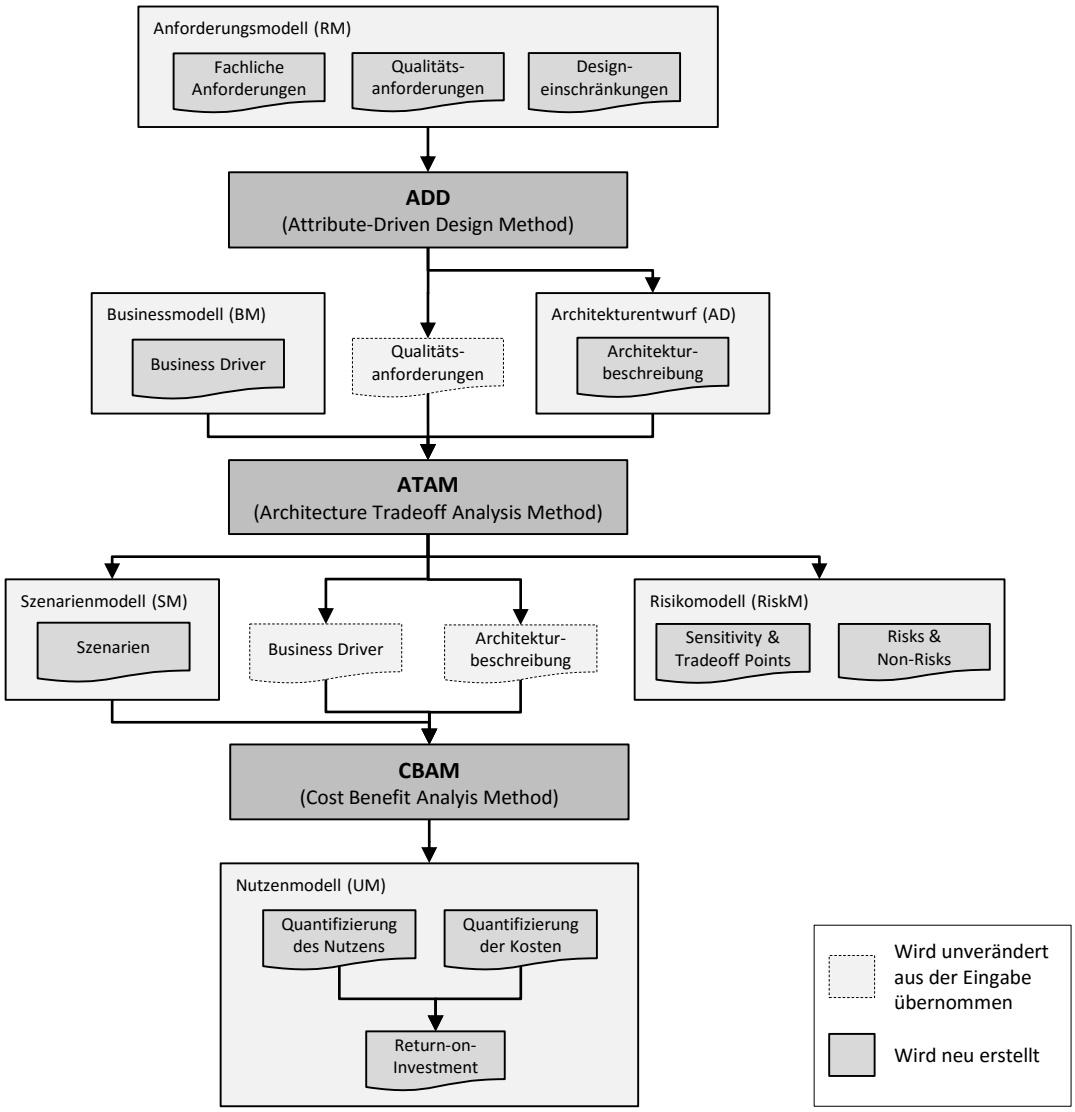

Abb. 6-1 Verkettung der bestehenden Architekturmethoden und deren Modelle

Dennoch ist dieser Ansatz nicht optimal. Er weist folgende Schwächen auf, die in den folgenden Abschnitten beseitigt werden sollen:

Einzelne Arbeitsschritte werden wiederholt ausgeführt. So werden Anforderungen mehrfach priorisiert. Dies könnte zusammengefasst einmalig geschehen, so dass die Priorisierungen dann für alle Folgeschritte zur Verfügung stehen. Daraus ergibt sich für den integrierten Gesamtprozess:

Anforderung 9 (Einmalige Ausführung von Entwurfs- und Bewertungsschritten). Bei der Durchführung des Gesamtprozesses sollen einzelne Arbeitsschritte nach Möglichkeit nur einmal durchgeführt werden.

Prozessbeteiligte müssen immer wieder punktuell hinzugezogen werden. Für die wiederholten Priorisierungen der Anforderungen müssen spätere Anwender, Auftraggeber etc. dediziert und nur für diese Arbeitsschritte einbezogen werden. Es würde für die Betroffenen eine Erleichterung darstellen, wenn sie ihre Aufgaben zusammengefasst erledigen könnten.

Anforderung 10 (Zusammenfassung von Arbeitsschritten einzelner Stakeholder). Die Einsätze einzelner Stakeholder sollten so weit wie möglich zusammengefasst werden, so dass deren Beteiligung sich auf die minimal notwendigen Zeiten beschränkt.

Plausibilitätsprüfungen und entsprechende Rückkopplungsschleifen sind ein wichtiges Element, um dem zu Beginn dieser Arbeit beschriebenen irrationalen Verhalten der Beteiligten (siehe Abschnitt 1.5) zu begegnen. Diese finden aber nur innerhalb der Methoden statt. So ist bei der ADD (siehe 4.2.3.1) und der CBAM (siehe 4.2.3.3) vorgesehen, zu vorangegangenen Prozessschritten zurückzukehren, um etwaige Abweichungen der erzielten Ergebnisse von erwarteten Ergebnissen durch Korrekturen der Ausgangsannahmen zu beseitigen. Solche Rückkopplungen werden auch methodenübergreifend benötigt. So ist es vorstellbar, dass ein Entwurf, der durch den Gesamtprozess als bester identifiziert wurde, nicht mit der Erfahrung des Architekten in Einklang gebracht werden kann. In diesem Fall kann es sinnvoll sein, zu Anforderungen, Priorisierungen etc. zurück zu gehen und diese anzupassen.

Anforderung 11 (Regelmäßige Plausibilitätsprüfungen). Während des Prozesses sollen regelmäßige Plausibilitätsprüfungen mit entsprechenden Rückkopplungsschleifen sicherstellen, dass Abweichungen von Erfahrungswerten möglichst frühzeitig erkannt und korrigiert werden können.

Bestimmte Arbeitsschritte wie die Präsentation der Vorgehensweise, Vorstellung von Ein- und Ausgabeartefakten werden methodenspezifisch durchgeführt. Bei der Zusammenführung der Methoden zu einem Gesamtprozess können einige Schritte zusammengefasst werden (z.B. die Präsentation der Vorgehensweise) andere Schritte entfallen (z.B. Vorstellung von Artefakten, da diese aus vorangegangenen Arbeitsschritten bereits bekannt sind).

Anforderung 12 (Minimierung der Anzahl an Arbeitsschritten). Die wiederholte Ausführung von Arbeitsschritten bzw. unnötige Ausführung von Zwischenschritten sollte weitestgehend vermieden werden.

Umgruppierungen von Aktivitäten und Einführung von Rückkopplungsschleifen sollen diese Schwächen nun beseitigen. Ein angepasster Gesamtprozess ist in Abb. 6-2 zu sehen.

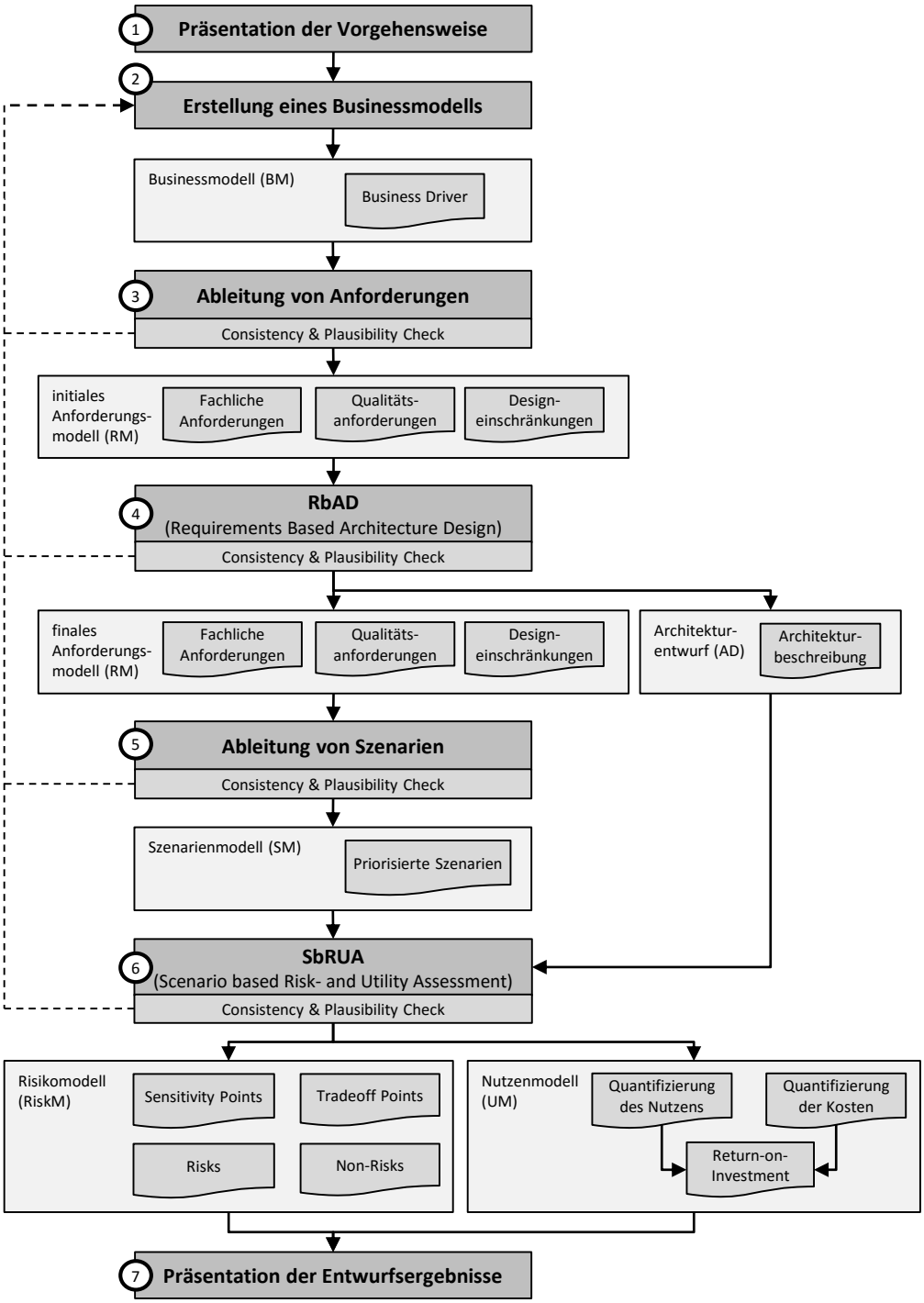

Abb. 6-2 Ablaufdiagramm des integrierten Gesamtprozesses

In diesem Ansatz wird nach einer allgemeinen Vorstellung der Vorgehensweise (1) zunächst ein Business Modell erstellt (2), aus diesem werden die initialen Anforderungen abgeleitet (3). Aus diesen Anforderungen wird dann der Architekturentwurf sowie das finale Anforderungsmodell abgeleitet (4). Dieses wird über Szenarien konkretisiert (5), auf deren Basis die Analyse und Bewertung erfolgt (6). Am Ende folgt eine Präsentation der Entwurfs- und Bewertungsergebnisse (7).

In den meisten dieser Schritte werden einzelne Schritte aus den betrachteten Ausgangsmethoden durchgeführt. Dabei wurde zum Teil die Reihenfolge verändert. Auch wurden (aufgrund redundanter Durchführung) nicht alle Schritte in den Gesamtprozess übernommen.

Schritt	**Aktivität**	**Schritte bestehender Methoden**
1	Präsentation der Vorgehensweise	keine
2	Erstellung eines Business Models	keine
3	Ableitung von Anforderungen	keine
4	Anforderungsbasierter Architekturentwurf	ADD, Schritt 2: Auswahl Systemelement
		ADD, Schritt 3: Identifikation architekturrelevanter Anford.
		ADD, Schritt 4: Auswahl Designkonzept
		ADD, Schritt 5: Instanziierung von Architekturelementen
		ADD, Schritt 6: Definition Schnittstellen
		ADD, Schritt 7: Verfeinerung von Anforderungen
5	Ableitung von Szenarien	ATAM, Schritt 5: Erstellung eines QA-Baums
		ATAM, Schritt 7: Priorisierung von Szenarien
		CBAM, Schritt 1: Sortierung von Szenarien
		CBAM, Schritt 2: Verfeinerung von Szenarien
		CBAM, Schritt 3: Priorisierung von Szenarien
6	Szenariobasierte Risiko- und Nutzenbewertung	ATAM, Schritt 6: Analyse von Architekturansätzen
		CBAM, Schritt 4: Nutzenwerte zu Szenarien
		CBAM, Schritt 5: Auswirkungen Architekturansätze auf QA
		CBAM, Schritt 6: Bestimmung der Gesamtnutzenwerte
		CBAM, Schritt 7: Berechnung des Gesamtnutzens
7	Präsentation der Entwurfs- und Bewertungsergebnisse	keine

Tabelle 6-1: Durchführung von Schritten der Ausgangsmethoden im Gesamtprozess

Tabelle 6-1 gibt einen Überblick über die Übertragung einzelner Schritte in den Gesamtprozess. Sie zeigt, dass die ADD zum großen Teil in Schritt 4 des Gesamtprozesses aufgeht, in dem der Architekturentwurf erstellt wird. Schritt 1 der ADD entfällt, da bereits in den ersten Schritten des Gesamtprozesses die Vollständigkeit des Anforderungsmodells sichergestellt wird. Die Schritte aus ATAM und CBAM, bei denen im Wesentlichen Anforderungen erstellt, verfeinert und priorisiert werden, finden sich in Schritt 5. Der folgende Schritt 6 umfasst die Schritte aus ATAM und CBAM, bei denen der Architekturentwurf bewertet wird. Die Schritte 1 bis 3 der ATAM entfallen, da durch die Einbeziehung aller Stakeholder von Beginn an die Präsentation bisher erzielter Ergebnisse nicht erforderlich ist. Schritt 4 der ATAM entfällt, da Architekturansätze durch Anwendungs- und Variabilitätsmodell beschrieben sind. Schritt 9 der ATAM entfällt, da die Präsentation der Ergebnisse am Ende des Gesamtprozesses erfolgt. Aus der CBAM entfällt Schritt 8, und Schritt 9 geht in der Plausibilitätsprüfung im Bewertungsschritt des Gesamtprozesses auf. In den folgenden Abschnitten werden die einzelnen Schritte des Gesamtprozesses und entsprechende Verweise zu den Ausgangsmethoden genauer beschrieben.

6.2 Schritt 1: Präsentation der Vorgehensweise

Zu Beginn des Gesamtvorhabens sollte die Vorgehensweise allen Stakeholdern präsentiert werden. Ziel ist hierbei, ein gemeinsames Verständnis zu den Verantwortlichkeiten, den Zeitpunkten der Mitarbeit und den Aufgaben in den einzelnen Prozessschritten zu entwickeln. Die folgende Tabelle listet auf, welche Stakeholder an welchen Bearbeitungsschritten beteiligt sind.

Schritt	**Aktivität**	**Erstellte Modelle**	**Beteiligte Stakeholder**
1	Präsentation der Vorgehensweise		Alle
2	Erstellung eines Business Models	Businessmodell	Alle
3	Ableitung von Anforderungen	(initiales) Anforderungsmodell	Alle
4	Anforderungsbasierter Architekturentwurf	Architekturentwurf und (finales) Anforderungsmodell	Architekten
5	Ableitung von Szenarien	Szenarienmodell	Alle
6	Szenariobasierte Risiko- und Nutzenbewertung	Risikomodell und Nutzenmodell	Evaluierungsteam, Architekten
7	Präsentation der Entwurfs- und Bewertungsergebnisse		Alle

Tabelle 6-2: Beteiligung von Stakeholdern an Entwurfs- und Bewertungsschritten

Die Tabelle zeigt, welche Stakeholder in den einzelnen Schritten erforderlich sind. Selbstverständlich können alle Stakeholder durchgehend mitwirken. Die Erfahrung zeigt allerdings, dass es sinnvoller ist, insbesondere den eigentlichen Entwurf und die Bewertung auf die Architekten bzw. das Evaluierungsteam zu beschränken, um ein unabhängiges Arbeiten in diesen Schritten gewährleisten zu können und Störungen zu vermeiden. Letztere können entstehen, wenn während Entwurf und Bewertung Anforderungen und Szenarien durch Nutzer nachgereicht werden. Dies wäre aus pragmatischer Sicht wünschenswert. Da ein Ziel des Gesamtprozesses aber die Offenlegung und Dokumentation zunächst „unausgesprochener" Anforderungen ist, sollte hier eine strikte Trennung zwischen der Aufstellung von Anforderungen und den weiteren Schritten erfolgen. Entwurf und Bewertung erhalten damit klare Eingaben. Führt der Entwurf dann beispielsweise zu einer Public-Cloud-basierten Ausführungsumgebung als bestbewertete Option, die wiederum von einzelnen Stakeholdern abgelehnt wird, müssen im Rahmen der Konsistenz- und Plausibilitätsprüfung entsprechende Anforderungen (z.B. die Entwurfseinschränkung „die Anwendung muss lokal betrieben werden") nachformuliert und entsprechend klar priorisiert werden.

6.3 Schritt 2: Erstellung eines Business Models

Nachdem die Vorgehensweise allen Stakeholdern vorgestellt wurde, erfolgt die Erstellung eines Business Models. An diesem zweiten Prozessschritt sind ebenfalls alle Stakeholder, d.h. alle Personen, die von der Erstellung des Systems betroffen sind, beteiligt. Dabei ist es wichtig, dass tatsächlich alle Betroffenen gehört werden. Ziel ist es, die grundsätzlichen Anforderungen, Wünsche und Ziele, die mit dem zu erstellenden System verbunden sind, zusammenzutragen. Daraus ergibt sich der Rahmen, innerhalb dessen die Systementwicklung, zu der eine Architekturbeschreibung erstellt werden soll, ablaufen wird. Zu diesem Rahmen gehören allgemeine Funktionalitäten des Systems (die im späteren Verlauf in Form formaler Anforderungen weiter ausformuliert werden), relevante Einschränkungen (die bestimmte Entwurfsentscheidungen vorwegnehmen) und Geschäftsziele, die mit der Erstellung des Systems unterstützt oder erreicht werden sollen.

Dieser Rahmen wird über ein Business Model beschrieben. Dieses enthält eine Menge von Business Drivern, die – in Prosa formuliert – die einzelnen Aspekte des Anforderungsrahmens enthält. Schritt 2 lässt sich somit wie in Abb. 6-3 gezeigt beschreiben.

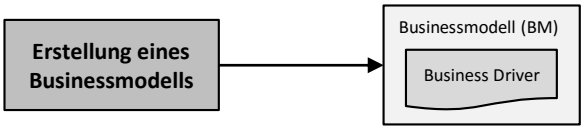

Abb. 6-3 Erstellung eines Businessmodells (Schritt 2)

Für die Zusammenstellung der Business Driver müssen vom Projektteam folgende Fragen beantwortet werden:

- Welches sind die grundsätzlichen Funktionalitäten des Systems?
- Welche technischen, ökonomischen und politischen Entscheidungen müssen als gesetzt angesehen werden?
- Welche Ziele sollen mit dem System verfolgt bzw. erreicht werden?

Die Antworten auf diese Fragen sind Motivation für alle weiteren Schritte und Entwurfsartefakte. Oder anders herum: Anforderungen, Designeinschränkungen etc. müssen jeweils mit mindestens einem Business Driver in Beziehung stehen. Sofern Business Driver vergessen wurden und dies in nachfolgenden Schritten erkannt wird (weil für ein Entwurfsartefakt kein Business Driver identifiziert werden kann), können durch die im Gesamtprozess vorgesehenen Rückkopplungsschleifen Business Driver auch nachträglich eingefügt werden. Aus den gewonnenen Business Drivern, den Stakeholdern und den jeweiligen Beziehungen zwischen Business Drivern und Stakeholdern lässt sich nun das Business Model erstellen. Dabei muss auf Einhaltung von Regel 4 (Jeder Business Driver hat einen Stakeholder) geachtet werden. D.h. es darf keinen Business Driver geben, der nicht durch einen Stakeholder vertreten wird. Dies erlaubt es später, den Ursprung bzw. den Verantwortlichen für einzelne Anforderungen zu ermitteln.

6.4 Schritt 3: Ableitung und Priorisierung der Anforderungen

Aus dem Businessmodell wird in diesem Schritt, wie in Abb. 6-4 skizziert, das initiale Anforderungsmodell abgeleitet, welches zentrale Eingabe für den anforderungsbasierten Architekturentwurf ist. In diesem initialen Modell sind alle Anforderungen dem Gesamtsystem zugewiesen. Erst im späteren Architekturentwurf werden die Anforderungen auf einzelne Architekturkomponenten heruntergebrochen und einzelnen Komponenten zugewiesen.

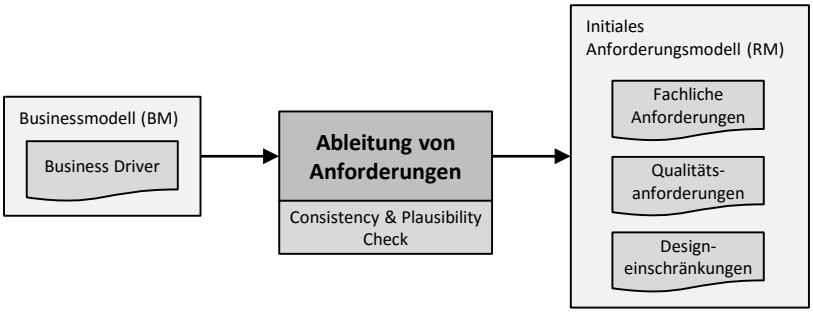

Abb. 6-4 Ableitung von Anforderungen (Schritt 3)

6.4.1 Schritt 3.1: Ableitung von Anforderungen

In diesem Schritt wird auf Basis des Business Models ein Anforderungsmodell erstellt. Zunächst werden für jeden einzelnen Business Driver konkrete Anforderungen abgeleitet und entsprechend im Anforderungsmodell die Beziehungen zum Business Model hergestellt.

```
// initialisiere eine leere Liste für fachliche Anforderungen,
// qualitativen Anforderungen und Designeinschränkungen
var FR = new List<fr>();
var QR = new List<qr>();
var DC = new List<dc>();

// erstelle entsprechend leere Abbildungen für die Beziehungen
// zwischen Anforderungen und Business Drivern sowie für die
// Priorisierungen
var freqBD = new Dictionary<fr, List<bd>>();
var freqPrio = new Dictionary<fr, Priority>();
var qreqBD = new Dictionary<qr, List<bd>>();
var qreqPrio = new Dictionary<qr, Priority>();
var dcBD = new Dictionary<dc, List<bd>>();

// für jeden Business Driver
foreach( var bdrv in BD )
{
   // erstelle konkrete Anforderungen für den betreffenden Business Driver
   List<req> requirements = getReqForBd( bdrv );

   // Fehlermeldung, wenn es keine Anforderungen für einen Business
   // Driver gibt
   if ( requirements.Count() == 0 )
      throw new RequirementsForBusinessDriverMissingException();

   // nimm jede dieser Anforderungen in die entsprechende
   // Anforderungsmenge auf und füge eine Abbildung zum Business
   // Driver in das Anforderungsmodell ein
   foreach( var req in requirements )
   {
      switch( typeof( req ) )
      {
```

```
            case fr:
               FR.Add( (fr)req );
               freqBD.TryGetValue( fr ).Add( bdrv );
               break;
            case qr:
               QR.Add( (qr)req );
               qreqBD.TryGetValue( qr ).Add( bdrv );
               break;
            case dc:
               DC.Add( (dc)req );
               dcBD.TryGetValue( dc ).Add( bdrv );
               break;
         }
      }
   }

   // erstelle aus den einzelnen Elementen ein Anforderungsmodell
   var RM =
      new RequirementsModel( FR, QR, DC, freqPrio, freqBD, ↪
         qreqPrio, qreqBD, dcBD );
```

Listing 6-1: Herleitung eines Anforderungsmodells

6.4.2 Schritt 3.2: Priorisierung von Anforderungen

Anschließend werden die einzelnen Anforderungen priorisiert. Sowohl an der Aufstellung der Anforderungen als auch an der Priorisierung ist das gesamte Projektteam beteiligt. Die Priorisierung der Qualitätsattribute folgt dabei den gleichen Anforderungen wie bereits in Definition 30 für die Priorisierung von Szenarien festgelegt wurde. D.h. solange diese Anforderungen erfüllt sind, kann eine beliebige konkrete Priorisierung vorgenommen werden. Beispielsweise können jedem beteiligten Stakeholder eine bestimmte Anzahl von Abstimmungspunkten gegeben werden, die dieser dann nach eigenem Ermessen (auch gehäufelt) auf die einzelnen Anforderungen verteilen kann. Die Summe der Abstimmungspunkte einer fachlichen oder qualitativen Anforderung wird dann entsprechend in der betreffenden Abbildung des Anforderungsmodells hinterlegt.

```
void prioritizeRequirements( RM )
{
   // für jede fachliche Anforderung des Anforderungsmodells
   foreach( var fr in RM.FR )
   {
      // priorisiere die Anforderung
      var priority = getPriority( fr );

      // schreibe die Priorisierung in das Anforderungsmodell
      RM.freqPrio.Add( fr, priority );
   }

   // für jede Qualitätsanforderung des Anforderungsmodells
   foreach( var qr in RM.QR )
   {
```

```
        // priorisiere die Anforderung
        var priority = getPriority( qr );

        // schreibe die Priorisierung in das Anforderungsmodell
        RM.qreqPrio.Add( qr, priority );
    }
}
```

Listing 6-2: Priorisierung von Anforderungen

6.4.3 Schritt 3.3: Konsistenz- und Plausibilitätsprüfung

Bei der Erstellung des Anforderungsmodells kann es passieren, dass für einzelne Anforderungen kein Business Driver gefunden werden kann, zu dem eine Abbildung definiert werden soll. Sofern der für die Anforderung verantwortliche Stakeholder auch an der Erstellung des Business Modells beteiligt war, ist dies ein Hinweis auf (bewusst oder unbewusst) unausgesprochene oder angenommene Rahmenbedingungen für das zu erstellende System.

Um diese Rahmenbedingungen offenzulegen, kann am Ende der Erstellung des Anforderungsmodells eine Konsistenz- und Plausibilitätsprüfung erfolgen. Im Rahmen dieser Prüfung kann das Projektteam den Bedarf neu zu formulierender Business Driver identifizieren. Ein Beispiel: häufig werden bei der Zusammenstellung von Anforderungen aus technischer Sicht bereits bestimmte Produkte (z.B. „Verwendung einer Open-Source-Datenbank“) gefordert, ohne dass dies durch einen zuvor erstellten Business Driver (z.B. „das Projekt unterstützt die Open-Source-Strategie im Unternehmen“) gedeckt wäre. Oft werden auch Entwurfsentscheidungen anderer Art (z.B. „Datenspeicherung im lokalen Rechenzentrum“) ohne entsprechenden Business Driver (z.B. „das Projekt muss entsprechende Compliance-Regeln zur Datenspeicherung einhalten“).

Erfahrene Projektmitglieder können darüber hinaus die bis dahin erzielten Modelle auf Plausibilität, d.h. Stimmigkeit zu ihren Erfahrungen, prüfen. Wenn beispielsweise die Menge der Entwurfseinschränkungen leer ist, könnte gezielt nachgefragt werden, ob nicht doch bestimmte Technologien, Frameworks oder Produkte gesetzt und diese Einschränkungen durch einen passenden Business Driver motiviert sind.

Ggf. muss daraufhin eine Rückkopplungsschleife gedreht werden und fehlende Business-Driver nachformuliert werden. Im Gesamtprozess bedeutet dies, dass ein Rücksprung zu Schritt 2 (siehe Abschnitt 6.3) erfolgt.

6.5 Schritt 4: Entwurf der Softwarearchitektur

Nachdem der Rahmen über Business Driver und initiale, formale Anforderungen ausreichend beschrieben ist, erfolgt nun der Entwurf der Softwarearchitektur. Dieser Schritt entspricht im

Wesentlichen der Attribute Driven Design Method (ADD). Ergebnis ist ein Architekturentwurf mit Anwendungs- und Variabilitätsmodell und entsprechende Abbildungen, die Bezüge zu den jeweiligen Anforderungen herstellen (siehe Abb. 6-5).

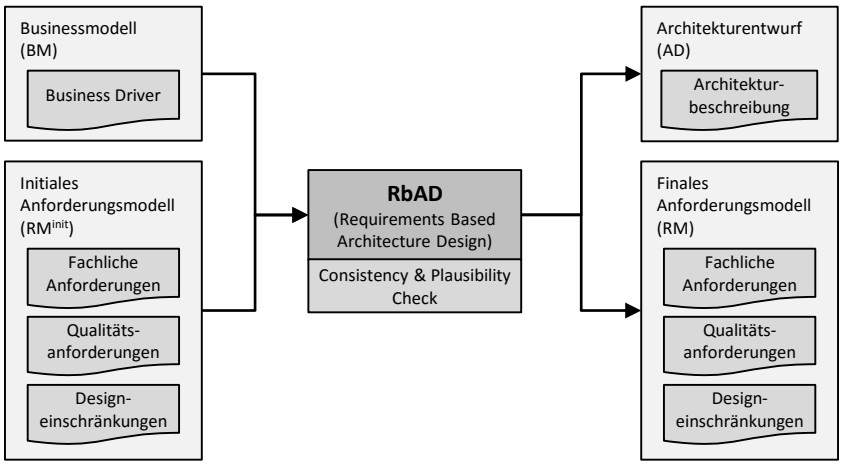

Abb. 6-5 Anforderungsbasierter Architekturentwurf (Schritt 4)

6.5.1 Schritt 4.1: Auswahl eines Systemelements, das zerlegt werden soll

Auf Basis des bis dahin erstellten Architekturentwurfs (in der ersten Iteration der ADD ist dies das Gesamtsystem als einzige Komponente) und den Anforderungen wird aus der Menge der für eine weitere Zerlegung auswählbaren Komponenten (siehe Regel 3) ein Element ausgewählt, das zerlegt werden soll. Dieser Schritt entspricht Schritt 2 der ADD (siehe 5.1.3.2).

6.5.2 Schritt 4.2: Identifikation möglicher architekturrelevanter Anforderungen

Aus der Menge der Anforderungen des initialen Anforderungsmodells werden diejenigen Elemente ermittelt, die für die Zerlegung der ausgewählten Komponente relevant sind. Aus dieser Auswahl werden wiederum diejenigen fachlichen und qualitativen Anforderungen bestimmt, die zuvor als am wichtigsten eingestuft wurden. Zusammen mit den für die Komponente gültigen Designeinschränkungen bilden diese entsprechend Definition 12 die Menge der architekturrelevanten Anforderungen der Komponente. Dieses Vorgehen entspricht Schritt 3 der ADD. Weitere Einzelheiten dazu finden sich in Abschnitt 5.1.3.3.

6.5.3 Schritt 4.3: Auswahl eines Designkonzepts, das die architekturrelevanten Anforderungen erfüllt

Es folgt für die zu zerlegende Komponente eine Auswahl eines Designkonzepts. Dieses muss alle architekturrelevanten Anforderungen erfüllen. Der Schritt ist damit identisch zu Schritt 4 der ADD. Wie bereits in Abschnitt 5.1.3.4 erläutert, ist hier die Expertise des Software-Architekten gefragt, der auf Basis seiner Erfahrung eine Auswahl eines (oder mehrerer) Designkonzepte treffen muss.

6.5.4 Schritt 4.4: Instanziierung der Architekturelemente und Zuordnung der Anforderungen zu Elementen

Es folgen Aktivitäten entsprechend Schritt 5 der ADD (siehe 5.1.3.5): gemäß des ausgewählten Designkonzepts wird die Komponente in Sub-Komponenten zerlegt. Dabei werden die bis dahin erstellten Anwendungs- und Variabilitätsmodelle entsprechend erweitert, d.h. die Komponente gemäß Definition 13 verfeinert und neue temporäre Abbildungen zu Anforderungen erstellt.

6.5.5 Schritt 4.5: Definition der Schnittstellen der instanziierten Elemente

Dieser Schritt entspricht Schritt 6 der ADD (siehe 5.1.3.6). Abhängig vom gewählten Designkonzept müssen noch weitere Änderungen und Erweiterungen im Anwendungs- und Variabilitätsmodell vorgenommen werden. Auch hier ist die Expertise des Software-Architekten von zentraler Bedeutung, da dieser Schritt weniger formalen Regeln unterliegt als mehr vom Entwurfsgeschick des Softwarearchitekten.

6.5.6 Schritt 4.6: Verifikation und Verfeinerung der Anforderungen und Umwandlung in Einschränkungen für die instanziierten Elemente

Nun erfolgt noch eine Anpassung der ursprünglichen Anforderungen an konkretere Anforderungen, die an die neu eingeführten Systemelemente angepasst sind und diesen zugeordnet werde. Detailliert ist dieser Schritt, der Schritt 7 der ADD entspricht, in 5.1.3.7 beschrieben. In diesem Schritt wird nach und nach das finale Anforderungsmodell erzeugt, welches dann Basis für die späteren Analyse- und Bewertungsschritte ist.

6.5.7 Schritt 4.7: Konsistenz- und Plausibilitätsprüfung

Nach Abschluss der Entwurfsphase erfolgt eine weitere Konsistenz- und Plausibilitätsprüfung, bei der der Architekturentwurf dahingehend betrachtet wird, ob er grundsätzlich mit den Erwartungen der Prozessbeteiligten übereinstimmt. Hierbei treten Abweichungen von

den Erwartungen häufig weniger im Detail (z.B. konkreten Implementierungsdetails) als mehr auf konzeptioneller Ebene auf. Oft ist wieder das Fehlen von Anforderungen (oder Business Drivern) ursächlich. So könnte der Architekturentwurf beispielsweise eine Datenspeicherung in der Public Cloud vorsehen, da dies unter den gegebenen Rahmenbedingungen als geeignete Lösung erschien. Aus Compliance-Gesichtspunkten (z.B. rechtliche Anforderung zur Datenspeicherung in eigenen Rechenzentrum) könnte dies aber ausgeschlossen sein, was im Anforderungsmodell aber nicht als Entwurfseinschränkung formuliert wurde. Dieser Widerspruch muss dann durch eine Rückkopplung, d.h. Nach-Formulierung im Anforderungsmodell in Schritt 3 und entsprechende Anpassung des Architekturentwurfs (z.B. durch erneute (ggf. teilweise) Durchführung der vorangegangenen Entwurfsschritte) aufgelöst werden.

6.6 Schritt 5: Ableitung und Priorisierung von Szenarien

Bei der Ableitung von Szenarien wird aus den finalen Anforderungen zunächst ein Szenarienmodell abgeleitet. Anschließend werden über entsprechende Abbildungen vom Szenarienmodell Bezüge zum Anforderungsmodell hergestellt (siehe Abb. 6-6).

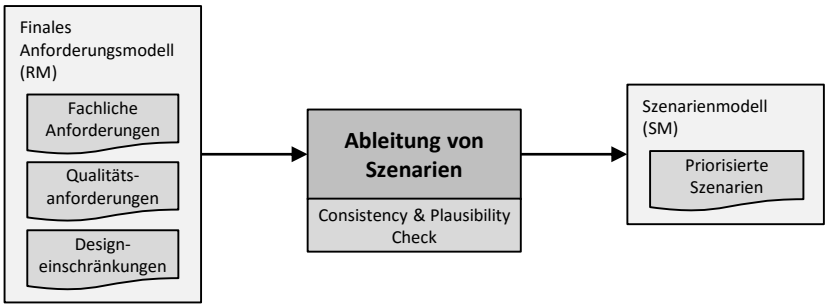

Abb. 6-6 Ableitung von Szenarien (Schritt 5)

6.6.1 Schritt 5.1: Erstellung eines Qualitätsattribute-Baums

Dieser Schritt entspricht Schritt 5 der ATAM, welcher wiederum in Abschnitt 5.2.3.5 ausführlich beschrieben ist. Die (zum Teil abstrakten) Anforderungen des finalen Anforderungsmodells werden schrittweise verfeinert und durch Szenarien konkretisiert. Die Verfeinerungen werden durch die Verästelung des Qualitätsattribute-Baums repräsentiert. Die Szenarien sind die Blätter des Baumes. Diese enthalten entsprechende Referenzen zu den Anforderungen, d.h. es kann über das Szenarienmodell zu einem bestimmten Szenario die zugehörige Anforderung bestimmt werden.

6.6.2 Schritt 5.2: Verfeinerung der Szenarien

Die erstellten Szenarien werden nun verfeinert, d.h. es werden für jedes Szenario konkrete Zielerreichungsgrade („worst“, „current“, „desired“ und „best“) entsprechend Definition 29 formuliert. Damit wird der Rahmen für die spätere Nutzenbetrachtung einzelner Architekturalternativen aufgespannt. Dieser Verfeinerungsschritt wird entsprechend Schritt 2 der CBAM durchgeführt. Dieser ist in Abschnitt 5.3.3.2 ausführlich beschrieben.

6.6.3 Schritt 5.3: Priorisierung von Szenarien

Bei einer einfachen Verkettung der Ausgangsmethoden erfolgt eine Priorisierung von Szenarien zum einen in der ATAM in Schritt 7 (siehe 5.2.3.7) und zum anderen in der CBAM in Schritt 3 (siehe 5.3.3.3). Diese Aktivitäten werden für den Gesamtprozess zu einem Schritt zusammengefasst. Analog der Priorisierung von Anforderungen gilt auch für die Priorisierung der Szenarien, dass diese nach verschiedenen Ansätzen vorgenommen werden kann, solange die in Definition 30 für die Priorisierung von Szenarien formulierten Anforderungen erfüllt sind. So können jedem beteiligten Stakeholder eine bestimmte Anzahl von Abstimmungspunkten gegeben werden, die dieser dann (auch gehäufelt) auf die einzelnen Szenarien verteilen kann. Die Summe der Abstimmungspunkte wird dann für jedes Szenario in der betreffenden Abbildung des Szenarienmodells hinterlegt.

6.6.4 Schritt 5.4: Konsistenz- und Plausibilitätsprüfung

Nachdem Szenarien dediziert aus zuvor formulierten Anforderungen abgeleitet werden, sollte aus formalen Gründen eigentlich keine Konsistenz- und Plausibilitätsprüfung erforderlich sein. Da Szenarien aber greifbarer als die (ggf. abstrakt) formulierten Anforderungen beschrieben werden, kann es passieren, dass im Rahmen dieser Prüfung das Fehlen von (eigentlich erwarteten) Szenarien durch einzelne Stakeholder einfacher bemerkt wird als zuvor das Fehlen formaler Anforderungen. An dieser Stelle ist es deshalb möglich, Szenarien auch ohne Anforderungen nachzureichen. Die zugehörigen Anforderungen müssen dann im Rahmen der Rückkopplungsschleife im Anforderungsmodell nachgetragen werden. Sofern sich daraus dann Anforderungen ergeben, die keinem Business Driver zugeordnet werden können, muss die Rückkopplung weiter gehen und auch entsprechende Business Driver nachformuliert werden.

6.7 Schritt 6: Bewertung der Softwarearchitektur

In diesem Abschnitt werden, wie in Abb. 6-7 skizziert, die zentralen Bewertungsschritte aus ATAM und CBAM zusammengefasst. Aus dem Szenarienmodell und dem Architekturent-

wurf werden ein Risikomodell sowie ein Nutzenmodell abgeleitet, welche Auskunft über Risiken und Nutzen des Architekturentwurfs und darin enthaltenen Entwurfsentscheidungen geben.

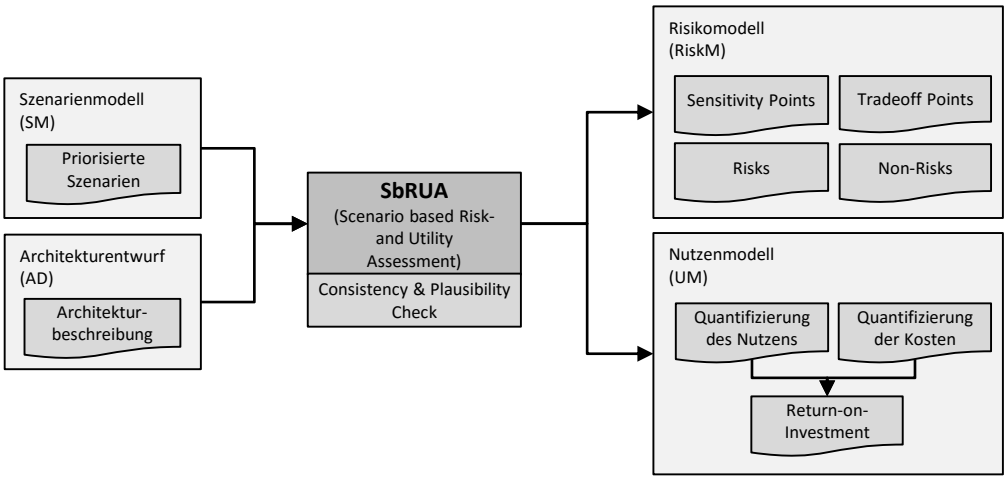

Abb. 6-7 Szenariobasierte Risiko- und Nutzenbewertung (Schritt 6)

6.7.1 Schritt 6.1: Bewertung der Risiken

In diesem Schritt erfolgen die Aktivitäten, die bei einer isolierten Durchführung der ATAM dort in Schritt 6 erfolgen (siehe 5.2.3.6). Schritt 8 der ATAM wird nicht benötigt, da Szenarien ohne konkrete Anforderungen (diese würden in Schritt 8 ja analysiert) durch die Konsistenz- und Plausibilisierungsprüfung in Schritt 4.4 des Gesamtprozesses ausgeschlossen werden.

Für die hoch priorisierten Szenarien werden die betroffenen qualitativen Anforderungen ermittelt und für diese die relevanten Architekturkomponenten und Variabilitätspunkte. Für diese wird dann bestimmt, ob es sich um Sensitivity Points, Tradeoff Points, Risks oder Non-Risks handelt. Aus den Ergebnissen (vier Listen) wird dann ein Risikomodell entsprechend Definition 27 erstellt. Für weitere Informationen zum konkreten Vorgehen siehe 5.2.3.8.

6.7.2 Schritt 6.2: Bewertung des Return-on-Investment

Zur Bewertung des Return-on-Investment werden die zentralen Bewertungsschritte der CBAM durchgeführt. Ergebnis ist dann ein Nutzenmodell, welches die Bewertung einzelner Architekturalternativen enthält.

6.7.2.1 Schritt 6.2.1: Zuordnung von Nutzwerten zu Szenarien

Analog Schritt 4 der CBAM werden zu den zuvor formulierten Szenarien Nutzwerte zwischen 0 und 100 zugeordnet (siehe 5.3.3.4). Diese treffen eine Aussage darüber, welche Auswirkungen eine Architekturalternative auf ein einzelnes Szenario haben kann. Je enger die Nutzengrade für die einzelnen Zielerreichungsgrade (worst, current, desired, best) liegen, desto geringer ist die Auswirkung einer Alternative.

6.7.2.2 Schritt 6.2.2: Bestimmung der erwarteten Auswirkungen auf Szenarien

Für Variabilitätspunkte (diese entsprechen Architekturansätzen im Sinne von Schritt 5 der CBAM) werden nun zunächst entsprechend Definition 32 die Auswirkungen ihrer Architekturalternativen, d.h. welche Szenarien von den Alternativen betroffen sind, ermittelt.

Anschließend wird abgeschätzt, wie sich einzelne konkrete Alternativen auf den Nutzengrad der Szenarien auswirken. Es wird also der Architekturentscheidung-bezogene Zielerreichungsgrad eines Szenarios gemäß Definition 33 bestimmt. In Abschnitt 5.3.3.5 wird dies detailliert beschrieben.

6.7.2.3 Schritt 6.2.3: Bestimmung der Nutzwertänderungen einzelner Architekturalternativen

Für den im letzten Schritt ermittelten Zielerreichungsgrad wird nun durch Interpolation der entsprechende Nutzengrad gemäß Definition 34 ermittelt. Das Vorgehen hierzu entspricht Schritt 6 der CBAM und wird in Abschnitt 5.3.3.6 genauer beschrieben.

6.7.2.4 Schritt 6.2.4: Berechnung des Gesamtnutzens einer Architekturstrategie

Abschließend wird nun noch der Gesamtnutzen der Architekturentscheidungen nach Definition 35 berechnet. Dazu werden die Nutzwertänderungen durch einzelne Architekturentscheidungen unter Berücksichtigung der Gewichtung der betroffenen Szenarien aufsummiert. Der Schritt wird analog Schritt 7 der CBAM durchgeführt und wird in Abschnitt 5.3.3.7 genauer beschrieben.

6.7.3 Schritt 6.3: Konsistenz- und Plausibilitätsprüfung

Auch am Ende der Architekturbewertung steht eine Konsistenz- und Plausibilitätsprüfung. Werden hierbei Abweichungen zwischen den erwarteten und erzielten Ergebnissen aufgedeckt können die Kosten für mögliche Korrekturen (z.B. Rücksprung zu vorgelagerten Prozessschritten und Nachtrag von Anforderungen) jedoch beträchtlich sein, da hierzu größere Teile des Gesamtprozesses wiederholt durchlaufen werden müssen.

6.8 Schritt 7: Präsentation der Entwurfs- und Bewertungsergebnisse

Sofern die letzte Konsistenz- und Plausibilitätsprüfung keinen weiteren Korrekturbedarf ergeben hat, können am abschließenden Schritt die Entwurfs- und Bewertungsergebnisse dem Projektteam, d.h. allen Stakeholdern präsentiert werden. Die Ergebnisse umfassen dabei folgende Artefakte:

- *Alle für den Entwurf verwendeten Eingangsmodelle.*
  Diese Modelle umfassen das Business Modell, das Anforderungsmodell und das Szenarienmodell.

- *Den Architekturentwurf.*
  Dieser enthält das Anwendungsmodell, das Variabilitätsmodell sowie Implementierungsdetails.

- *Die Bewertungsmodelle.*
  Hierunter fallen das Risikomodell und das Nutzenmodell.

- *Protokolle der einzelnen Konsistenz- und Plausibilitätsprüfungen.*
  Diese Protokolle geben Aufschluss darüber, ob und wo Anforderungen erst zu späteren Prozessphasen nachgeliefert wurden. Diese Information kann bei etwaigen Folgeprojekten als Prüfliste dienen, die dort in frühen Prozessphasen bei Vollständigkeitsprüfungen herangezogen werden kann.

Damit ist der Prozess zu Entwurf und Bewertung einer Softwarearchitektur abgeschlossen. Die gewonnenen Artefakte können nun wiederum als Ausgangsmaterial für folgende Aktivitäten (Implementierung, Test, Betrieb etc.) verwendet werden.

7 Anwendung des Leitfadens auf Cloud-Anwendungen

Der in Kapitel 6 beschriebene Gesamtprozess soll nun in den folgenden Abschnitten anhand mehrerer Beispielprojekte Praxistests unterzogen werden. Ziel ist dabei, den Prozess auf Erfüllung der an ihn gestellten Anforderungen zu überprüfen. Zunächst soll dabei in einem ersten Fallbeispiel allgemein die Durchführbarkeit demonstriert werden, in weiteren Fallbeispielen liegt dann der Fokus auf einzelnen, zu Beginn der Arbeit an den Prozess gestellten Anforderungen.

7.1 Überprüfung der Anforderungen an den Gesamtprozess

In den Abschnitten 4.1 und 6.1 wurde eine Reihe von Anforderungen an den Gesamtprozess gestellt, die die verwendeten Einzelmethoden (ADD, ATAM und CBAM) nicht erfüllen können. Tabelle 7-1 stellt die Anforderungen kurz zusammen und referenziert die entsprechenden Abschnitte aus den vorangegangenen Kapiteln.

#	Anforderung	Kapitel	Fallbeispiel			
			1	2	3	4
1	Formale Beschreibung der Anforderungen	4.3.1	●	●	●	●
2	Formale Beschreibung der Anforderungen einer Komponente			●	●	●
3	Formale Beschreibung von Entwurfsalternativen			●	●	●
4	Unterstützung hybrider Cloud-Architekturen	4.3.2	●	●	●	
5	Bewertbarkeit von Entwurfsalternativen		●	●	●	●
6	Vermeidung unnötigen Overheads		●			
7	Ein- und Ausgaben des Gesamtprozesses	4.3.3	●	●	●	●
8	Modellierbarkeit von Architekturmustern	4.3.4		●		
9	Einmalige Ausführung von Entwurfs- und Bewertungsschritten	6.1	●	●	●	●
10	Zusammenfassung von Arbeitsschritten einzelner Stakeholder			●	●	●
11	Regelmäßige Plausibilitätsprüfungen			●	●	●
12	Minimierung der Anzahl an Arbeitsschritten		●	●	●	●

Tabelle 7-1: Anforderungen an den Gesamtprozess im Überblick

In den Gesamtprozess wurden mit ADD, ATAM und CBAM drei etablierte Methoden zur Erstellung und Bewertung einer Softwarearchitektur eingebracht. Damit sind bereits die Ein- und Ausgaben – und somit Erfüllung von Anforderung 7 – des Prozesses festgelegt. Dieser Umstand soll in den folgenden Abschnitten deshalb nur untergeordnet explizit betrachtet werden. Tabelle 7-1 zeigt, bei welchen der folgenden Fallbeispiele der Untersuchungsfokus auf welchen Anforderungen liegt. Folgende Fallbeispiele werden betrachtet:

- *Fallbeispiel 1: Berechnungsalgorithmus*
  Die hier zu erstellende Software ist bewusst einfach konzipiert, d.h. im Wesentlichen soll nur ein Berechnungsalgorithmus implementiert werden. Das Fallbeispiel dient primär dazu, den Leitfaden auf Einfachheit (Anforderung 6) hin zu prüfen.

- *Fallbeispiel 2: 3-Schicht-Web-Anwendung*
  Hier wird eine fiktive Software, die konzeptionell in klassischer 3-Schichten-Architektur angelegt ist, darauf analysiert, welche der Schichten lokal oder in der Cloud ausgeführt werden.

- *Fallbeispiel 3: Cloud-basierte CAD-Anwendung*
  Für ein reales Anforderungsszenario zur Weiterentwicklung einer bestehenden CAD-Anwendung sollen alternative Ausführungsumgebungen einzelner Komponenten durchgespielt werden. Dabei soll diskutiert werden, ob und, wenn ja, welche Elemente in der Cloud betrieben werden können.

- *Fallbeispiel 4: Migration eines bestehenden Katalog-Webservice in die Cloud*
  In diesem realen Kundenszenario sollte untersucht werden, ob die Migration eines bestehenden Webservice sinnvoll wäre und ob durch grundlegende Änderungen in der Anwendungsarchitektur der Nutzen zusätzlich erhöht werden könnte.

In den beiden ersten Fallbeispielen werden fiktive Softwaresysteme betrachtet, d.h. die Systeme werden nur zum Zweck der Evaluierung des Leitfadens entworfen und betrachtet. Ziel ist dabei, die grundsätzliche Eignung des Leitfadens und Erfüllung der an ihn gestellten Anforderungen zu verifizieren. Die beiden letztgenannten entstammen realen Projekten, für die der Leitfaden zur Analyse von Architekturentscheidungen herangezogen wurde.

7.2 Fallbeispiel 1: Berechnungsalgorithmus

Dieses Fallbeispiel ist bewusst auf Einfachheit ausgelegt. Der Algorithmus ist wenig komplex und die Umsetzung recht naheliegend. Dennoch sollen Alternativen durchgespielt werden, die die Eignung des Leitfadens zeigen. Nur dieses Fallbeispiel soll in Details ausgearbeitet werden, die weiteren Beispiele dann nur noch in den jeweils relevanten Aspekten.

7.2.1 Beschreibung des Fallbeispiels

In diesem Projekt soll die Architektur eines Softwaresystems entwickelt werden, mit dem über eine Monte-Carlo-Simulation die Kreiszahl Pi berechnet werden soll. Nachdem absehbar ist, dass die Softwarearchitektur vergleichsweise einfach bleibt, dient dieses Beispiel primär der Evaluierung hinsichtlich Anforderung 6 (Vermeidung unnötigen Overheads). Zwar sollen auch hier alternative Ausführungsumgebungen miteinander verglichen werden, der eigentliche Algorithmus zur Ergebnisberechnung bleibt dabei aber im Wesentlichen gleich. Im Rahmen der Bewertung sollen zwei Alternativen für die Ausführungsumgebung des eigentlichen Berechnungsmoduls gegenübergestellt werden. Die Berechnung kann in einer lokalen Umgebung (PC des Anwenders) oder in der Cloud ausgeführt werden. Bei der Cloud-Alternative muss deshalb noch die Kommunikation zwischen der lokalen Umgebung und dem Cloud-basierten Berechnungsservice berücksichtigt werden.

7.2.2 Anwendung des Leitfadens

Im Folgenden soll nun der Leitfaden auf das Fallbeispiel angewandt werden. Es werden dabei nur die evaluierungsrelevanten Schritte und Artefakte eingehender betrachtet.

7.2.2.1 Schritt 1: Präsentation der Vorgehensweise

Dieser Schritt soll hier nicht weiter beschrieben werden. Im Wesentlichen wird die in Kapitel 6 ausgeführte Vorgehensweise und die sich daraus ergebenden Rollen, Artefakte und Prozessschritte vorgestellt.

7.2.2.2 Schritt 2: Erstellung eines Business Modells

Zunächst wird das Business Modell erstellt. Dieses enthält die Motivation für das Entwurfs- und Entwicklungsvorhaben und die Information darüber, welcher der beteiligten Stakeholder welche Interessen an das Softwaresystem hat. Das folgende Listing enthält das Business Modell in XML-Notation:

```
<BusinessModel>
   <BusinessDrivers>
      <BusinessDriver
         bdId="0"
         Descr="Effiziente Berechnung von Pi mit einstellbarer
            Genauigkeit." />
      <BusinessDriver
         bdId="1"
         Descr="Minimaler Trainingsaufwand für die Entwicklung durch
            Einsatz bekannter Technologien bei der Umsetzung." />
   </BusinessDrivers>
   <Stakeholders>
```

```
      <Stakeholder shId="0" descr="Anwender" />
      <Stakeholder shId="1" descr="Entwickler" />
   </Stakeholders>
   <StakeBds>
      <StakeBd bdId="0" shId="0" />
      <StakeBd bdId="1" shId="1" />
   </StakeBds>
</BusinessModel>
```

Listing 7-1: Business Modell der Anwendung (Fallbeispiel 1)

Dieses Modell ist sehr einfach gehalten. Letztlich gibt es nur eine vom Anwender gewünschte, knapp umrissene Funktionalität. Für den Entwickler ist wichtig, dass nur bekannte Technologien zum Einsatz kommen. Bereits dieses einfache Modell hat einen nicht zu unterschätzenden Nutzen. Da alle später erfassten Anforderungen an das System aus einem Business Driver abgeleitet werden müssen, kann hierüber bereits eine Diskussion über Motive einzelner Beteiligter erfolgen. Stellt beispielsweise ein Beteiligter die Anforderung in den Raum, der Betrieb müsse vollständig im eigenen Rechenzentrum, also ohne die Ausführungsoption der Public Cloud, erfolgen, kann hinterfragt werden, auf welchen Business Driver sich diese Anforderung bezieht. In obigem Modell ist kein Business Driver enthalten, der eine solche nahelegen würde. Ein solcher Business Driver kann durchaus nachträglich eingefügt werden. Dadurch ist er aber dann dokumentiert und entsprechend einer Diskussion zugänglich.

7.2.2.3 Schritt 3: Ableitung und Priorisierung der Anforderungen

In Schritt 3 wird nun das Anforderungsmodell erstellt. Die einzelnen Business Driver werden dazu in konkrete Anforderungen an das System überführt. Dabei muss geprüft werden, dass sich jede Anforderung aus einem Business Driver ableitet bzw. durch ihn motiviert ist. Die XML-Notation des Anforderungsmodells ist in Listing 7-2 zu sehen.

```
<RequirementsModel>
   <FRs>
      <FR   frId="0"
            stsrc="User"
            st="User starts the application"
            env="Normal operations"
            res="Application asks for the number of iterations"
            descr="Base functionality" />
      <FR   frId="1"
            stsrc="User"
            st="User enters the number of iterations"
            env="Normal operations"
            res="Application calculates Pi and displays the result"
            descr="Base functionality" />
   </FRs>
   <QRs>
      <QR   qrId="0"
            stsrc="User"
```

```
            st="User submits number of iterations"
            env="Normal operations"
            res="Application displays the result"
            resmsr="Efficient calculation of Pi"
            descr="" />
      </QRs>
      <DCs>
         <DC   dcId="0"
               typec=""
               implTc="Custom Code -> Code in C#"
               descr="All custom code must be written in C#" />
         <DC   dcId="1"
               typec=""
               implTc=""
               descr="Calculation to be done with Monte Carlo simulation" />
      </DCs>
      <freqPrios>
         <freqPrio frId="0" prio="10" />
         <freqPrio frId="1" prio="10" />
      </freqPrios>
      <freqBDs>
         <freqBD frId="0" bdId="0" />
         <freqBD frId="1" bdId="0" />
      </freqBDs>
      <qreqPrios>
         <qreqPrio qrId="0" prio="10" />
      </qreqPrios>
      <qreqBDs>
         <qreqBD qrId="0" bdId="0" />
      </qreqBDs>
      <dcBDs>
         <dcBD dcId="0" bdId="1" />
         <dcBD dcId="1" bdId="0" />
      </dcBDs>
   </RequirementsModel>
```

Listing 7-2: Anforderungsmodell der Anwendung (Fallbeispiel 1)

7.2.2.4 Schritt 4: Ableitung und Priorisierung von Szenarien

Mit den bisherigen Schritten liegen das Business- sowie das Anforderungsmodell vor. Nachdem letzteres nur eine einzige Qualitätsanforderung hat, vereinfacht sich die Erstellung des Szenarienmodells: für die Anforderung müssen Szenarien gefunden werden, die die Anforderung genauer beschreiben. Listing 7-3 zeigt das entwickelte Modell in XML-Notation.

```
   <ScenarioModel>
      <Scenarios>
         <Scenario scenId="0"
            desc="With 100M iterations, calculation of Pi in less than 1min." />
         <Scenario scenId="1"
            desc="With 1M iterations, calculation of Pi in less than 1sec." />
         <Scenario scenId="2"
            desc="Calculation with use of all processor cores." />
```

```
    </Scenarios>
    <scenQRs>
       <scenQr scenId="0" qrId="0" />
       <scenQr scenId="1" qrId="0" />
       <scenQr scenId="2" qrId="0" />
    </scenQRs>
    <scenPrios>
       <scenPrio scenId="0" prio="10" />
       <scenPrio scenId="0" prio="10" />
       <scenPrio scenId="0" prio="5" />
    </scenPrios>
</ScenarioModel>
```

Listing 7-3: Szenarienmodell der Anwendung (Fallbeispiel 1)

In diesem Fall sind drei Szenarien ausreichend. Die ersten beiden stellen Performanz-Anforderungen, das dritte zielt auf eine effiziente Ressourcennutzung ab. Alles Szenarien, die die Anforderung der Effizienz genauer beschreiben.

7.2.2.5 Schritt 5: Entwurf der Softwarearchitektur

Aufgrund der Einfachheit der Anwendung ergibt sich recht schnell ein möglicher Architekturentwurf. Aus diesem sind in Abb. 7-1 das Anwendungs- und das Variabilitätsmodell skizziert.

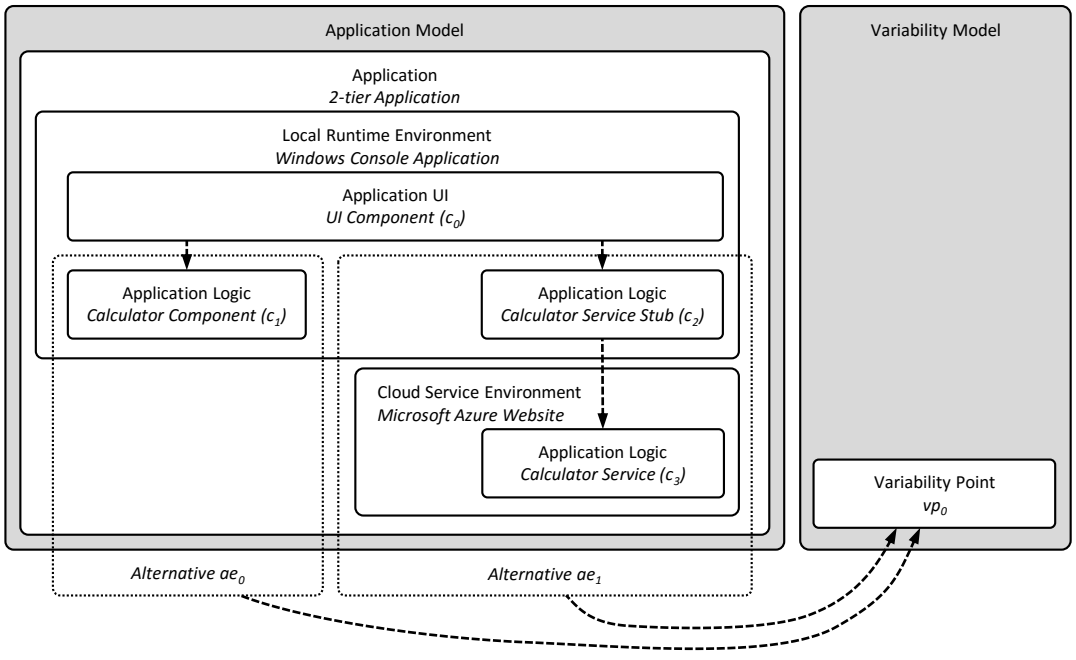

Abb. 7-1 Anwendungs- und Variabilitätsmodell der Berechnungsanwendung

Es wird unmittelbar deutlich, dass das Anwendungsmodell noch Alternativen für die Implementierung der Berechnungskomponente enthält. Die Berechnung kann entweder in der lokalen Ausführungsumgebung erfolgen oder als Service in die Cloud ausgelagert werden. Bei einer Berechnung in der Cloud wird lokal noch ein Service Stub ausgeführt, der die Kommunikation mit dem Cloud Service koordiniert und das Ergebnis an die UI-Komponente zurückgibt. Die Alternativen werden über den Variabilitätspunkt vp_0 beschrieben.

7.2.2.6 Schritt 6: Bewertung der Softwarearchitektur

Für die Bewertung des Architekturentwurfs wird dieser eine Variabilitätspunkt untersucht, d.h. es wird bewertet, welche Auswirkungen eine Entscheidung für Alternative 1 bzw. Alternative 2 auf die Anforderungen hat und wie sich diese letztlich auf den Return-on-Investment auswirkt. Durch die Tatsache, dass nur ein Variabilitätspunkt vorliegt, reduziert sich die Bewertung der Architektur auf die Betrachtung dieses Variabilitätspunkts.

Element des Risikomodells	**Elemente der Menge**	**Anmerkung**
Sensitivity Points	vp_0	Die Auswahl für eine der Alternativen hat Auswirkung auf die Antwortzeiten, also auf die Szenarien $scen_0$ und $scen_1$, die wiederum qr_0 konkretisieren.
Tradeoff Points	$\perp$	Nachdem es nur eine Qualitätsanforderung gibt, kann es auch keinen Tradeoff Point geben.
Risks	vp_0	Eine prototypische Implementierung hat ergeben, dass das System bei Auswahl von Alternative ae_0 (rein lokale Ausführung) qr_0 verletzt, da die Anforderung aus $scen_0$ nicht erfüllt werden kann, d.h. die Durchführung von 100M Iterationen benötigt mehr als 1 Minute Ausführungszeit.
Non-Risks	$\perp$	Da vp_0 einziger Variabilitätspunkt und dieser bereits als Risk eingestuft ist, ist die Menge der Non-Risks leer.
sensitivities	$vp_0 \rightarrow qr_0$	Die Auswahl einer der zu vp_0 gehörenden Alternativen hat Auswirkung auf den Erfüllungsgrad von qr_0.

Tabelle 7-2: Elemente des Risikomodells (Fallbeispiel 1)

Tabelle 7-2 beschreibt das Risikomodell. Aufgrund der Einfachheit der Aufgabenstellung ist dieses entsprechend übersichtlich. Der einzige Variabilitätspunkt vp_0 ist – nachdem die mit ihm verbundenen Alternativen sich auf die Qualitätsanforderung qr_0 auswirken – als Sensitivitätspunkt und als Risiko eingestuft. Insofern muss bei der Auswahl einer Alternative besonderes Augenmerk auf ebendiese Anforderung geworfen werden.

Im nächsten Bewertungsschritt soll nicht nur beurteilt werden „ob“ die qualitativen Anforderungen erfüllt werden sondern auch „wie gut“. Hierzu werden für die einzelnen Szenarien

die Zielerreichungsgrade für die Erreichungsgrade „worst“, „desired“, „best“ aufgestellt. Der Erreichungsgrad „current“ entfällt, da es sich um eine Neuentwicklung handelt. Tabelle 7-3 listet die einzelnen Zielerreichungsgrade auf.

		Zielerreichungsgrad			
Szenario	**Priorität**	**Worst**	**Current**	**Desired**	**Best**
1	10	< 120 sec	n/a	< 60 sec	< 30 sec
2	10	< 3 sec	n/a	< 1 sec	< 0,5 sec
3	5	Single CPU	n/a	Multiple CPU	Multiple CPU

Tabelle 7-3: Response Goals und Prioritäten bezogen auf die Szenarien (Fallbeispiel 1)

Im nächsten Schritt werden die Nutzengrade für die einzelnen Zielerreichungsgrade bestimmt. D.h. es wird festgelegt, welchen Wert die Erreichung einer bestimmten Zielerfüllung (hier auf einer Skala von 0 bis 100 Wertpunkten) hat. Tabelle 7-4 gibt einen Überblick über die einzelnen Wertpunkte.

		Nutzengrad			
Szenario	**Priorität**	**Worst**	**Current**	**Desired**	**Best**
1	10	30	n/a	80	100
2	10	20	n/a	80	100
3	5	10	n/a	100	100

Tabelle 7-4: Nutzengrade bezogen auf die Szenarien (Fallbeispiel 1)

Aufgrund der geringen Komplexität der Aufgabenstellung konnten für die Berechnung von Pi Prototypen erstellt, in den beiden relevanten Ausführungsumgebungen (lokal und Cloud) getestet und entsprechende Werte für die Performanz ermittelt werden. Diese geben bereits recht genaue Hinweise auf die zu erwartende Zielerreichung. Entsprechend genau sind die Angaben in Tabelle 7-5.

Variabilitäts-punkt	Architektur-entscheidung	Beschreibung	Betroffenes Szenario	Erwartete Zielerreichung
vp_0	ae_0	lokale Berechnung	1	2 min 3 sec
			2	1 sec 167 ms
			3	Multiple CPU
	ae_1	Berechnung in Azure	1	23 sec 234 ms
			2	234 ms
			3	Multiple CPU

Tabelle 7-5: Erwartete Zielerreichung für die Szenarien (Fallbeispiel 1)

Für diese erwarteten Zielerreichungen können durch Interpolation die zugehörigen Wertpunkte ermittelt werden. Es kann also berechnet werden, welchem Wert die Performanz-Zahlen entsprechen. Die in Tabelle 7-6 enthaltenen Werte wurden durch grobe Schätzung und nicht durch ein mathematisches Modell ermittelt.

Variabilitäts-punkt	Architektur-entscheidung	Betroffenes Szenario	Erwartete Nutzengrade	Szenario Priorität	Gewichteter Nutzengrad
vp_0	ae_0	1	30	10	300
		2	75	10	750
		3	100	5	500
	ae_1	1	100	10	1000
		2	100	10	1000
		3	100	5	500

Tabelle 7-6: Nutzengrade der Architekturentscheidungen (Fallbeispiel 1)

Bereits hier wird deutlich, dass Alternative ae_1 (Berechnung in der Cloud) bei einer reinen Nutzenbetrachtung deutlich besser abschneidet. Ohne Gegenüberstellung der Kosten würde demnach diese Alternative als die „bessere“ bewertet.

Für eine Berechnung des Return-on-Investment ist nun noch eine Einbeziehung der Kosten erforderlich. Diese werden hier grob auf Basis von Erfahrungswerten geschätzt. Hier wird unterstellt, dass eine Implementierung von Alternative ae_1 ca. 50% teurer als eine Implementierung von Alternative ae_0 ist. Tabelle 7-8 listet die einzelnen Werte für Gesamtnutzen und den Kosten auf.

Architektur-entscheidung	Gewichtete Nutzengrade	Gesamtnutzen der AE	Kosten der AE	ROI der AE
ae_0	300 + 750 + 500	1550	100	15,5
ae_1	1000 + 1000 + 500	2500	150	16,7

Tabelle 7-7: Berechnung des ROI für die Architekturentscheidungen (Fallbeispiel 1)

Das Ergebnis zeigt, dass Alternative ae_1 mit einem ROI-Wert von 16,7 besser abschneidet als Alternative ae_0 mit einem ROI-Wert 15,5. Nachdem das Variabilitätsmodell nur einen einzigen Variabilitätspunkt enthält, entfällt ein Aufsummieren zur Berechnung des ROI-Wertes von Architekturalternativen.

7.2.2.7 Schritt 7: Präsentation der Entwurfs- und Bewertungsergebnisse

Der Unterschied der beiden Return-on-Investment-Werte kann nun verwendet werden, um für die Umsetzung von Alternative ae_1 zu plädieren. Bei der Ergebnispräsentation können neben diesen Werten noch weitere Erkenntnisse vorgetragen werden:

- Die Entscheidung zwischen pro und contra Cloud hat – wie zu erwarten – Konsequenzen auf die Performanz der Anwendung. Dies ist in Tabelle 7-5 zu sehen.
- Die am Architekturentwurf beteiligten Personen gewichten die Performanz-Vorteile der Cloud höher ein als den Nachteil der höheren Kosten. Dies ist erkennbar am höheren Return-on-Investment der Cloud-Alternative in Tabelle 7-7.

Auf Basis der Entwurfsentscheidungen, Priorisierungen und Analyseergebnisse kann eine Empfehlung für Alternative ae_1 ausgesprochen werden.

Bereits dieses einfache Fallbeispiel zeigt aber auch, dass das Ergebnis von vielen subjektiv eingebrachten Faktoren (Priorisierungen, Schätzungen etc.) abhängt, die wiederum Gegenstand von Diskussionen sein können:

- Würde die Performanz-Anforderung etwas weniger streng gesetzt, könnte Alternative ae_0 eine höhere Nutzenbewertung erzielen, wodurch sich deren Return-on-Investment erhöhen würde.
- Sofern eine Umsetzung mit Hilfe von Cloud-Technologien mehr als 50% Kostensteigerungen verursachte, wäre der Return-on-Investment für Alternative ae_1 schlechter als oben bewertet und Alternative ae_0 doch besser geeignet.
- Prozessbeteiligte, die Cloud Computing per se ablehnen, müssten in die obigen Modelle konkrete Einwände einbringen (z.B. Sicherheitsbedenken mit entsprechenden

Priorisierungen), um das Endergebnis in Richtung einer lokalen Implementierung zu beeinflussen.

Sofern nun tatsächlich eine Entscheidung für Alternative ae_1 erfolgt, kann nun genau begründet werden, auf welche Entscheidungskriterien sich diese Wahl bezog.

7.2.3 Ergebnis der Durchführung

Die oben beschriebene Anwendung des Leitfadens hat eine Entscheidung für eine Architekturalternative hervorgebracht. Tabelle 7-8 beurteilt, inwieweit dabei die Anforderungen an den Leitfaden erfüllt wurden.

#	Anforderung	Bewertung	Anforderung erfüllt
1	Formale Beschreibung der Anforderungen	Die Anforderungen wurden formal beschrieben.	Ja.
2	Formale Beschreibung der Anforderungen einer Komponente	Nicht untersucht.	n/a
3	Formale Beschreibung von Entwurfsalternativen	Nicht untersucht.	n/a
4	Unterstützung hybrider Cloud-Architekturen	Das Fallbeispiel enthielt eine hybride Cloud-Alternative. Der Leitfaden hat deren Bewertung unterstützt.	Ja.
5	Bewertbarkeit von Entwurfsalternativen	Die Alternativen wurden bewertet.	Ja.
6	Vermeidung unnötigen Overheads	Die begrenzte Komplexität der Anwendung hat Vereinfachungen (z.B. Grobschätzung von Nutzenwerten) erlaubt.	Ja.
7	Ein- und Ausgaben des Gesamtprozesses	Ein- und Ausgaben wurden beschrieben.	Ja.
8	Modellierbarkeit von Architekturmustern	Nicht untersucht.	n/a
9	Einmalige Ausführung von Entwurfs- und Bewertungsschritten	Priorisierungen wurden nur einmal vorgenommen.	Ja.
10	Zusammenfassung von Arbeitsschritten einzelner Stakeholder	Nicht untersucht.	n/a
11	Regelmäßige Plausibilitätsprüfungen	Nicht untersucht.	n/a

#	Anforderung	Bewertung	Anforderung erfüllt
12	Minimierung der Anzahl an Arbeitsschritten	Vereinfachungen in der Durchführung waren möglich. Keine Wiederholung von Prozess-schritten.	Ja.

Tabelle 7-8: Beurteilung der Anwendung des Leitfadens in Fallbeispiel 1

7.3 Fallbeispiel 2: 3-Schicht-Web-Anwendung

In diesem Fallbeispiel soll eine typische Fragestellung im Rahmen von Cloud-basierten Anwendungen durchgespielt werden: die Frage nach der besten Ausführungsalternativen für die Schichten einer klassischen 3-Schicht-Web-Anwendung, bestehend aus Frontend, Backend und Datenbank. Das Fallbeispiel soll dabei helfen, typische Argumentationen zum Thema Cloud Computing (Kontrolle über die Umgebung vs. Flexibilität bei der Ausführung) formal durchzuspielen.

7.3.1 Beschreibung des Fallbeispiels

Dieses Fallbeispiel bildet ein klassisches Vorhaben nach: es soll eine typische 3-Schicht-Web-Anwendung erstellt werden. Für die Funktionalität wird ein Bilder-Archiv als Vorbild genommen. Über eine Web-Frontend-Eingabemaske sollen Bilder in die Anwendung geladen werden können, eine Backend-Komponente soll dann asynchron (also entkoppelt vom Frontend) diese Bilder in unterschiedliche Formate konvertieren und im Speicher der Anwendung ablegen. Von dort sollen sie dann, wiederum über die Frontend-Maske abrufbar sein. Das Fallbeispiel ist bewusst einfach gehalten, um den Entwurf und die Bewertung der Architektur mit Hilfe des Leitfadens durchspielen zu können. Komplexere Funktionalitäten sollten an der grundsätzlichen Durchführung nichts ändern. Für die Umsetzung sollen klassische Patterns für eine Implementierung auf Basis von Microsoft Azure Verwendung finden. Es sollen, neben einer Umsetzung in einer lokalen Umgebung, auch eine Umsetzung via Azure Web App und eine Implementierung auf Basis von Azure Virtual Machines durchgespielt werden.

7.3.2 Anwendung des Leitfadens

Mit Hilfe des Leitfadens soll nun schrittweise eine Anwendungsarchitektur mit verschiedenen Umsetzungsalternativen erstellt und die Alternativen bewertet werden. Wie im vorangegangenen Fallbeispiel soll der Fokus auf den entwurfs- und evaluierungsrelevanten Schritten und Artefakten liegen.

7.3.2.1 Schritt 1: Präsentation der Vorgehensweise

Dieser Schritt soll hier ebenfalls nicht weiter beschrieben werden. Er besteht aus der Vorstellung in Kapitel 6 ausgeführte Vorgehensweise und den sich daraus ergebenden Rollen, Artefakten und Prozessschritten.

7.3.2.2 Schritt 2: Erstellung eines Business Modells

Die Diskussionen finden üblicherweise unter Beteiligung des Produktmanagements, das Entscheidungen zu funktionalen und qualitativen Anforderungen trifft, der Entwicklungsleitung, das Fragen zur technischen Umsetzung beantwortet, und des Systembetriebs statt, der Eingaben zum Betrieb des späteren Systems macht. Die einzusetzenden Technologien werden häufig durch diese Gruppen bestimmt, d.h. selten besteht hier Wahlfreiheit. Im konkreten Fall soll von einer Umsetzung auf Basis der .NET-Plattform mittels C# sowie einem Einsatz eines SQL-basierten Datenbanksystems ausgegangen werden. Die grobe Architektur (3-Schichten) ist durch das Fallbeispiel vorgegeben. Weitere Business Driver beziehen sich auf die Fragestellung nach dem Für und Wider der Cloud, d.h. Flexibilität der Cloud vs. Kontrolle über die Umgebung. Tabelle 7-9 zeigt die wichtigsten Inhalte des Business Modells.

Id	Business Driver	Stakeholder	Beschreibung
bd_1	Funktionalität eines einfachen Bilder-Archivs	Produktmanagement	Die Anwendung soll den Upload von Bildern in hoher Auflösung gestatten. Diese werden in verschiedene Auflösungen konvertiert und zum Download bereitgestellt.
bd_2	Minimaler Trainingsaufwand für Entwickler	Entwicklungsleitung	Einsatz bekannter Technologien (C#/.NET/SQL).
bd_3	Hohe "Responsiveness" der Anwendung	Produktmanagement	Das Frontend soll während der Bilderkonvertierung zugreifbar sein, d.h. nicht blockiert sein.
bd_4	Nutzung der Elastizität der Cloud	Entwicklungsleitung	Die Vorteile der Cloud (Skalierbarkeit, schnelle Bereitstellung etc.) sollten genutzt werden.
bd_5	Kontrolle über die Umgebung	Systembetrieb	Zugriff auf die jeweiligen Umgebungen soll für administrative Zwecke möglich sein.

Tabelle 7-9: Business Driver mit zugehörigen Stakeholdern (Fallbeispiel 2)

Als zu betrachtende Cloud Plattform wird Microsoft Azure gewählt (vergleichbare Betrachtungen sind jedoch auch mit anderen Cloud Plattformen möglich). Hierbei sollen typische

Architekturmuster zur Umsetzung einer 3-Schichten-Architektur miteinander verglichen und auf Basis der priorisierten Anforderungen miteinander verglichen werden.

7.3.2.3 Schritt 3: Ableitung und Priorisierung der Anforderungen

Im nächsten Schritt wird aus den Business Drivern ein initiales Anforderungsmodell erstellt. Auch hier ist wichtig, dass sich jeder Business Driver in mindestens einer Anforderung wiederfindet bzw. jede Anforderung durch einen Business Driver motiviert ist, um den Ursprung der Anforderung rückverfolgen zu können. Tabelle 7-10 stellt das initiale Anforderungsmodell vereinfacht dar.

Id	Typ	Beschreibung	Business Driver
fr_1^{init}	fachlich	Möglichkeit zum Upload von Bildern	bd_1
fr_2^{init}	fachlich	Hintergrundverarbeitung von Bildern	bd_1
fr_3^{init}	fachlich	Anzeige von Bildern	bd_1
qr_1^{init}	qualitativ	Skalierbarkeit der Anwendung	bd_4
qr_2^{init}	qualitativ	Kontrolle über die Anwendungsumgebung	bd_5
dc_1^{init}	Einschränkung	Beibehaltung des grundlegenden Architekturaufbaus	bd_1
dc_2^{init}	Einschränkung	Lose Koppelung von Frontend und Backend	bd_3
dc_3^{init}	Einschränkung	Beibehaltung der Technologien (.NET, SQL)	bd_2

Tabelle 7-10: Initiales Anforderungsmodell mit Business Drivern (Fallbeispiel 2)

Durch die Entwurfseinschränkungen ist bereits der grobe Architekturaufbau in Form einer 3-Schichten-Architektur absehbar.

7.3.2.4 Schritt 4: Entwurf der Softwarearchitektur

Durch die Festlegung auf eine 3-Schichten-Architektur ist die Ableitung des Architekturentwurfs recht einfach. Dieser ist in Abb. 7-2 zu sehen.

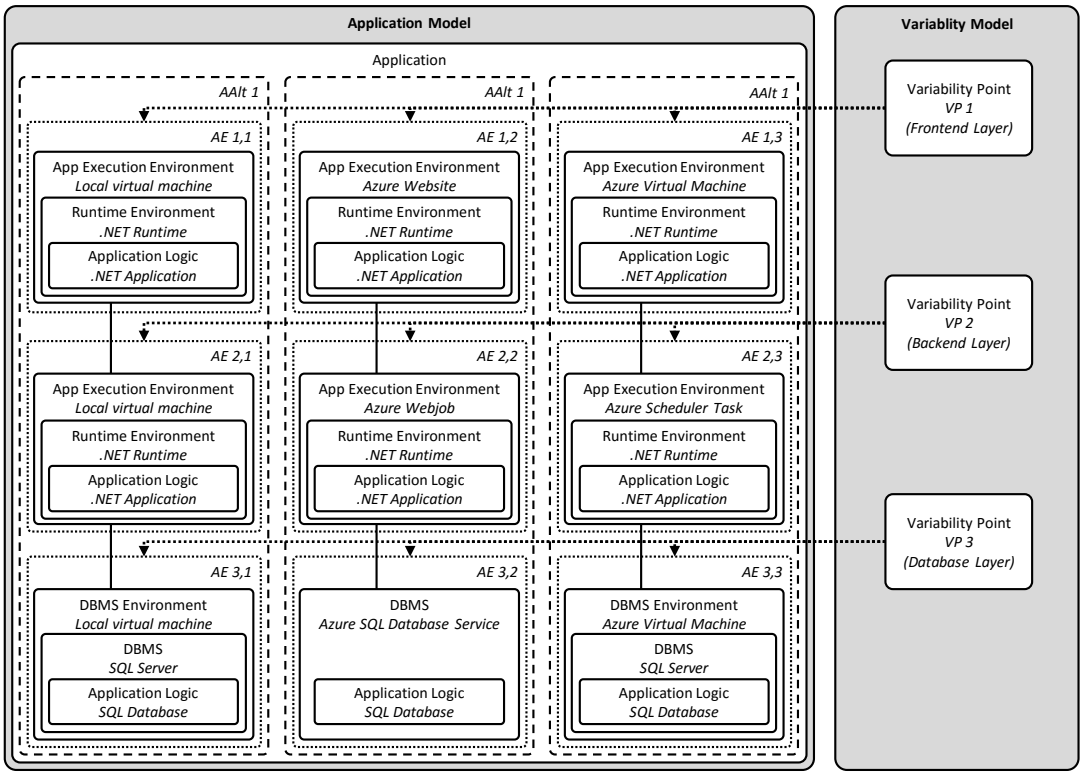

Abb. 7-2 Anwendungs- und Variabilitätsmodell der 3-Schicht-Web-Anwendung

Das Anwendungsmodell zeigt die drei Schichten der Architektur. Jede dieser Schichten ist mit einem Variabilitätspunkt verbunden, da in jeder Schicht je drei Architekturentscheidungen möglich sind. Diese Alternativen werden in Tabelle 7-11 aufgelistet.

Variabilitätspunkt	**Beschreibung**	**Architekturentscheidung**	**Beschreibung**
vp_1	Ausführung der Frontend-Logik	$ae_{1,1}$	Ausführung in lokaler Umgebung
		$ae_{1,2}$	Ausführung in einer Azure Web App
		$ae_{1,3}$	Ausführung in einer Azure Virtual Machine
vp_2	Ausführung der Backend-Logik	$ae_{2,1}$	Ausführung in lokaler Umgebung
		$ae_{2,2}$	Ausführung als Azure Webjob
		$ae_{2,3}$	Ausführung über den Azure Scheduler
vp_3	Ausführung der Datenbank	$ae_{3,1}$	Ausführung in lokaler Datenbank (SQL Server)
		$ae_{3,2}$	Ausführung via Azure SQL Database
		$ae_{3,3}$	Ausführung in SQL Server in einer Azure VM

Tabelle 7-11: Variabilitätspunkte und Architekturentscheidungen (Fallbeispiel 2)

Nicht jede Kombination der Architekturentscheidungen soll weiter betrachtet werden. Durch die Vorgabe der Architekturmuster beschränken sich die Alternativen (also Kombinationen aus Architekturentscheidungen) auf drei: einen vollständig lokalen Betrieb aller Schichten, einen Betrieb entsprechend dem Azure Web App Pattern und einen Betrieb nach dem Azure Virtual Machine Pattern. Tabelle 7-12 listet die Alternativen nochmals auf.

Architektur-alternative	Tupel zugehöriger Architekturentscheidungen	Beschreibung
$Aalt_1$	$(ae_{1,1}, ae_{2,1}, ae_{3,1})$	Vollständiger Betrieb der Anwendung in einer lokalen Ausführungsumgebung (lokale Server).
$Aalt_2$	$(ae_{2,2}, ae_{2,2}, ae_{3,2})$	Betrieb nach den Azure Web App Pattern (Azure Web App, Webjobs, SQL Database Service).
$Aalt_3$	$(ae_{1,3}, ae_{2,3}, ae_{3,3})$	Betrieb nach dem Azure Virtual Machine Pattern (jede Schicht in einer Azure Virtual Machine, Hintergrundprozesse über Azure Scheduler gesteuert).

Tabelle 7-12: Zu bewertende Architekturalternativen (Fallbeispiel 2)

Ergebnis der folgenden Bewertungsschritte soll demnach eine Empfehlung für eine dieser Architekturalternativen sein.

Im Rahmen des Architekturentwurfs wurde das initiale Anforderungsmodell in ein finales Modell überführt, dabei wurden die Qualitätsanforderungen für die einzelnen Schichten heruntergebrochen. Dabei wurden aus qr_1^{init} die spezifischeren Anforderungen qr_1, qr_2 und qr_3 und aus qr_2^{init} die spezifischeren Anforderungen qr_4, qr_5 und qr_6 abgeleitet. Daraus ergab sich somit das in Tabelle 7-13 abgebildete finale Anforderungsmodell, das Grundlage für die weiteren Evaluierungsschritte war.

Id	Typ	Beschreibung	Business Driver
fr_1	fachlich	Möglichkeit zum Upload von Bildern	bd_1
fr_2	fachlich	Hintergrundverarbeitung von Bildern	bd_1
fr_3	fachlich	Anzeige von Bildern	bd_1
qr_1	qualitativ	Skalierbarkeit des Frontends	bd_4
qr_2	qualitativ	Skalierbarkeit des Backends	bd_4
qr_3	qualitativ	Skalierbarkeit der Datenbank	bd_4
qr_4	qualitativ	Kontrolle über die Umgebung des Frontend	bd_5
qr_5	qualitativ	Kontrolle über die Umgebung des Backends	bd_5
qr_6	qualitativ	Kontrolle über die Umgebung der Datenbank	bd_5
dc_1	Einschränkung	Beibehaltung des grundlegenden Architekturaufbaus	bd_1
dc_2	Einschränkung	Lose Koppelung von Frontend und Backend	bd_3

Id	Typ	Beschreibung	Business Driver
dc_3	Einschränkung	Beibehaltung der Technologien (.NET, SQL)	bd_2

Tabelle 7-13: Finales Anforderungsmodell mit Business Drivern (Fallbeispiel 2)

7.3.2.5 Schritt 5: Ableitung und Priorisierung von Szenarien

Es folgt die Erstellung des Szenarienmodells. Die Anforderungen zur Skalierbarkeit werden dabei heruntergebrochen auf je ein Szenario zur Skalierbarkeit um zusätzliche Instanzen und die Bereitstellungszeit für Bugfixes. Die Anforderungen zur Kontrolle werden über Szenarien zum Zugriff auf die jeweilige Umgebung konkretisiert. Tabelle 7-14 stellt die priorisierten Szenarien vereinfacht dar.

Id	Szenario	Priorität	Anforderung
$scen_{1,1}$	Skalierbarkeit des Frontends zu Peak-Zeiten um zusätzliche Instanzen	8	qr_1
$scen_{1,2}$	Bereitstellungszeit für Bugfixes für das Frontend	8	qr_1
$scen_{2,1}$	Skalierbarkeit des Backends zu Peak-Zeiten um zusätzliche Instanzen	10	qr_2
$scen_{2,2}$	Bereitstellungszeit für Bugfixes für das Backend	8	qr_2
$scen_{3,1}$	Skalierbarkeit der Datenbank zu Peak-Zeiten um zusätzliche Instanzen	5	qr_3
$scen_{3,2}$	Bereitstellungszeit für Bugfixes für die Datenbank	8	qr_3
$scen_{4,1}$	Zugriff auf die Laufzeitumgebung des Frontends	6	qr_4
$scen_{5,1}$	Zugriff auf die Laufzeitumgebung des Backends	8	qr_5
$scen_{6,1}$	Zugriff auf die Laufzeitumgebung der Datenbank	3	qr_6

Tabelle 7-14: Priorisierte Szenarien (Fallbeispiel 2)

Diese neun Szenarien dienen in der Folge der Bewertung des Architekturentwurfs. Die Priorisierungen zeigen, dass der Skalierbarkeit des Backends hohe Bedeutung zukommt. Die Datenbank wird demgegenüber als weniger kritisch angesehen, deren Skalierbarkeit also als weniger wichtig eingestuft. Insgesamt werden die Szenarien zur Flexibilität der Cloud etwas höher priorisiert als die Szenarien zur Kontrolle über die Umgebung.

7.3.2.6 Schritt 6: Bewertung der Softwarearchitektur

Die Risikobetrachtung ergibt recht schnell, dass jeder der Variabilitätspunkte auf mindestens zwei Qualitätsanforderungen Einfluss hat und jeweils mindestens eine Entscheidung zur Verletzung einer der Anforderungen führt. Somit sind alle drei Variabilitätspunkte Sensitivity Points, Tradeoff Points und Risiken. Das Risikomodell ist in Tabelle 7-15 dargestellt.

Element des Risikomodells	**Elemente der Menge**	**Anmerkung**
Sensitivity Points	vp_1, vp_2, vp_3	Jeder der Variabilitätspunkte hat Abhängigkeiten zum Erfüllungsgrad von Qualitätsanforderungen. Somit sind alle Sensitivity Points.
Tradeoff Points	vp_1, vp_2, vp_3	Die Architekturentscheidungen aller Variabilitätspunkte haben Auswirkungen auf jeweils zwei Qualitätsanforderungen (siehe sensitivities). Somit sind alle Variabilitätspunkte auch Tradeoff Points.
Risks	vp_1, vp_2, vp_3	In allen Variabilitätspunkten können Entscheidungen zur Verletzung von Anforderungen führen. In jeder Schicht (also jedem Variabilitätspunkt) verletzt die Entscheidung für einen lokalen Betrieb die Anforderung an Skalierbarkeit.
Non-Risks	$\perp$	Alle Variabilitätspunkte wurden als Risks eingestuft.
sensitivities	$vp_1 \rightarrow qr_1$, $vp_1 \rightarrow qr_4$, $vp_2 \rightarrow qr_2$, $vp_2 \rightarrow qr_5$, $vp_3 \rightarrow qr_3$, $vp_3 \rightarrow qr_6$	

Tabelle 7-15: Elemente des Risikomodells (Fallbeispiel 2)

Für die einzelnen Szenarien können nun Zielerreichungsgrade aufgestellt werden. Da es sich um eine Neuentwicklung handelt, bleibt der Zielerreichungsgrad für „current“ leer. Das Szenarienmodell stellt sich wie folgt dar:

		Zielerreichungsgrad			
Szenario	**Prioritäten**	**Worst**	**Current**	**Desired**	**Best**
$scen_{1,1}$	8	1 Tag	n/a	1 Stunde	5 Minuten
$scen_{1,2}$	8	1 Tag	n/a	1 Stunde	5 Minuten
$scen_{2,1}$	10	1 Tag	n/a	1 Stunde	5 Minuten
$scen_{2,2}$	8	1 Tag	n/a	1 Stunde	5 Minuten
$scen_{3,1}$	5	1 Tag	n/a	1 Stunde	5 Minuten

		Zielerreichungsgrad			
Szenario	**Prioritäten**	**Worst**	**Current**	**Desired**	**Best**
$scen_{3,2}$	8	1 Tag	n/a	1 Stunde	5 Minuten
$scen_{4,1}$	6	Admin Tools	n/a	Admin Konsole / Remote Desktop	Zugriff auf phys. Umgebung
$scen_{5,1}$	8	Admin Tools	n/a	Admin Konsole / Remote Desktop	Zugriff auf phys. Umgebung
$scen_{6,1}$	3	Admin Tools	n/a	Admin Konsole / Remote Desktop	Zugriff auf phys. Umgebung

Tabelle 7-16: Response Goals und Prioritäten bezogen auf die Szenarien (Fallbeispiel 2)

Für die Szenarien $scen_{4,1}$, $scen_{5,1}$ und $scen_{6,1}$ werden keine kontinuierlichen Werte sondern verschiedene Ausprägungen angegeben. Die Möglichkeit zu einem Zugriff auf die physische Umgebung wird dabei im Sinne des Szenarios als ideal, Zugriff via Admin Konsole oder Remote Desktop als akzeptabel und ein Zugriff über Verwaltungswerkzeuge als eher schlecht angesehen. Diesen Zielerreichungsgraden wurden die in Tabelle 7-17 aufgelisteten Nutzengrade zugeordnet. Es wurde eine Skala von 0 bis 100 Wertpunkten gewählt.

		Nutzengrad			
Szenario	**Prioritäten**	**Worst**	**Current**	**Desired**	**Best**
$scen_{1,1}$	8	0	n/a	80	100
$scen_{1,2}$	8	20	n/a	80	100
$scen_{2,1}$	10	0	n/a	80	100
$scen_{2,2}$	8	20	n/a	80	100
$scen_{3,1}$	5	0	n/a	80	100
$scen_{3,2}$	8	10	n/a	80	100
$scen_{4,1}$	6	10	n/a	60	100
$scen_{5,1}$	8	10	n/a	60	100
$scen_{6,1}$	3	20	n/a	60	100

Tabelle 7-17: Nutzengrade bezogen auf die Szenarien (Fallbeispiel 2)

Im wohl wichtigsten folgenden Bewertungsschritt werden für die einzelnen Architekturentscheidungen die erwarteten Zielerreichungen abgeschätzt und der erwartete Nutzengrad in-

terpoliert. Zusammen mit der Priorität des betreffenden Szenarios ergibt sich dann der gewichtete Nutzengrad einer Architekturentscheidung für ein bestimmtes Szenario. Tabelle 7-18 listet die einzelnen Werte auf.

Architektur-entscheidung	Betroffenes Szenario	Erwartete Zielerreichung	Erwarteter Nutzengrad	Priorität des Szenarios	Gewichteter Nutzengrad
$ae_{1,1}$	$scen_{1,1}$	1 Tag	0	8	0
	$scen_{1,2}$	1 Stunde	80	8	640
	$scen_{4,1}$	Zugriff auf phys. Umg.	100	6	600
$ae_{1,2}$	$scen_{1,1}$	5 Minuten	100	8	800
	$scen_{1,2}$	5 Minuten	100	8	800
	$scen_{4,1}$	Admin Konsole	30	6	180
$ae_{1,3}$	$scen_{1,1}$	15 Minuten	90	8	720
	$scen_{1,2}$	15 Minuten	90	8	720
	$scen_{4,1}$	Remote Desktop	60	6	360
$ae_{2,1}$	$scen_{2,1}$	1 Tag	0	10	0
	$scen_{2,2}$	1 Stunde	80	8	640
	$scen_{5,1}$	Zugriff auf phys. Umg.	100	8	800
$ae_{2,2}$	$scen_{2,1}$	5 Minuten	100	10	1000
	$scen_{2,2}$	5 Minuten	100	8	800
	$scen_{5,1}$	Admin Konsole	30	8	240
$ae_{2,3}$	$scen_{2,1}$	15 Minuten	90	10	900
	$scen_{2,2}$	15 Minuten	90	8	720
	$scen_{5,1}$	Remote Desktop	60	8	480
$ae_{3,1}$	$scen_{3,1}$	1 Tag	0	5	0
	$scen_{3,2}$	5 Minuten	100	8	800
	$scen_{6,1}$	Zugriff auf phys. Umg.	100	3	300
$ae_{3,2}$	$scen_{3,1}$	15 Minuten	90	5	450
	$scen_{3,2}$	5 Minuten	100	8	800
	$scen_{6,1}$	Admin Tools	20	3	60
$ae_{3,3}$	$scen_{3,1}$	30 Minuten	85	5	425
	$scen_{3,2}$	5 Minuten	100	8	800
	$scen_{6,1}$	Remote Desktop	60	3	180

Tabelle 7-18: Erwartete Zielerreichung für die Szenarien (Fallbeispiel 2)

Nach den Nutzengraden fehlt zur Berechnung noch eine Abschätzung der Kosten. Anstelle einer Betrachtung mit konkreten Geldbeträgen werden hier nur relative Kostengrößen gegenübergestellt. Dabei wird zwischen Implementierungskosten und den Betriebskosten unterschieden. Nachdem die eigentliche Implementierung in allen Architekturalternativen sehr ähnlich ist (der Code für einen lokalen Betrieb unterscheidet sich kaum vom Code für einen Betrieb in einer Azure Web App oder Azure Virtual Machine) sind die Kosten sehr ähnlich angesetzt. Lediglich die Umsetzung der Datenbank über eine Azure SQL Database wird hier etwas höher angesehen, da hier ggf. Anpassungen im Datenmodell erforderlich sind. Dafür hat Azure SQL Database deutliche Kostenvorteile im Betrieb, da hier ein vollautomatisch betriebener Datenbank-Service und kein manuell zu konfigurierendes Datenbanksystem zum Einsatz kommt. Tabelle 7-19 listet die Kosten auf.

Architektur-entscheidung	**Kosten für die Implementierung**	**Kosten für den Betrieb (1 Jahr)**	**Kosten der Architekturentscheidung**
$ae_{1,1}$	100	150	250
$ae_{1,2}$	100	50	150
$ae_{1,3}$	100	100	200
$ae_{2,1}$	70	150	220
$ae_{2,2}$	70	50	120
$ae_{2,3}$	70	100	170
$ae_{3,1}$	80	150	230
$ae_{3,2}$	150	20	170
$ae_{3,3}$	80	100	180

Tabelle 7-19: Kosten für die Architekturentscheidungen (Fallbeispiel 2)

Aus den so gewonnenen Werten können nun die Return-on-Investments für die einzelnen Architekturentscheidungen abgeleitet werden. Diese sind in Tabelle 7-20 zu sehen.

Architektur-entscheidung	**Gewichtete Nutzengrade**	**Gesamtnutzen der AE**	**Kosten der AE**	**ROI der AE**
$ae_{1,1}$	0 + 640 + 600	1240	250	4,96
$ae_{1,2}$	800 + 800 + 180	1780	150	11,87
$ae_{1,3}$	720 + 720 + 360	1800	200	9,00
$ae_{2,1}$	0 + 640 + 800	1440	220	6,55
$ae_{2,2}$	1000 + 800 + 240	2040	120	17,00
$ae_{2,3}$	900 + 720 + 480	2100	170	12,35
$ae_{3,1}$	0 + 800 + 300	1100	230	4,78

Architektur-entscheidung	Gewichtete Nutzengrade	Gesamtnutzen der AE	Kosten der AE	ROI der AE
$ae_{3,2}$	450 + 800 + 60	1310	170	7,71
$ae_{3,3}$	425 + 800 + 180	1405	180	7,81

Tabelle 7-20: Berechnung des ROI für die Architekturentscheidungen (Fallbeispiel 2)

Bereits hier ist erkennbar, dass insbesondere im Backend-Bereich mit den hohen Skalierbarkeitsanforderungen hohe ROI-Werte für einen Betrieb in der Cloud zu erwarten sind. Aber auch beim Frontend und der Datenbank sind unter den gegebenen Voraussetzungen Vorteile der Cloud zu sehen. Dies schlägt sich auch in der ROI-Betrachtung der Architekturalternativen nieder. Diese ist in Tabelle 7-21 zu sehen.

Architektur-alternative	Nutzengrade der betreffenden Architekturentscheidungen	Gesamtnutzen der AAlt	Kosten der AAlt	ROI der AAlt
$Aalt_1$	1240 + 1440 + 1100	3780	700	5,40
$Aalt_2$	1780 + 2040 + 1310	5130	440	11,66
$Aalt_3$	1800 + 2100 + 1405	5305	550	9,65

Tabelle 7-21: Berechnung des ROI für die Architekturalternativen (Fallbeispiel 2)

Unter den gegebenen Anforderungen und deren Priorisierungen ist demnach $Aalt_2$ die Alternative mit dem höchsten ROI-Wert und somit zu empfehlen. Dies ist aus folgenden Gründen plausibel:

- Mit dieser Alternative ergeben sich die besten Werte bei der Skalierbarkeit.
- Die eingesetzte SQL Database hat deutliche Vorteile in den Betriebskosten.
- Zwar ist die Kontrolle über die Umgebung etwas schlechter als bei den anderen Alternativen, diese wurde im Rahmen der Priorisierung als weniger wichtig eingestuft.

Dass die Empfehlung von der Priorisierung der Szenarien stark abhängt, soll mit Hilfe einer hypothetischen Umpriorisierung verdeutlicht werden. Tabelle 7-22 zeigt alternative Prioritäten für die einzelnen Szenarien und die daraus resultierenden alternativen ROI-Werte der Architekturalternativen.

	Tatsächlich angenommene Prioritäten	Prioritäten, wenn Kontrolle sehr wichtig angesehen wird	Prioritäten, wenn dazu Bereitstellungszeit und Skalierbarkeit nicht wichtig sind
$scen_{1,1}$	8	8	5
$scen_{1,2}$	8	8	3

	Tatsächlich angenommene Prioritäten	Prioritäten, wenn Kontrolle sehr wichtig angesehen wird	Prioritäten, wenn dazu Bereitstellungszeit und Skalierbarkeit nicht wichtig sind
$scen_{2,1}$	10	10	5
$scen_{2,2}$	8	8	3
$scen_{3,1}$	5	5	5
$scen_{3,2}$	8	8	3
$scen_{4,1}$	6	36	36
$scen_{5,1}$	8	48	48
$scen_{6,1}$	3	18	18
	Return-on-Investments der Architekturalternativen		
$Aalt_1$	5,40	17,54	15,69
$Aalt_2$	11,66	17,11	11,89
$Aalt_3$	9,65	18,92	15,06

Tabelle 7-22 Hypothetische ROIs bei alternativen Prioritäten (Fallbeispiel 2)

Häufig wird in Kundengesprächen die Kontrolle über die Anwendungssysteme (Server, Datenbank etc.) als sehr wichtig angesehen. Bevorzugt werden Umgebungen, auf die entweder physischer Zugriff oder zumindest administrativer Zugriff via Remote Desktop möglich sind. Spalte 3 in Tabelle 7-22 spiegelt dies in entsprechenden Prioritätswerten der betreffenden Szenarien $scen_{4,1}$, $scen_{5,1}$ und $scen_{6,1}$ wider. Tatsächlich ergeben sich dann deutlich bessere ROI-Werte für einen lokalen Betrieb ($Aalt_1$) und einen Betrieb in Azure Virtual Machines ($Aalt_3$). $Aalt_2$ ist verdrängt auf den dritten Platz. Immer noch bringt aber die Cloud ihre Vorteile ein. Werden diese geringer wichtig angesehen (Spalte 4 in Tabelle 7-22), ist der lokale Betrieb der mit dem höchsten ROI-Wert.

7.3.2.7 Schritt 7: Präsentation der Entwurfs- und Bewertungsergebnisse

Die Ergebnisse der Entwurfs- und Bewertungsaktivitäten können nun wie folgt zusammengefasst werden:

- Alle drei Variabilitätspunkte wurden als Risiko eingestuft. Es wurden Anforderungen formuliert, die in jedem Variabilitätspunkt bei einer Entscheidung verletzt wurden. Der lokale Betrieb einer Komponente verletzt in der Regel die Anforderungen an Skalierbarkeit, die bei der Cloud gegeben sind. Es obliegt nun den Beteiligten zu entscheiden, ob die Anforderungen ggf. aufgehoben oder aufgeweicht werden können.

- Die Architekturalternative $Aalt_2$ wurde als die Alternative mit dem höchsten Return-on-Investment identifiziert. Unter den gegebenen priorisierten Anforderungen hat sie ihre Stärken in der Skalierbarkeit und den Bereitstellungszeiten. Zwar hat sie ihre Schwäche in der Kontrolle über die Umgebung, doch diese wurde zu Beginn als nicht entscheidend wichtig eingestuft.

- Zweitbeste Alternative war $Aalt_3$, d.h. der Betrieb in Azure Virtual Machines. Auch hier bestehen unter den gegebenen Voraussetzungen Vorteile der Cloud, die höheren Betriebskosten im Vergleich zu $Aalt_2$ verschlechtern aber den ROI-Wert.

- Ein lokaler Betrieb entsprechend Architekturalternative $Aalt_1$ wurde als drittbeste Alternative angesehen. Dies ist den Priorisierungen der Anforderungen geschuldet. Würden die Anforderungen an die Skalierbarkeit niedriger und die an Kontrolle über die Umgebung deutlich höher priorisiert, würde diese Alternative den Vorzug erhalten.

Diese Ergebnisse entsprechen absolut den Erfahrungen aus bereits umgesetzten Cloud-Projekten. Je höher die Anforderungen an die Kontrolle über die Ausführungsumgebung sind, desto eher eignet sich ein lokaler Betrieb eines Softwaresystems. Je mehr die Vorteile der Cloud genutzt werden sollen, desto eher kommen Cloud-Plattform-Dienste entsprechend dem PaaS-Ansatz in Frage, wie dies im vorliegenden Beispiel in $Aalt_2$ mit Azure Web Apps und Azure SQL Database gegeben ist.

Dank der erstellten Entwurfs- und Bewertungsmodelle lässt sich dieser Zusammenhang bis zu den betreffenden Stakeholdern zurückverfolgen. Insbesondere das Produktmanagement hat ein Interesse an geringen Antwortzeiten und schneller Bereitstellung von Bugfixes. Dies ist durch die Business Driver bd_3 und bd_4 formuliert, aus denen sich letztlich die Qualitätsanforderungen und Szenarien ableiten. Dem gegenüber haben die Vertreter des Systembetriebs ein hohes Interesse an der Kontrolle über die Ausführungsumgebungen. Dies wiederum manifestiert sich im Business Driver bd_5, aus dem sich ebenfalls entsprechende Anforderungen und Szenarien ableiten. Somit ist folgende Aussage, welche sich in konkreten Kundenprojekten bestätigt, legitim: Je höher der Einfluss des Systembetriebs auf die Architekturdiskussion ist, desto höher die Wahrscheinlichkeit, dass eine lokale Ausführungsumgebung gewählt wird, über die der Systembetrieb volle Kontrolle hat. Je höher der relative Einfluss der Produktmanager ist, desto eher werden Cloud-Umgebungen gewählt, die Vorteile in der Flexibilität, Skalierbarkeit und Bereitstellungszeit bieten. Dank der in dieser Arbeit erstellten integrierten Entwurfs- und Bewertungsmethode können derartige Zusammenhänge zwischen Wünschen und Prioritäten der Stakeholder auf der einen und den gewählten Architekturalternativen auf der anderen Seite aufgedeckt werden.

7.3.3 Ergebnis der Durchführung

Im Rahmen dieses Evaluierungsfallbeispiels konnte der Leitfaden erfolgreich eingesetzt werden. Drei Architekturalternativen wurden hergeleitet und auf Basis von priorisierten Anforderungen gegenübergestellt. Dies ergab eine klare Empfehlung für eine Architekturalternative. Tabelle 7-23 zeigt, inwieweit dabei die Anforderungen an den Leitfaden erfüllt wurden.

#	Anforderung	Bewertung	Anforderung erfüllt
1	Formale Beschreibung der Anforderungen	Die Anforderungen wurden formal beschrieben.	Ja.
2	Formale Beschreibung der Anforderungen einer Komponente	Die Anforderungen einzelner Komponenten wurden formal beschrieben.	Ja.
3	Formale Beschreibung von Entwurfsalternativen	Entwurfsalternativen wurden formal beschrieben.	Ja.
4	Unterstützung hybrider Cloud-Architekturen	Es wurden drei hybride Cloud-Architekturen betrachtet.	Ja.
5	Bewertbarkeit von Entwurfsalternativen	Die Alternativen wurden bewertet.	Ja.
6	Vermeidung unnötigen Overheads	Nicht untersucht.	n/a
7	Ein- und Ausgaben des Gesamtprozesses	Ein- und Ausgaben wurden beschrieben.	Ja.
8	Modellierbarkeit von Architekturmustern	Architekturmuster wurden zur Begrenzung der Alternativen auf drei Architekturalternativen verwendet.	Ja.
9	Einmalige Ausführung von Entwurfs- und Bewertungsschritten	Priorisierungen wurden nur einmal vorgenommen.	Ja.
10	Zusammenfassung von Arbeitsschritten einzelner Stakeholder	Stakeholder mussten nur zu Beginn und am Ende hinzugezogen werden.	Ja.
11	Regelmäßige Plausibilitätsprüfungen	In den einzelnen Bewertungsschritten wurden Ergebnisse bewertet.	Ja.
12	Minimierung der Anzahl an Arbeitsschritten	Keine Wiederholung von Prozessschritten.	Ja.

Tabelle 7-23: Beurteilung der Anwendung des Leitfadens in Fallbeispiel 2

7.4 Fallbeispiel 3: Cloud-basierte CAD-Anwendung

Dieses Fallbeispiel stammt aus einem konkreten Kundenprojekt. Mit dem Kunden, einem Hersteller einer als Rich Client implementierten CAD-Anwendung, wurden verschiedene Alternativen für die Weiterentwicklung seiner Software diskutiert. Diese sollte, so der Wunsch des Kunden, „Cloud-ready" gemacht werden, wobei erst in der Architekturdiskussion herausgearbeitet wurde, was im konkreten Fall genau unter diesem Begriff zu verstehen war und wie dieses Attribut in den einzelnen Alternativen ausgeprägt war.

7.4.1 Beschreibung des Fallbeispiels

Die zu analysierende Software war eine Anwendung im Bereich Computer Aided Design (CAD). Über einen Editor konnten Schaltkästen, elektronische Schaltungen etc. entworfen und geprüft werden. Über eine Symbol-Bibliotheks-Komponente, konnten Symbole mit entsprechenden Meta-Informationen über einen zentral bereitgestellten Download-Service heruntergeladen und in die Anwendung bzw. bestehende Zeichnungen importiert werden. Mit Hilfe einer Berechnungskomponente konnten Zeichnungen validiert sowie verschiedene Darstellungen (z.B. 3D-Renderings) für einen Schaltungsentwurf erzeugt werden. Nicht zuletzt aus Gründen der besseren Vermarktbarkeit hatte der Kunde den Wunsch, die Anwendung „Cloud-ready" zu machen, d.h. im Rahmen der Weiterentwicklung der Software auch auf Cloud Technologien zu setzen.

7.4.2 Anwendung des Leitfadens

Im Rahmen der Architekturdiskussion konnte der Entwurfs- und Bewertungsmethode als Leitfaden für die zu diskutierenden Aspekte der Software genutzt werden. Die folgenden Abschnitte zeigen auf, welche Ergebnisse in den einzelnen Schritten erzielt werden konnten.

7.4.2.1 Schritt 1: Präsentation der Vorgehensweise

Dieser Schritt wurde nur sehr kurz abgehandelt. Die Vorgehensweise wurde vorgestellt und die Agenda für den Workshop zur Architekturdiskussion entsprechend ausgestaltet.

7.4.2.2 Schritt 2: Erstellung eines Business Modells

An der Architekturdiskussion beteiligt waren Vertreter der Geschäftsführung sowie der Entwicklungsleiter und ein Produktmanager. Einigkeit bestand darin, dass die Software aus Anwendersicht im Rahmen der kurzfristigen Weiterentwicklung weitestgehend unverändert bleiben sollte. Ausnahmen hiervon waren die Symbolbibliotheks- und die Berechnungskomponente. Für diese sollten Alternativen durchgesprochen und jeweils bewertet werden, wie

mit dadurch das Prädikat „Cloud-ready“ erhalten werden könnte. Aus technischer Sicht sollten weiterhin .NET-Technologien eingesetzt werden, da die Entwickler hier bereits Erfahrungen hatten. Entsprechend wurden die folgenden Business Driver identifiziert.

Id	Business Driver	Stakeholder	Beschreibung
bd_1	Beibehaltung der bestehenden Funktionalität (Ausnahme: Symbol-Bibliotheks- und Berechnungskomponente)	Geschäftsführung	Weiterentwicklung nur in der Komponente, mit der neue Symbole in die Anwendung geladen werden können und in der Berechnungskomponente für 3D-Renderings.
bd_2	Erlangung des Prädikats „Cloud-ready“	Geschäftsführung	Erweiterung der Anwendung, um sie entsprechend als „modern“ vermarkten zu können.
bd_3	Minimaler Trainingsaufwand für Entwickler	Entwicklungsleitung	Beibehaltung der eingesetzten Technologien (C#/.NET).
bd_4	Beibehaltung der User Experience für Anwender	Produktmanagement	Beibehaltung der grundsätzlichen Funktionalität und insbesondere der Performance (gute Antwortzeiten auf Benutzerinteraktion).

Tabelle 7-24: Business Driver mit Stakeholdern (Fallbeispiel 3)

Im Fokus für die weiteren Betrachtungen standen somit die Symbolbibliotheks- und die Berechnungskomponente. Auch eine vollständige Migration in die Cloud sollte betrachtet werden. Als Cloud Plattform wurde Microsoft Azure gesetzt, da diese unter den gegebenen Voraussetzungen (Entwickler-Know-How, Möglichkeit zu einer 1:1-Migration der Anwendung) als bestgeeignet erschien.

7.4.2.3 Schritt 3: Ableitung und Priorisierung der Anforderungen

Im nächsten Schritt wurden die Anforderungen abgeleitet. Tabelle 7-25 zeigt das Anforderungsmodell in vereinfachter Form.

Id	Typ	Beschreibung	Business Driver
fr_1^{init}	fachlich	Beibehaltung der bestehenden Funktionalität	bd_1
qr_1^{init}	qualitativ	„Cloud-ready“-Prädikat der Gesamtanwendung	bd_2
qr_2^{init}	qualitativ	Performanz der Gesamtanwendung	bd_4
dc_1^{init}	Einschränkung	Beibehaltung des grundlegenden Architekturaufbaus	bd_2
dc_2^{init}	Einschränkung	Beibehaltung der Technologien	bd_3

Tabelle 7-25: Initiales Anforderungsmodell (Fallbeispiel 3)

Nachdem die Grundfunktionalität durch die bestehende Anwendung vorgegeben war, fokussierten sich die weiteren Betrachtungen auf die beiden Bereiche „Cloud-ready“ und – stellvertretend für den Bereich User Experience – die Performanz. Wie bereits im 3. Schritt in Fallbeispiel 2 (siehe 7.3.2.3) wurden diese Anforderungen zunächst mit der Gesamtanwendung in Beziehung gesetzt und im initialen Anforderungsmodell entsprechend erfasst.

Die fachliche Anforderung fr_1^{init} sowie die Entwurfseinschränkungen dc_1^{init} und dc_2^{init} repräsentierten den Wunsch nach Beibehaltung der Funktionalität, der Grund-Architektur und der eingesetzten Technologien. Im Rahmen der weiteren Betrachtungen sollte nun untersucht werden, welche Teile der Anwendung in der Cloud ausgeführt werden konnten, und welche Auswirkungen dies auf die einzelnen Anforderungen haben würde.

7.4.2.4 Schritt 4: Entwurf der Softwarearchitektur

Aus diesen Anforderungen wurde ein grober Architekturentwurf erstellt. Dessen grundsätzlicher Aufbau war durch die Business Driver vorgegeben. Dieser ist in Abb. 7-3 dargestellt.

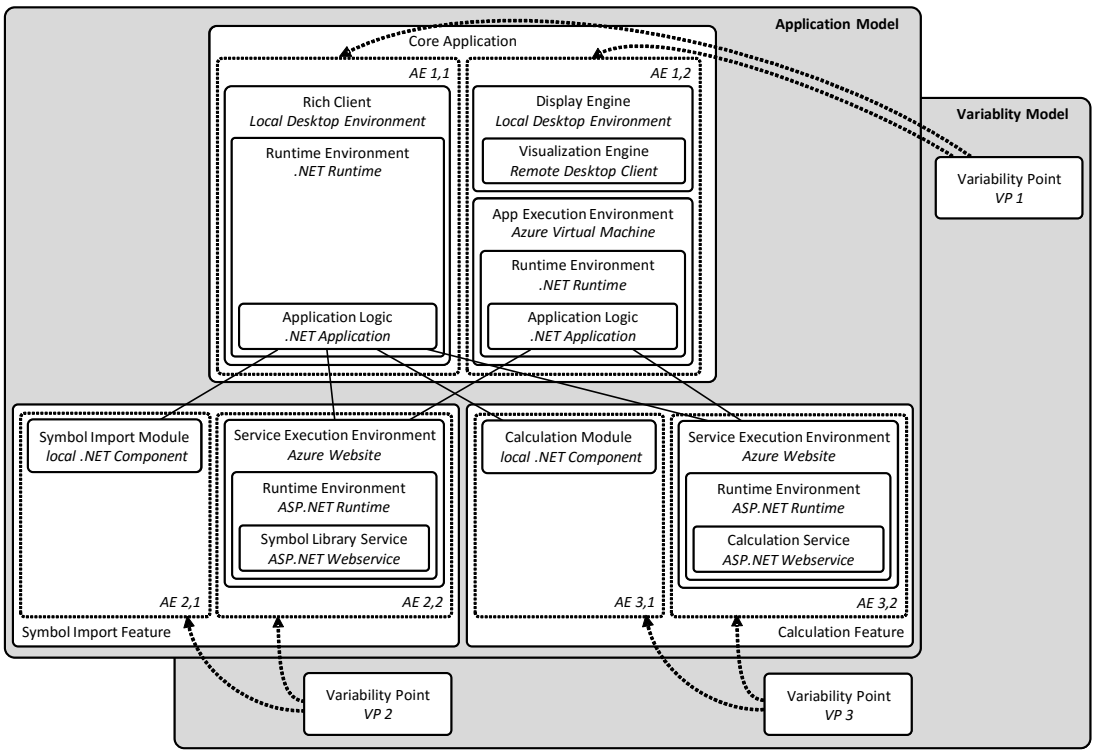

Abb. 7-3 Anwendungs- und Variabilitätsmodell der CAD-Anwendung

Interessant für die Diskussion des Entwurfs waren die drei Variabilitätspunkte, die jeweils für einen von drei Bereichen der Architektur jeweils eine lokal betriebene und eine Cloud-Alternative in Form entsprechender Architekturentscheidungen gegenüberstellten. Variabilitätspunkt vp_1 beschrieb die Gegenüberstellung für die Kern-Anwendung, vp_2 für den Symbolbibliothek-Service und vp_3 für den Berechnungsservice. Tabelle 7-26 gibt einen Überblick über die Variabilitätspunkte mit ihren alternativen Architekturentscheidungen.

Variabilitäts-punkt	**Beschreibung**	**Architektur-entscheidung**	**Beschreibung**
vp_1	Ausführung der Anwendungslogik	$ae_{1,1}$	Ausführung in lokaler Umgebung
		$ae_{1,2}$	Ausführung in einer Azure Virtual Machine (Cloud)
vp_2	Ausführung des Symbol-Bibliothek-Service	$ae_{2,1}$	Ausführung in lokaler Umgebung
		$ae_{2,2}$	Ausführung in einer Azure Web App (Cloud)
vp_3	Ausführung des Berechnungsservice	$ae_{3,1}$	Ausführung in lokaler Umgebung
		$ae_{3,2}$	Ausführung in einer Azure Web App (Cloud)

Tabelle 7-26: Variabilitätspunkte und Architekturentscheidungen (Fallbeispiel 3)

Nicht alle Kombinationen der dort gezeigten Architekturentscheidungen wurden weiter betrachtet. So wurde eine Verlagerung der Kern-Anwendungslogik in die Cloud bei gleichzeitig lokalem Betrieb von Symbolbibliothek- und Berechnungsservice als unsinnig angesehen. Vier Architekturalternativen wurden herausgearbeitet. Diese sind in Tabelle 7-27 aufgelistet.

Architektur-alternative	**Tupel zugehöriger Architekturentscheidungen**	**Beschreibung**
$Aalt_1$	$(ae_{1,1}, ae_{2,1}, ae_{3,1})$	Vollständiger Betrieb der Anwendung in der lokalen Ausführungsumgebung (Client PC)
$Aalt_2$	$(ae_{1,1}, ae_{2,2}, ae_{3,1})$	Auslagerung des Symbol-Bibliothek-Service in die Cloud (Azure Web App)
$Aalt_3$	$(ae_{1,1}, ae_{2,2}, ae_{3,2})$	Auslagerung sowohl des Symbol-Bibliothek-Service als auch des Berechnungsservice in die Cloud (jeweils Azure Web App)
$Aalt_4$	$(ae_{1,2}, ae_{2,2}, ae_{3,2})$	Vollständiger Betrieb der Anwendung in der Cloud und Anbindung via Remote Desktop (Terminal Service)

Tabelle 7-27: Zu bewertende Architekturalternativen (Fallbeispiel 3)

Zum besseren Verständnis sind die vier Architekturalternativen nochmals vereinfacht in Abb. 7-4 dargestellt. Die Alternative, nur den Berechnungsservice in die Cloud auszulagern, wurde vom Kunden ausgeschlossen.

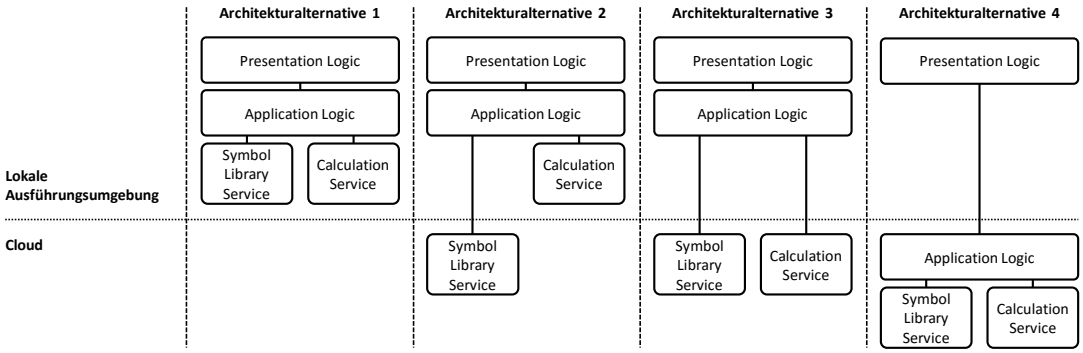

Abb. 7-4 Architekturalternativen der CAD-Anwendung

Im Rahmen des Architekturentwurfs wurden die initialen Anforderungen auf die Komponenten Kern-Anwendung, den Symbolbibliothek- und den Berechnungsservice heruntergebrochen. Somit ergab sich folgendes finales Anforderungsmodell.

Id	Typ	Beschreibung	Business Driver
fr_1	fachlich	Beibehaltung der bestehenden Funktionalität	bd_1
qr_1	qualitativ	„Cloud-ready"-Prädikat der Anwendung	bd_2
qr_2	qualitativ	„Cloud-ready"-Prädikat des Symbol-Bibliothek-Service	bd_2
qr_3	qualitativ	„Cloud-ready"-Prädikat des Berechnungsservice	bd_2
qr_4	qualitativ	Performanz der Anwendung	bd_4
qr_5	qualitativ	Performanz des Symbol-Bibliothek-Service	bd_4
qr_6	qualitativ	Performanz des Berechnungsservice	bd_4
dc_1	Einschränkung	Beibehaltung des grundlegenden Architekturaufbaus	bd_2
dc_2	Einschränkung	Beibehaltung der Technologien	bd_3

Tabelle 7-28: Finales Anforderungsmodell mit Business Drivern (Fallbeispiel 3)

7.4.2.5 Schritt 5: Ableitung und Priorisierung von Szenarien

Für die sechs Qualitätsanforderungen wurden Szenarien entwickelt und diese priorisiert. Dabei wurden die Bereitstellungszeit für Bugfixes, Skalierbarkeit zu Peak-Zeiten und Performanz betrachtet. Für die Kern-Anwendung wurde die Skalierbarkeit als wenig wichtig angesehen, die Bereitstellungszeit für Bug-Fixes als wichtiger. Für den Symbolbibliothek-Service

wurde die Skalierbarkeit höher priorisiert, da hier bei schlechter Performanz negative Auswirkungen auf die User Experience befürchtet wurden. Bereitstellungszeiten für Bugfixes sowie neue Symbole wurden ebenfalls wichtig eingestuft. Beim Berechnungsservice wurde die Skalierbarkeit als sehr wichtig angesehen, da hier Wettbewerbsvorteile bei entsprechend performanten Berechnungen angepeilt wurden. In diesem Bereich spielte auch die Möglichkeit eine Rolle, mehrere Performanz-Stufen für unterschiedliche Kundengruppen anbieten zu können (Silber für kostengünstige, mittlere Berechnungsgeschwindigkeit und Gold für höherpreisige, dafür performantere Berechnung). Als sehr wichtig für die Anwendung wurde auch die Reaktion der Benutzeroberfläche auf Benutzerinteraktion gesehen. Diese trägt entscheidend zur Anwenderzufriedenheit bei. Es ergab sich folgendes Szenarienmodell:

Id	Szenario	Priorität	Anforderung
$scen_{1,1}$	Bereitstellungszeit für Bugfixes für die Anwendung	6	qr_1
$scen_{1,2}$	Skalierbarkeit zu Peak-Zeiten um zusätzliche Instanzen	1	qr_1
$scen_{2,1}$	Bereitstellungszeit für Bugfixes für den Symbol-Bibliothek-Service	8	qr_2
$scen_{2,2}$	Bereitstellungszeit für neue Symbole	8	qr_2
$scen_{2,3}$	Skalierbarkeit zu Peak-Zeiten um zusätzliche Instanzen	6	qr_2
$scen_{3,1}$	Bereitstellungszeit für Bugfixes für die Berechnungskomponente	8	qr_3
$scen_{3,2}$	Skalierbarkeit zu Peak-Zeiten um zusätzliche Instanzen	10	qr_3
$scen_{3,3}$	Verfügbarkeit verschiedener Leistungsstufen (Silber, Gold)	6	qr_3
$scen_{4,1}$	Reaktion UI auf Benutzereingaben	10	qr_4
$scen_{5,1}$	Schnelle Integrierbarkeit von Symbolen	5	qr_5
$scen_{6,1}$	Schnelle Verfügbarkeit von 3D-Renderings	8	qr_6

Tabelle 7-29: Priorisierte Szenarien (Fallbeispiel 3)

Die höchste Priorität wurde also auf der Performanz und Flexibilität des Berechnungsservice sowie die Reaktionszeit der Benutzeroberfläche gelegt.

7.4.2.6 Schritt 6: Bewertung der Softwarearchitektur

Für die Risikobetrachtung wurden die drei Variabilitätspunkte untersucht. Sehr schnell wurde deutlich, dass alle drei Sensitivity Points, Tradeoff Points und Risiken sind. Alle drei haben jeweils Einfluss auf den Erfüllungsgrad mehrerer Qualitätsanforderungen und besitzen Alternativen, die Anforderungen verletzen.

Element des Risikomodells	Elemente der Menge	Anmerkung
Sensitivity Points	vp_1, vp_2, vp_3	Alle Variabilitätspunkte sind Sensitivity Points, da alle Entscheidungen auf mindestens zwei Qualitätsanforderungen Auswirkungen haben.
Tradeoff Points	vp_1, vp_2, vp_3	Da die Architekturentscheidungen aller Variabilitätspunkte jeweils Auswirkungen auf mehr als eine Qualitätsanforderung haben (vp_1 auf qr_1 und qr_4 bzw. vp_2 auf qr_2 und qr_5 bzw. vp_3 auf qr_3 und qr_6), sind alle Variabilitätspunkte auch Tradeoff Points.
Risks	vp_1, vp_2, vp_3	In allen Variabilitätspunkten können Entscheidungen zur Verletzung von Anforderungen führen. Beim lokalen Betrieb von Symbol-Bibliothek- oder Berechnungsservice kann nicht skaliert werden, beim Cloud-Betrieb der Anwendung fallen ggf. Antwortzeiten zu groß aus.
Non-Risks	$\perp$	Alle Variabilitätspunkte wurden als Risks eingestuft.
sensitivities	$vp_1 \rightarrow qr_1$, $vp_1 \rightarrow qr_4$, $vp_2 \rightarrow qr_2$, $vp_2 \rightarrow qr_5$, $vp_3 \rightarrow qr_3$, $vp_3 \rightarrow qr_6$	

Tabelle 7-30: Elemente des Risikomodells (Fallbeispiel 3)

Für die einzelnen Szenarien wurden Zielerreichungsgrade ermittelt. Da im lokalen Betrieb keine Skalierung möglich ist, wurden der aktuelle sowie der schlechteste Zielerreichungsgrad leer gelassen.

		Zielerreichungsgrad			
Szenario	**Prioritäten**	**Worst**	**Current**	**Desired**	**Best**
$scen_{1,1}$	6	1 Tag	1 Tag	1 Stunde	5 Minuten
$scen_{1,2}$	1	n/a	n/a	20 Minuten	5 Minuten
$scen_{2,1}$	8	1 Tag	1 Tag	1 Stunde	5 Minuten
$scen_{2,2}$	8	15 Minuten	15 Minuten	5 Minuten	5 Minuten
$scen_{2,3}$	6	n/a	n/a	20 Minuten	5 Minuten
$scen_{3,1}$	8	1 Tag	1 Tag	1 Stunde	15 Minuten
$scen_{3,2}$	10	n/a	n/a	20 Minuten	5 Minuten
$scen_{3,3}$	6	Nein	Nein	Ja	Ja
$scen_{4,1}$	10	2 Sek	10 ms	10 ms	5 ms

		Zielerreichungsgrad			
Szenario	**Prioritäten**	**Worst**	**Current**	**Desired**	**Best**
$scen_{5,1}$	5	15 Minuten	15 Minuten	2 Minuten	2 Minuten
$scen_{6,1}$	8	30 Min.	5 Min.	30 Sek.	10 Sek.

Tabelle 7-31: Response Goals und Prioritäten bezogen auf die Szenarien (Fallbeispiel 3)

Diesen Zielerreichungsgraden wurden die in Tabelle 7-32 aufgelisteten Nutzengrade zugeordnet. Es wurde eine Skala von 0 bis 100 Wertpunkten gewählt. Bei den Ausprägungen, die die Anforderungen an Skalierbarkeit verletzten bzw. keine Leistungsstufen möglich machten, wurde der Nutzengrad auf 0 gesetzt.

		Nutzengrad			
Szenario	**Prioritäten**	**Worst**	**Current**	**Desired**	**Best**
$scen_{1,1}$	6	60	60	80	100
$scen_{1,2}$	1	0	0	80	100
$scen_{2,1}$	8	60	60	80	100
$scen_{2,2}$	8	60	60	100	100
$scen_{2,3}$	6	0	0	80	100
$scen_{3,1}$	8	60	60	80	100
$scen_{3,2}$	10	0	0	80	100
$scen_{3,3}$	6	0	0	100	100
$scen_{4,1}$	10	10	70	70	100
$scen_{5,1}$	5	60	60	100	100
$scen_{6,1}$	8	20	40	80	100

Tabelle 7-32: Nutzengrade bezogen auf die Szenarien (Fallbeispiel 3)

Als nächstes wurden für die einzelnen Architekturentscheidungen und die zugehörigen Szenarien die erwarteten Zielerreichungen abgeschätzt und daraus die erwarteten Nutzengrade interpoliert. Mit den entsprechenden Prioritäten der einzelnen Szenarien wurden dann die gewichteten Nutzengrade berechnet.

Architektur-entscheidung	**Betroffenes Szenario**	**Erwartete Zielerreichung**	**Erwarteter Nutzengrad**	**Priorität des Szenarios**	**Gewichteter Nutzengrad**
$ae_{1,1}$	$scen_{1,1}$	1 Tag	60	6	360
	$scen_{1,2}$	n/a	0	1	0
	$scen_{4,1}$	10 ms	70	10	700

Architektur-entscheidung	Betroffenes Szenario	Erwartete Zielerreichung	Erwarteter Nutzengrad	Priorität des Szenarios	Gewichteter Nutzengrad
$ae_{1,2}$	$scen_{1,1}$	30 Minuten	90	6	540
	$scen_{1,2}$	15 Minuten	85	1	85
	$scen_{4,1}$	2 Sekunden	10	10	100
$ae_{2,1}$	$scen_{2,1}$	1 Tag	60	8	480
	$scen_{2,2}$	15 Minuten	60	8	480
	$scen_{2,3}$	n/a	0	6	0
	$scen_{5,1}$	15 Minuten	60	5	300
$ae_{2,2}$	$scen_{2,1}$	5 Minuten	100	8	800
	$scen_{2,2}$	5 Minuten	100	8	800
	$scen_{2,3}$	15 Minuten	85	6	510
	$scen_{5,1}$	2 Minuten	100	5	500
$ae_{3,1}$	$scen_{3,1}$	1 Tag	60	8	480
	$scen_{3,2}$	n/a	0	10	0
	$scen_{3,3}$	Nein	0	6	0
	$scen_{6,1}$	5 Minuten	40	8	320
$ae_{3,2}$	$scen_{3,1}$	5 Minuten	100	8	800
	$scen_{3,2}$	15 Minuten	85	10	850
	$scen_{3,3}$	Ja	100	6	600
	$scen_{6,1}$	30 Sekunden	80	8	640

Tabelle 7-33: Erwartete Zielerreichung für die Szenarien (Fallbeispiel 3)

Im Anschluss an die Nutzenbetrachtung wurden Schätzungen für die zu erwartenden Kosten der einzelnen Alternativen angestellt.

Architektur-entscheidung	Kosten für die Implementierung	Kosten für den Betrieb (1 Jahr)	Kosten der Architekturentscheidung
$ae_{1,1}$	50	0	50
$ae_{1,2}$	50	300	350
$ae_{2,1}$	50	50	100
$ae_{2,2}$	100	50	150
$ae_{3,1}$	200	0	200
$ae_{3,2}$	200	50	250

Tabelle 7-34: Kosten für die Architekturentscheidungen (Fallbeispiel 3)

Die Kosten wurden dabei zum einen für Implementierungsaufwände zum anderen für den auf ein Jahr angesetzten Betrieb abgeschätzt. Anstelle konkreter Preise wurden auch hier Kostengrößen für eine relative Abschätzung verwendet. Die Abschätzungen sind in Tabelle 7-34 zu sehen.

Beim lokalen Betrieb fielen für die Anwendung und die Berechnungskomponente keine Kosten an. Da beim lokalen Betrieb des Symbolbibliothek-Service auch eine Online-Bibliothek vorgehalten werden musste, aus der Anwender Symbole laden konnten, mussten Kosten angesetzt werden. Höhere Kosten fielen beim Cloud-Betrieb der Kern-Anwendung an, da hier die vergleichsweise teuren Azure Virtual Machines zum Einsatz kamen. Damit lagen alle Daten vor, um für die Architekturentscheidungen die Return-on-Investments zu berechnen.

Architektur-entscheidung	**Gewichtete Nutzengrade**	**Gesamtnutzen der AE**	**Kosten der AE**	**ROI der AE**
$ae_{1,1}$	360 + 0 + 700	1060	50	21,20
$ae_{1,2}$	540 + 85 +100	725	350	2,07
$ae_{2,1}$	480 + 480 + 0 + 300	1260	100	12,60
$ae_{2,2}$	800 + 800 + 510 + 500	2610	150	17,40
$ae_{3,1}$	480 + 0 + 0 + 320	800	200	4,00
$ae_{3,2}$	800 + 850 + 600 + 640	2890	250	11,56

Tabelle 7-35: Berechnung des ROI für die Architekturentscheidungen (Fallbeispiel 3)

Bereits hier war erkennbar, dass der Return-on-Investment für einen Cloud-Betrieb der Kern-Anwendung sehr niedrig ausfallen würde. Dies war zum einen bedingt durch den primär wegen höherer Antwortzeiten als schlechter eingestuften Gesamtnutzen zum anderen wegen der höheren Kosten. Entscheidung $ae_{1,1}$ schneidet somit bereits deutlich besser als Entscheidung $ae_{1,2}$ ab. Umgekehrtes gilt für den Berechnungsservice, da zum einen der Nutzen der Cloud-Variante aufgrund viel besserer Skalierungsmöglichkeiten und der Möglichkeit von Leistungsstufendeutlich besser bewertet wurde, zum anderen die Kosten dieser Variante nicht entscheidend höher angesetzt wurden.

Architektur-alternative	**Nutzengrade der betreffenden Architekturentscheidungen**	**Gesamtnutzen der AAlt**	**Kosten der AAlt**	**ROI der AAlt**
$Aalt_1$	1060 + 1260 + 800	3120	350	8,91
$Aalt_2$	1060 + 2270 + 800	4470	400	11,18
$Aalt_3$	1060 + 2270 + 2890	6560	450	14,58
$Aalt_4$	725 + 2270 + 2890	6225	750	8,30

Tabelle 7-36: Berechnung des ROI für die Architekturalternativen (Fallbeispiel 3)

Auf Basis dieser Werte für die einzelnen Entscheidungen konnten nun die Werte für die vier Architekturalternativen bestimmt werden. Diese sind in Tabelle 7-36 zu sehen.

Den höchsten ROI-Wert konnte die Alternative mit lokalem Betrieb der Kern-Anwendung und Cloud-Betrieb von Symbolbibliothek- und Berechnungsservice erzielen. Dies erschien unmittelbar plausibel. Diese Alternative kombiniert die guten Antwortzeiten der Benutzeroberfläche, die sehr hoch priorisiert wurden, mit den Möglichkeiten der Cloud (Skalierbarkeit, „Cloud ready"-Attribut, etc.), die ebenfalls als strategisch wichtig betrachtet wurden.

7.4.2.7 Schritt 7: Präsentation der Entwurfs- und Bewertungsergebnisse

Im letzten Schritt wurden die Ergebnisse der Entwurfs- und Bewertungsaktivitäten wie folgt zusammengefasst:

- Alle drei Variabilitätspunkte wurden als Risiko eingestuft. Die Entscheidungen dieser Punkte hatten also alle ihre Für und Wider. Im Bereich der Kern-Anwendung war eine Entscheidung für die Cloud problematisch für die hoch priorisierten Antwortzeiten. Beim Symbolbibliothek- und Berechnungsservice hätte ein lokaler Betrieb den Verzicht auf einige als strategisch wichtig angesehene Vorteile der Cloud bedeutet.

- Die Architekturalternative $Aalt_3$ wurde als die Alternative mit dem höchsten Return-on-Investment identifiziert. Es wurde also eine Empfehlung für eine Beibehaltung des lokalen Betriebs der Kern-Anwendung auf den Anwender-PCs sowie für eine Verlagerung der beiden Services in die Cloud ausgesprochen.

- Zweitbeste Alternative war $Aalt_2$ mit einem lokalen Verbleib des Berechnungsservice und Auslagerung des Symbolbibliothek-Service in die Cloud. Tatsächlich entschied sich das Projektteam für ein schrittweises Vorgehen, bei dem zunächst $Aalt_2$ und später $Aalt_3$ umgesetzt werden sollte. Zunächst sollte also der Service für die Symbolbibliothek und im zweiten Schritt der Berechnungsservice in die Cloud verlagert werden.

- Eine vollständige Migration der Anwendung in die Cloud, also Alternative $Aalt_4$ wurde in Bezug auf den Return-on-Investment als schlechteste Variante, demnach sogar schlechter als die aktuelle Umsetzung $Aalt_1$, eingestuft und deshalb verworfen. Grund war, dass die Antwortzeiten der Benutzerschnittstelle bei einer Verlagerung der Kern-Anwendung in die Cloud inakzeptabel hoch geworden wären und dies die Vorteile bei der Skalierbarkeit und Flexibilität bei der Bereitstellung von Bugfixes zunichte gemacht hätten.

7.4.3 Ergebnis der Durchführung

Der Leitfaden konnte im Rahmen dieses Evaluierungsfallbeispiels erfolgreich eingesetzt werden. Es wurden mehrere Architekturalternativen hergeleitet und auf Basis von Anforderungen bewertet. Dies ergab eine klare Empfehlung für eine Architekturalternative. Tabelle 7-37 zeigt, inwieweit dabei die Anforderungen an den Leitfaden erfüllt wurden.

#	Anforderung	Bewertung	Anforderung erfüllt
1	Formale Beschreibung der Anforderungen	Die Anforderungen wurden formal beschrieben.	Ja.
2	Formale Beschreibung der Anforderungen einer Komponente	Die Anforderungen einzelner Komponenten wurden formal beschrieben.	Ja.
3	Formale Beschreibung von Entwurfsalternativen	Entwurfsalternativen wurden formal beschrieben.	Ja.
4	Unterstützung hybrider Cloud-Architekturen	Es wurden drei hybride Cloud-Architekturen betrachtet.	Ja.
5	Bewertbarkeit von Entwurfsalternativen	Die Alternativen wurden bewertet.	Ja.
6	Vermeidung unnötigen Overheads	Nicht untersucht.	n/a
7	Ein- und Ausgaben des Gesamtprozesses	Ein- und Ausgaben wurden beschrieben.	Ja.
8	Modellierbarkeit von Architekturmustern	Nicht untersucht.	n/a
9	Einmalige Ausführung von Entwurfs- und Bewertungsschritten	Priorisierungen wurden nur einmal vorgenommen.	Ja.
10	Zusammenfassung von Arbeitsschritten einzelner Stakeholder	Stakeholder mussten nur zu Beginn und am Ende hinzugezogen werden.	Ja.
11	Regelmäßige Plausibilitätsprüfungen	In den einzelnen Bewertungsschritten wurden Ergebnisse bewertet.	Ja.
12	Minimierung der Anzahl an Arbeitsschritten	Keine Wiederholung von Prozessschritten.	Ja.

Tabelle 7-37: Beurteilung der Anwendung des Leitfadens in Fallbeispiel 3

7.5 Fallbeispiel 4: Cloud-Migration eines bestehenden Webservice

In diesem konkreten Kundenszenario bestand die Aufgabe darin, zu analysieren, in welchem Aufwand-Nutzen-Verhältnis verschiedene Migrationsoptionen für einen bestehenden Webservice stehen. Daraus sollte abgeleitet werden, ob eine Migration in die Cloud sinnvoll und, wenn ja, ob bei der Migration grundlegende Änderungen am Service vorgenommen werden sollten.

7.5.1 Beschreibung des Fallbeispiels

Der Kunde in diesem Fallbeispiel war ein Unternehmen aus der Zulieferindustrie für den Fahrzeug- und Maschinenbau. Der zu migrierende Webservice stellte Katalog- und Recherchefunktion für eine Anwendungssoftware bereit, die wiederum als Rich Client implementiert war. Über den Webservice konnten Bauteile für Maschinenplanung nachgeschlagen und deren Daten in die Planungsdatei übernommen werden. Die Einbettung des Webservice erfolgte über ein Web-Control in einer Windows-Anwendung, d.h. der Webservice war für die Darstellung von Eingabeformular, das Nachschlagen in einer Bauteile-Datenbank und die Präsentation der Suchergebnisse verantwortlich. Durch die Implementierung als Webservice konnten die Recherche-Funktionalität und die zugehörige Datenbasis zentral verwaltet und aktualisiert werden ohne Änderungen an der Client-Anwendung vornehmen zu müssen. Zum Zeitpunkt der Architekturanalyse wurde der Webservice an einem Ort zentral betrieben. Die Überlegungen zu einer Migration in die Cloud waren motiviert von dem Wunsch, eine weltweit wachsende Anwendergemeinde auch von weltweit verteilten Rechenzentrumsstandorten bedienen zu können. Insbesondere eine Expansion in die Volksrepublik China war treibender Faktor.

7.5.2 Anwendung des Leitfadens

Die folgenden Abschnitte zeigen auf, wie über Anwendung des Leitfadens verschiedene Umsetzungsszenarien erstellt und miteinander hinsichtlich ihrer Aufwand-Nutzen-Verhältnisse verglichen werden konnten.

7.5.2.1 Schritt 1: Präsentation der Vorgehensweise

Eine eingehende Betrachtung dieses Schrittes soll hier unterbleiben. Die Vorgehensweise wurde im Projekt kurz vorgestellt.

7.5.2.2 Schritt 2: Erstellung eines Business Modells

An der Diskussion zum Architekturentwurf waren neben einem Software-Architekten von Unternehmensseite Vertreter der Geschäftsführung, der Entwicklungsleitung und der Anwender beteiligt. Motivation für die Architekturdiskussion war der Wunsch nach einer weltweiten Expansion des angebotenen Webservice, wobei dessen Funktionalität und User Experience weitgehend unverändert bleiben sollte. Aus technischer Sicht sollte die Service-Implementierung weitgehend unverändert bleiben. Entsprechend wurden folgende Business Driver formuliert:

Id	Business Driver	Stakeholder	Beschreibung
bd_1	Weltweite Skalierbarkeit	Geschäftsführung	Möglichkeit zur weltweiten Expansion (insbesondere in die Volksrepublik China).
bd_2	Minimaler Migrationsaufwand	Entwicklungs-leitung	Beibehaltung der grundlegenden Architektur (2-Schichten-Architektur).
bd_3	Minimaler Trainingsaufwand für Entwickler	Entwicklungs-leitung	Beibehaltung der eingesetzten Technologien (PHP und SQL-basiertes RDBMS).
bd_4	Beibehaltung der User Experience	Anwender	Beibehaltung der Funktionalität und Performance des Webservice.

Tabelle 7-38: Business Driver mit Stakeholdern (Fallbeispiel 4)

Somit stand eine Migration des bestehenden Webservice in eine Cloud Umgebung im Mittelpunkt der weiteren Betrachtungen. Als Cloud Plattform wurde Microsoft Azure gewählt.

7.5.2.3 Schritt 3: Ableitung und Priorisierung der Anforderungen

Das Anforderungsmodell konnte relativ einfach gehalten werden. Die fachlichen Anforderungen waren durch das bestehende System gesetzt. Die zu betrachtenden Architekturalternativen mussten nach bd_4 dessen Funktionalität ebenfalls abbilden.

Id	Typ	Beschreibung	Business Driver
fr_1^{init}	fachlich	Beibehaltung der bestehenden Funktionalität	bd_4
qr_1^{init}	qualitativ	Skalierbarkeit der Gesamtanwendung	bd_1
qr_2^{init}	qualitativ	Performance der Gesamtanwendung	bd_4
dc_1^{init}	Einschränkung	Beibehaltung der 2-Schichten Architektur	bd_2
dc_2^{init}	Einschränkung	Beibehaltung der eingesetzten Technologien	bd_3

Tabelle 7-39: Finales Anforderungsmodell mit Business Drivern (Fallbeispiel 4)

Als Qualitätsanforderungen wurden Skalierbarkeit und Performanz gesetzt. Diese wurden im initialen Anforderungsmodell der Gesamtanwendung zugewiesen, welches in Tabelle 7-39 zu sehen ist.

7.5.2.4 Schritt 4: Entwurf der Softwarearchitektur

Auf Basis dieses Anforderungsmodells wurde ein Architekturentwurf erstellt. Letztlich waren aber weniger der grundlegende Aufbau der Anwendung interessant, da durch bd_2 und bd_3 ja der 2-Schicht-Aufbau und die Technologien vorgegeben waren. Interessanter waren die Alternativen für die Umsetzung der beiden Schichten. Der bestehenden Umsetzung (d.h. Ausführung in einer virtuellen Maschine eines Hosters) wurden alternative Ausführungsumgebungen im Microsoft Azure gegenübergestellt. Abb. 7-5 skizziert den Architekturentwurf. Aus Gründen der grafischen Darstellbarkeit wurden nicht Variabilitäten in einen einzigen Entwurf eingezeichnet sondern die vier herausgearbeiteten Architekturalternativen getrennt dargestellt.

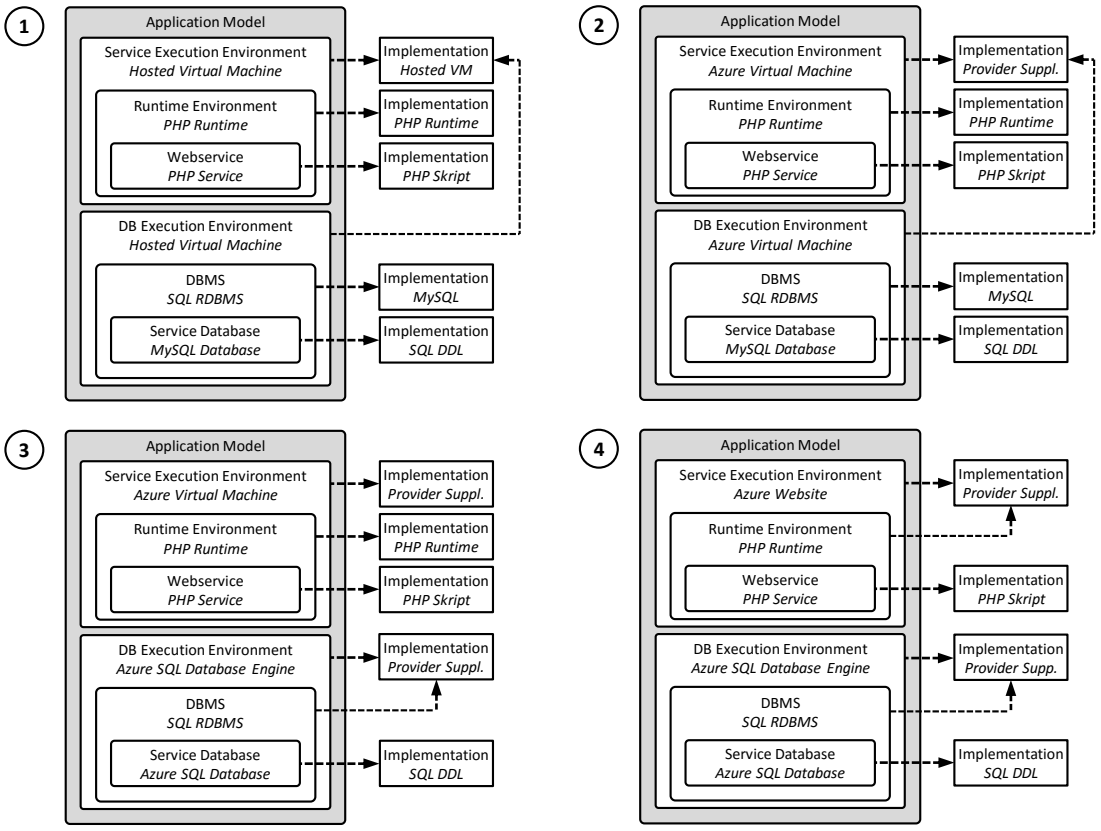

Abb. 7-5 Anwendungsmodell des zu migrierenden Webservice

Variabilitätspunkte waren also in der Ausführung der Service-Logik und Ausführung der Datenbank-Engine. Für die die Service-Logik wurde neben der bestehenden Lösung (virtuelle Maschine bei einem Hoster) noch die Ausführung in einer Azure Virtual Machine sowie in einer Azure Web App betrachtet. Für die Datenbank wurden die bestehende Umgebung, Betrieb in einer Azure Virtual Machine sowie Umsetzung in einer Azure SQL Database analysiert.

Variabilitäts-punkt	**Beschreibung**	**Architektur-entscheidung**	**Beschreibung**
vp_1	Ausführung der Service-Logik	$ae_{1,1}$	Virtuelle Maschine bei einem Hoster
		$ae_{1,2}$	Azure Virtual Machine
		$ae_{1,3}$	Azure Web App
vp_2	Ausführung der Datenbank-Engine	$ae_{2,1}$	Virtuelle Maschine bei einem Hoster
		$ae_{2,2}$	Azure Virtual Machine
		$ae_{2,3}$	Azure SQL Database

Tabelle 7-40: Variabilitätspunkte und Architekturentscheidungen (Fallbeispiel 4)

Tabelle 7-41 zeigt die Variabilitätspunkte mit den jeweiligen Architekturentscheidungen. Nicht alle Kombinationen dieser Entscheidungen sollten weiter betrachtet werden. So wurde eine Ausführung der Service-Logik beim Hoster und Betrieb der Datenbank in Azure verworfen, da Service-Logik und Datenbank in der gleichen Umgebung betrieben werden sollten. Somit ergaben sich folgende Architekturalternativen, die in den Folgeschritten näher untersucht werden sollten.

Architektur-alternative	**Tupel zugehöriger Architekturentscheidungen**	**Beschreibung**
$Aalt_1$	$(ae_{1,1}, ae_{2,1})$	Betrieb von PHP und MySQL in einer gehosteten virtuellen Maschine.
$Aalt_2$	$(ae_{1,2}, ae_{2,2})$	Betrieb von PHP und MySQL in einer Azure Virtual Maschine.
$Aalt_3$	$(ae_{1,2}, ae_{2,3})$	Betrieb von PHP in einer Azure VM und Azure SQL Database als Datenbanksystem.
$Aalt_4$	$(ae_{1,3}, ae_{2,3})$	Betrieb von PHP über eine Azure Web App und Azure SQL Database als Datenbanksystem.

Tabelle 7-41: Zu bewertende Architekturalternativen (Fallbeispiel 4)

Da nach bd_2 auch die grundsätzliche Architektur beibehalten werden sollte, konnten die Anforderungen Skalierbarkeit und Performanz, die im initialen Anforderungsmodell zunächst

mit der Gesamtanwendung verknüpft waren, im Rahmen des Architekturentwurfs auf die Anwendungs- und Datenbank-Schicht heruntergebrochen werden. Entsprechend ergab sich das in Tabelle 7-42 gezeigte finale Anforderungsmodell.

Id	Typ	Beschreibung	Business Driver
fr_1	fachlich	Beibehaltung der bestehenden Funktionalität	bd_4
qr_1	qualitativ	Skalierbarkeit der Anwendung	bd_1
qr_2	qualitativ	Performance der Anwendung	bd_4
qr_3	qualitativ	Skalierbarkeit der Datenbank	bd_1
qr_4	qualitativ	Performance der Datenbank	bd_4
dc_1	Einschränkung	Beibehaltung der 2-Schichten Architektur	bd_2
dc_2	Einschränkung	Beibehaltung der eingesetzten Technologien	bd_3

Tabelle 7-42: Finales Anforderungsmodell mit Business Drivern (Fallbeispiel 4)

7.5.2.5 Schritt 5: Ableitung und Priorisierung von Szenarien

Für die vier Qualitätsanforderungen wurden entsprechende Szenarien abgeleitet. Die Anforderung nach Skalierbarkeit wurde dabei heruntergebrochen in ein Szenario zur Skalierung eines Deployments (z.B. Hinzufügen zusätzlicher Instanzen) und in ein Szenario zum wiederholten Deployment in weitere Rechenzentren. Letzteres trägt dem Wunsch nach Expansion in weltweit verteilte Standorte Rechnung. Die Anforderung nach Performanz wurde über Antwortzeiten konkretisiert.

Id	Szenario	Priorität	Anforderung
$scen_1$	Skalierung zu Peak-Zeiten um eine Instanz.	5	qr_1
$scen_2$	Re-Deployment der Anwendung in weiteres Rechenzentrum.	10	qr_1
$scen_3$	Geringe Antwortzeiten.	10	qr_2
$scen_4$	Erhöhung der DB-Leistung zu Peak-Zeiten um den Faktor 2.	5	qr_3
$scen_5$	Re-Deployment der Datenbank in weiteres Rechenzentrum	10	qr_3
$scen_6$	Geringe Antwortzeiten	10	qr_4

Tabelle 7-43: Priorisierte Szenarien (Fallbeispiel 4)

Performanz und Expansion in neue Standorte wurden im Vergleich zur Skalierbarkeit eines Deployments als wichtiger angesehen. Entsprechend wurden diese Szenarien höher priorisiert.

7.5.2.6 Schritt 6: Bewertung der Softwarearchitektur

Zur Risikobewertung mussten zwei Variabilitätspunkte untersucht werden. Beide hatten jeweils Einfluss auf zwei Qualitätsanforderungen. Die Aufstellung des Risikomodells hatte demnach folgendes Ergebnis.

Element des Risikomodells	**Elemente der Menge**	**Anmerkung**
Sensitivity Points	vp_1, vp_2	Die Architekturentscheidungen beider Variabilitätspunkte haben Auswirkungen auf bestimmte Qualitätsanforderungen.
Tradeoff Points	vp_1, vp_2	Da die Architekturentscheidungen beider Variabilitätspunkte jeweils Auswirkungen auf mehr als eine Qualitätsanforderung haben (vp_1 auf qr_1 und qr_2 bzw. vp_2 auf qr_3 und qr_4), sind beide Variabilitätspunkte auch Tradeoff Points.
Risks	vp_1, vp_2	Für beide Variabilitätspunkte gilt, dass bei einer Beibehaltung des Status quo (Betrieb beim Hoster) die Anforderung an die lastabhängige Skalierbarkeit verletzt wird.
Non-Risks	$\perp$	Beide Variabilitätspunkte wurden als Risks eingestuft.
sensitivities	$vp_1 \rightarrow qr_1$, $vp_1 \rightarrow qr_2$, $vp_2 \rightarrow qr_3$, $vp_2 \rightarrow qr_4$	

Tabelle 7-44: Elemente des Risikomodells (Fallbeispiel 4)

Für die sechs aus den Qualitätsanforderungen abgeleiteten Szenarien wurden im nächsten Schritt die in Tabelle 7-45 aufgelisteten Zielerreichungsgrade erstellt.

		Zielerreichungsgrad			
Szenario	**Prioritäten**	**Worst**	**Current**	**Desired**	**Best**
$scen_1$	5	n/a	n/a	20 Minuten	5 Minuten
$scen_2$	10	1 Monat	1 Monat	1 Stunde	10 Minuten
$scen_3$	10	5 Sek.	< 1 Sek.	< 1 Sek.	< 0,1 Sek.
$scen_4$	5	n/a	n/a	20 Minuten	5 Minuten
$scen_5$	10	1 Monat	1 Monat	1 Stunde	10 Minuten
$scen_6$	10	5 Sek.	< 1 Sek.	< 1 Sek.	< 0,1 Sek.

Tabelle 7-45: Response Goals und Prioritäten bezogen auf die Szenarien (Fallbeispiel 4)

Nachdem die bestehende Implementierung durch die feste Instanzgröße und –anzahl keine Skalierungsmöglichkeiten bot, wurden der aktuelle und der schlechteste Zielerreichungsgrad für $scen_1$ und $scen_4$ leer gelassen. Diesen Zielerreichungsgraden wurden folgende Nutzengrade zugeordnet. Auch hier wurde eine Skala von 0 bis 100 Wertpunkten gewählt. Bei den beiden Szenarien, die von der bestehenden Lösung verletzt wurden, wurde der Nutzengrad auf 0 gesetzt.

		Nutzengrad			
Szenario	**Prioritäten**	**Worst**	**Current**	**Desired**	**Best**
$scen_1$	5	0	0	80	100
$scen_2$	10	20	20	90	100
$scen_3$	10	40	80	80	100
$scen_4$	5	0	0	80	100
$scen_5$	10	20	20	90	100
$scen_6$	10	40	80	80	100

Tabelle 7-46: Nutzengrade bezogen auf die Szenarien (Fallbeispiel 4)

Im nächsten Schritt wurden für Alternativen, die einen Betrieb in Microsoft Azure vorsahen, zu erwartende Zielerreichungen mit entsprechend interpolierten Nutzengraden ermittelt. Zusammen mit den Prioritäten der einzelnen Szenarien die gewichteten Nutzengrade abgeleitet.

Architektur-entscheidung	**Betroffenes Szenario**	**Erwartete Zielerreichung**	**Erwarteter Nutzengrad**	**Priorität des Szenarios**	**Gewichteter Nutzengrad**
$ae_{1,1}$	$scen_1$	n/a	20	5	100
	$scen_2$	1 Monat	20	10	200
	$scen_3$	< 1 Sek.	80	10	800
$ae_{1,2}$	$scen_1$	15 Minuten	85	50	425
	$scen_2$	1 Tag	80	10	800
	$scen_3$	< 0,5 Sek.	85	10	850
$ae_{1,3}$	$scen_1$	5 Minuten	100	5	500
	$scen_2$	1 Stunde	90	10	900
	$scen_3$	< 0,5 Sek.	85	10	850
$ae_{2,1}$	$scen_4$	n/a	20	5	100
	$scen_5$	1 Monat	20	10	200
	$scen_6$	< 1 Sek.	80	10	800
$ae_{2,2}$	$scen_4$	15 Minuten	85	5	425
	$scen_5$	1 Tag	80	10	800

Architektur-entscheidung	Betroffenes Szenario	Erwartete Zielerreichung	Erwarteter Nutzengrad	Priorität des Szenarios	Gewichteter Nutzengrad
	$scen_6$	< 0,5 Sek.	85	10	850
$ae_{2,3}$	$scen_4$	10 Minuten	100	5	500
	$scen_5$	1 Stunde	90	10	900
	$scen_6$	< 0,5 Sek.	85	10	850

Tabelle 7-47: Erwartete Zielerreichung für die Szenarien (Fallbeispiel 4)

Damit war die Nutzenbetrachtung der einzelnen Architekturentscheidungen zunächst abgeschlossen. Es folgte eine Kostenabschätzung für die Umsetzung der einzelnen Entscheidungen. Die Kosten setzten sich dabei aus geschätzten Kosten für die Migration und Kosten für den Betrieb der betreffenden Alternative zusammen. Dabei wurden die Betriebskosten für ein Jahr gegenübergestellt.

Architektur-entscheidung	Kosten für die Migration	Kosten für den Betrieb (1 Jahr)	Kosten der Architekturentscheidung
$ae_{1,1}$	0	100	100
$ae_{1,2}$	20	80	100
$ae_{1,3}$	25	50	75
$ae_{2,1}$	0	100	100
$ae_{2,2}$	20	80	100
$ae_{2,3}$	60	50	110

Tabelle 7-48: Kosten für die Architekturentscheidungen (Fallbeispiel 4)

Bei einem Umzug nach Microsoft Azure wurden also durchweg Migrationskosten angesetzt. Beim Einsatz einer Azure VM wurden diese als vergleichsweise niedrig angenommen. Bei einer Migration von MySQL nach Azure SQL Database wurde, da hier eine Datenbankmigration erforderlich war, ein höherer Wert unterstellt. Die Betriebskosten wurden in der Cloud im Vergleich zur bestehenden Lösung als niedriger angesehen. Mit den Kosten konnten nun auch die Return-on-Investments für die einzelnen Entscheidungen berechnet werden.

Architektur-entscheidung	Gewichtete Nutzengrade	Gesamtnutzen der AE	Kosten der AE	ROI der AE
$ae_{1,1}$	100 + 200 + 800	1100	100	11,00
$ae_{1,2}$	425 + 800 + 850	2075	100	20,75
$ae_{1,3}$	500 + 900 + 850	2250	75	30,00
$ae_{2,1}$	100 + 200 + 800	1100	100	11,00

Architektur-entscheidung	Gewichtete Nutzengrade	Gesamtnutzen der AE	Kosten der AE	ROI der AE
$ae_{2,2}$	425 + 800 + 850	2075	100	20,75
$ae_{2,3}$	500 + 900 + 850	2250	110	20,45

Tabelle 7-49: Berechnung des ROI für die Architekturentscheidungen (Fallbeispiel 4)

Aus diesen Werten wiederum konnten die Return-on-Investments für die vier betrachteten Architekturalternativen abgeleitet werden.

Architektur-alternative	Nutzengrade der betreffenden Architekturentscheidungen	Gesamtnutzen der AAlt	Kosten der AAlt	ROI der AAlt
$Aalt_1$	1100 + 1100	2200	200	11,00
$Aalt_2$	2075 + 2075	4150	200	20,75
$Aalt_3$	2075 + 2250	4325	210	20,60
$Aalt_4$	2250 + 2250	4500	185	24,32

Tabelle 7-50: Berechnung des ROI für die Architekturalternativen (Fallbeispiel 4)

Wie erwartet, schnitt bei der ROI-Betrachtung die bestehende Lösung am schlechtesten ab. Schließlich wurde die Architekturdiskussion vom Wunsch nach Beseitigung identifizierter Schwächen der bestehenden Lösung (schlechte Skalierbarkeit, hohe Kosten etc.) motiviert.

7.5.2.7 Schritt 7: Präsentation der Entwurfs- und Bewertungsergebnisse

Im abschließenden Schritt konnten einige Erkenntnisse der Entwurfs- und Bewertungsschritte zusammengefasst werden:

- Beide Variabilitätspunkte, d.h. beide Punkte im Architekturentwurf in dem noch Alternativen für die Umsetzung enthalten waren, wurden als Risiko eingestuft. Das bedeutet, dass bei der Wahl einer Alternative ggf. Anforderungen verletzt werden. Im konkreten Fall müssten bei einer Beibehaltung der bestehenden Lösung Abstriche an die Skalierbarkeit von Service-Logik und Datenbank in Kauf genommen werden.

- Es konnte eine Empfehlung für eine weitergehende Migration $Aalt_4$ ausgesprochen werden, d.h. die Service-Logik sollte in eine Azure Web App und die Datenbank von MySQL nach Azure SQL Database migriert werden. Zwar fielen dabei die höchsten Migrationskosten an, dies wurde jedoch durch die niedrigsten Betriebskosten kompensiert. Zudem ergaben sich bei dieser Lösung die höchsten Nutzenwerte bei Skalierbarkeit und Performanz.

- Zweitbeste Lösung war eine 1:1-Migration von der gehosteten virtuellen Maschine in eine Azure Virtual Machine $Aalt_2$. Zwar war der Nutzenwert nicht so hoch wie bei $Aalt_3$, allerdings waren auch die Umsetzungskosten geringer, weshalb sich ein ROI-Wert ergab, der bei $Aalt_2$ besser war.

7.5.3 Ergebnis der Durchführung

Ergebnis war eine Bewertung von Entwurfsalternativen und ein Empfehlung einer Alternative. Tabelle 7-51 zeigt, inwieweit dabei die Anforderungen an den Leitfaden erfüllt wurden.

#	Anforderung	Bewertung	Anforderung erfüllt
1	Formale Beschreibung der Anforderungen	Die Anforderungen wurden formal beschrieben.	Ja.
2	Formale Beschreibung der Anforderungen einer Komponente	Die Anforderungen einzelner Komponenten wurden formal beschrieben.	Ja.
3	Formale Beschreibung von Entwurfsalternativen	Entwurfsalternativen wurden formal beschrieben.	Ja.
4	Unterstützung hybrider Cloud-Architekturen	Nicht untersucht.	n/a
5	Bewertbarkeit von Entwurfsalternativen	Die Alternativen wurden bewertet.	Ja.
6	Vermeidung unnötigen Overheads	Nicht untersucht.	n/a
7	Ein- und Ausgaben des Gesamtprozesses	Ein- und Ausgaben wurden beschrieben.	Ja.
8	Modellierbarkeit von Architekturmustern	Nicht untersucht.	n/a
9	Einmalige Ausführung von Entwurfs- und Bewertungsschritten	Priorisierungen wurden nur einmal vorgenommen.	Ja.
10	Zusammenfassung von Arbeitsschritten einzelner Stakeholder	Stakeholder mussten nur zu Beginn und am Ende hinzugezogen werden.	Ja.
11	Regelmäßige Plausibilitätsprüfungen	In den einzelnen Bewertungsschritten wurden Ergebnisse bewertet.	Ja.
12	Minimierung der Anzahl an Arbeitsschritten	Keine Wiederholung von Prozessschritten.	Ja.

Tabelle 7-51: Beurteilung der Anwendung des Leitfadens in Fallbeispiel 4

8 Zusammenfassung und Ausblick

Dieses Kapitel fasst die in der vorliegenden Arbeit gewonnenen Forschungsergebnisse zusammen, wirft einen kritischen Blick auf die Grenzen dieser Arbeit und zeigt Anknüpfungspunkte auf, in denen sie in Folgearbeiten weitergeführt werden könnte.

8.1 Zusammenfassung

Beim Entwurf der Architektur eines Softwaresystems gibt es eine Reihe von Herausforderungen. Neben der grundsätzlichen Problemstellung, bei begrenzter Ressourcenausstattung den mitunter komplexen Bauplan eines Softwaresystems zu erstellen, erschweren unterschiedliche Interessenslagen und Anforderungen der am Entwurf beteiligten Personen das Vorhaben. Deren teils irrationales, oft von persönlichen Interessen beeinflusstes Verhalten erschwert die Formulierung von Anforderungen sowie deren Priorisierung und damit angemessene Berücksichtigung bei dem Ziel, einen – unter den gegebenen Bedingungen – bestmöglichen Architekturentwurf zu erstellen. Es besteht die Gefahr, dass Entwurfsentscheidungen ohne ausreichende Begründung und Dokumentation getroffen werden, dies letztlich zu einer sub-optimalen Architektur führt und hierdurch ein Schaden für die Beteiligten entsteht.

Im Bereich Cloud Computing lassen sich diese Phänomene oft beobachten: die Cloud als Betriebsalternative wird oftmals ohne fundierte Begründung per se ausgeschlossen, als unsicher oder zu riskant eingestuft. Eine entsprechende Diskussion über die Konsequenzen (z.B. Verzicht auf Cloud Computing als preisgünstigere, mitunter sogar sicherere Alternative) unterbleibt. Unternehmen entgehen somit oft Vorteile, ohne dass hierfür entsprechenden Rechtfertigungen dokumentiert werden, um sie zu einem späteren Zeitpunkt nachvollziehen zu können.

Motivation für diese Arbeit war deshalb der Wunsch, Software-Architekten Werkzeuge an die Hand zu geben, um Diskussionen zu Anforderungen und deren Prioritäten anzuregen, Motivationen für Anforderungen zu hinterfragen und zu begründen und Entscheidungen zum Entwurf und zur Verteilung von Softwarekomponenten zu dokumentieren und damit nachvollziehbar zu machen. Dabei konnte es nicht das Ziel sein, Irrationalitäten zu beseitigen. Es war vielmehr das Ziel, durch Offenlegung von Zusammenhängen zwischen priorisierten Anforderungen und Konsequenzen von Architekturentscheidungen diese einer Diskussion zugänglich zu machen.

Zu diesem Zweck wurde eine Methode zum Entwurf, der Beschreibung und Bewertung einer Cloud-basierten Softwarearchitektur entwickelt. Dabei werden mehrere Modelle zur Beschreibung von Entwurfs- und Bewertungsergebnissen erstellt und ineinander überführt. Mit einer durchgängigen, formalen Beschreibung von Anforderungen, Architekturelementen, offenen Entwurfsentscheidungen, Risiken, Kosten und Nutzen, können (über die einzelnen Entwurfs- und Bewertungsschritte hinweg) Zusammenhänge zwischen Anforderungen und Entscheidungen sowie deren Auswirkungen auf den Return-on-Investment des Gesamtsystems transparent gemacht werden. Abb. 8-1 skizziert das Vorgehen im Rahmen dieser Arbeit.

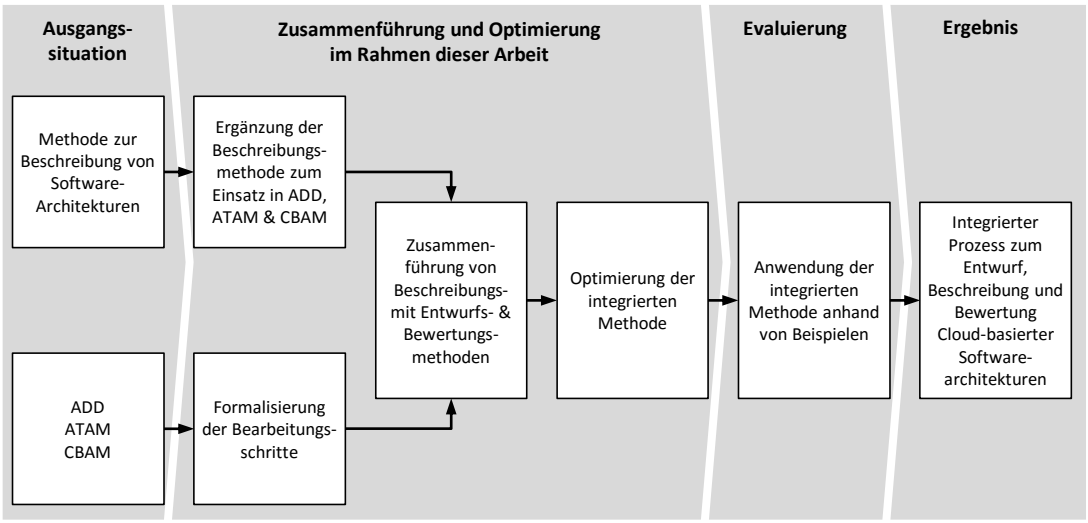

Abb. 8-1 Vorgehen im Rahmen dieser Arbeit

Ausgangspunkt für die Entwicklung der integrierten Vorgehensmethode waren zwei Quellen: zum einen wurde für die Beschreibung der Entwurfs- und Bewertungsartefakte ein Beschreibungsmodell von Ralph Mietzner [Mie10] als Basis verwendet. Über dieses können Softwarearchitekturen mit Variabilitäten (d.h. Entwurfsalternativen) beschrieben werden. Die Beschreibungsmethode sieht jedoch nicht vor, Anforderungen mit deren Prioritäten zu beschreiben, mit Architekturelementen in Beziehung zu setzen und Alternativen für Variabilitäten zu bewerten. Auch macht sie keine Vorgaben, wie letztlich aus Anforderungen ein Architekturentwurf abgeleitet werden kann. Somit ist es zwar möglich, Architekturen zu beschreiben, deren Implementierungen ganz oder in Teilen alternativ lokal oder in der Cloud ausgeführt werden können. Es kann jedoch nicht beurteilt werden, auf Basis welcher Anforderungen eine Entscheidung pro oder contra Cloud getroffen werden sollte und welche Auswirkungen dies auf den zu erwartenden Return-on-Investment haben kann. Um diese Lücken zu schließen, wurden drei bestehende Entwurfs- und Bewertungsmethoden ergänzend hinzugenommen:

die ADD (Attribute Driven Design Method) zur iterativen Ableitung von Architekturkomponenten aus gegebenen, priorisierten Anforderungen, die ATAM (Architecture Tradeoff Analysis Method) zur Bewertung einer gegebenen Architektur hinsichtlich bestehender Risiken sowie die CBAM (Cost Benefit Analysis Method) zur Abschätzung des Mehrwertes von Architekturalternativen auf Basis zu erwartender Kosten- und Nutzengrößen. Diese drei Methoden sehen keine bestimmte Beschreibungstechnik für Anforderungen, Szenarien, Prioritäten etc. vor. Deshalb ist es nur schwer möglich, konkrete Zusammenhänge zwischen Anforderungen und Architekturalternativen zu formulieren, um eine Empfehlung für eine der Alternativen auszusprechen und dies nachvollziehbar zu begründen.

Im Rahmen dieser Arbeit wurden die Defizite der Beschreibungsmethode auf der einen, sowie der Entwurfs- und Bewertungsmethoden auf der anderen Seite durch entsprechende Ergänzungen beseitigt und durch Zusammenführung zu einem integrierten Gesamtprozess entwickelt, dessen Anforderungs-, Entwurfs- und Bewertungsartefakte formal beschrieben werden können. In Kapitel 5 wurden die verwendeten Methoden jeweils einzeln formalisiert. Hierzu wurden die im Verlauf der betreffenden Methode verwendeten Ein- und Ausgabeartefakte formal definiert, d.h. es wurde die ursprüngliche Beschreibungsmethode jeweils um diese Artefakte erweitert. Neben den in der Methode bereits vorhandenen Modellen für den Architekturentwurf (bestehend zum einen aus dem **Anwendungsmodell** zur Beschreibung der Architekturkomponenten und deren Implementierungen und zum anderen aus dem **Variabilitätsmodell** zur Beschreibung von Alternativen im Architekturentwurf) wurden folgende neue Modelle definiert:

- **Businessmodell** zur Beschreibung der grundsätzlichen, fachlichen Motivationen für die Erstellung des Softwaresystems
- **Anforderungsmodell** zur Darstellung priorisierter fachlicher und qualitativer Anforderungen sowie Vorgaben für den Entwurf
- **Szenarienmodell** zur Konkretisierung von Qualitätsanforderungen in Form vergleichbarer, abgeleiteter Szenarien
- **Risikomodell** zur Auflistung bestehender Risiken im vorliegenden Architekturentwurf
- **Nutzenmodell** zur Beschreibung von Kosten- und Nutzengrößen und Ableitung des Return-on-Investment von Architekturalternativen

Anschließend wurden für die einzelnen Bearbeitungsschritte der Methode zunächst soweit wie möglich Algorithmen formuliert, die die Eingabeartefakte in die zugehörigen Ausgabeartefakte überführen. Für die erweiterten Ausgangsmethoden ergab sich dann:

- Die erweiterte ADD führt ein Anforderungsmodell in einen Architekturentwurf über.
- Die erweiterte ATAM führt ein Businessmodell und einen Architekturentwurf in ein Szenarienmodell und ein Risikomodell über.

- Die erweiterte CBAM führt ein Szenarienmodell, ein Businessmodell und einen Architekturentwurf in ein Nutzenmodell über.

Bei der Erweiterung der Beschreibungsmethode wurde sichergestellt, dass die einzelnen Modelle übergreifend über die drei Entwurfs- und Bewertungsmethoden hinweg verwendet werden konnten. Somit war es möglich, in Kapitel 6 einen ersten Entwurf für den integrierten Gesamtprozess durch sequenzielle Durchführung von ADD, ATAM und CBAM zu gewinnen. In diesem Entwurf waren jedoch einige Arbeitsschritte redundant enthalten (z.B. Priorisierung von Anforderungen), auch mussten verschiedene Beteiligte mehrfach für einzelne Arbeitsschritte hinzugezogen werden. Der Entwurf wurde deshalb einem Optimierungsschritt unterzogen. In diesem wurden einige Arbeitsschritte der verwendeten Ausgangsmethoden zusammengefasst und so umsortiert, dass Redundanzen in der Durchführung beseitigt und Einsätze Beteiligter minimiert wurden.

Ergebnis war der in Abb. 6-2 skizzierte Gesamtprozess, der im nächsten Schritt in vier Projektszenarien evaluiert wurde.

- *Fallbeispiel 1: Berechnungsalgorithmus* – für diesen bewusst einfach gehaltenen Algorithmus (Monte-Carlo-Berechnung von Pi) wurden die grundlegenden Anforderungen an die Methode (Einfachheit, Vermeidung von Overhead) validiert.

- *Fallbeispiel 2: 3-Schicht-Web-Anwendung* – hier wurde ein typisches Cloud-basiertes Anwendungsszenario, eine 3-schichtige Web-Anwendung, durchgespielt und dabei alternative Architekturmuster gegenübergestellt.

- *Fallbeispiel 3: Cloud-basierte CAD-Anwendung* – in diesem hybriden System (Teile in lokaler Ausführung, Teile in der Cloud) wurden für einzelne Anwendungskomponenten einer Client-Anwendung alternative Ausführungsmodelle bewertet.

- *Fallbeispiel 4: Cloud-Migration eines bestehenden Webservice* – für einen klassischen Webservice wurden Alternativen zur Migration in die Cloud und dort in mehrere Rechenzentren durchgespielt.

In diesen Fallbeispielen konnte nachgewiesen werden, dass die Anforderungen, die an den Gesamtprozess gestellt wurden, abgedeckt werden. Der Forschungsbeitrag dieser Arbeit lässt sich somit wie folgt zusammenfassen:

- Über eine Beschreibungs- und Vorgehensmethode kann aus gegebenen Business Drivern sowie fachlichen und qualitativen Anforderungen und Entwurfseinschränkungen der Architekturentwurf einer Cloud-basierten verteilten Unternehmensanwendung abgeleitet, dokumentiert und bewertet werden.

- Die Anforderungen können formal beschrieben werden. Dabei können Bezüge zur Gesamtarchitektur sowie einzelnen Architekturkomponenten dargestellt werden. Anforderungen können über die Modelle zu ihrem Ursprung, d.h. zu Business Drivern und damit der Personengruppe zurückverfolgt werden, die sie aufgestellt haben.

- Der Architekturentwurf kann noch Entwurfsalternativen, d.h. offene Architekturentscheidungen (insbesondere zur Verteilung von Softwarekomponenten) enthalten. Dabei werden sowohl rein lokal oder rein Cloud-basierte Systeme als auch hybride Alternativen unterstützt.

- Die Entwurfsalternativen können bezüglich ihres Einflusses auf qualitative Anforderungen sowie bezüglich ihres Kosten- und Nutzenbeitrags miteinander verglichen werden, so dass eine dokumentierte, nachvollziehbare Entscheidung für eine der Alternativen möglich wird.

- Die Entwurfs- und Bewertungsschritte sind dahingehend zusammengefasst und optimiert, dass einzelne Schritte nur ein Mal durchgeführt und die zugehörigen Beteiligten so wenig wie nötig hinzugezogen werden müssen. Insbesondere für kleinere Projekte wird so der Prozess-Overhead minimiert.

- Über regelmäßige im Prozess vorgesehene Plausibilitätsprüfungen können die jeweils erzielten Ergebnisse mit auf Erfahrungen basierende Erwartungen abgeglichen und Korrekturen in formalen Anforderungen und Priorisierungen vorgenommen werden.

Die Mehrwerte der im Ergebnis erhaltenen integrierte Beschreibungs- und Vorgehensmethode sind unter anderem:

- Entscheidungen können nachvollziehbar dokumentiert werden, was diese für nachfolgende Beteiligte verständlich macht.

- Beziehungen zwischen Anforderungen und Entscheidungen werden sichtbar und so einer Diskussion zugänglich gemacht.

- Unter gegebenen Voraussetzungen kann der Architekturentwurf mit dem höchsten Return-on-Investment abgeleitet werden.

Die Methode lässt sich somit in folgendem Kurzprofil zusammenfassen:

Ziele der Methode:	Entwurf einer Softwarearchitektur und Bewertung darin enthaltener Architekturalternativen hinsichtlich des jeweils zu erwartenden Return-on-Investment
Eingabe:	Business Driver der beteiligten Stakeholder
Zwischenergebnisse:	Szenarienmodell, Anforderungsmodell
Ausgabe:	Architekturentwurf, Risikomodell, Nutzenmodell
Qualitätsanforderungen:	Verschiedene (flexibel)
Evaluierungstechnik:	Utility-Baum, Szenarien
Anforderungen an das Evaluierungsteam:	Hoch, Expertise im Erzeugen eines Utility-Baums sowie der Bewertung von Architekturalternativen und Abschätzung von zu erwartenden Nutzen und Aufwänden erforderlich
Projektphase:	Früh, d.h. noch vor Erstellung eines Architekturentwurfs
Beteiligung von Stakeholdern:	Ja
Soziale Interaktion:	Meeting unter Beteiligung aller Stakeholder

Tabelle 8-1: Kurzprofil der gewonnenen Beschreibungs- und Vorgehensmethode

8.2 Kritische Betrachtung

Neben der Nennung der Mehrwerte soll eine kritische Betrachtung von Durchführung und Ergebnissen dieser Arbeit nicht unterbleiben.

Zwar konnte es von Beginn an nicht das Ziel sein, irrationales Verhalten der an einem Projekt beteiligten Personen zu verhindern. Dazu ist dieses, wie Studien von Dan Ariely [Ari08] zeigen, viel zu sehr im Unterbewusstsein der Menschen verankert. Demnach wäre der Anspruch verfehlt, durch die in dieser Arbeit erstellte Beschreibungs- und Bewertungsmethode den Entwurf einer Softwarearchitektur gänzlich rational zu gestalten. Wie die Evaluierungsszenarien jedoch gezeigt haben, hilft die Methode dabei, Zusammenhänge zwischen priorisierten Anforderungen auf der einen Seite und den Konsequenzen im Architekturentwurf auf der anderen Seite aufzuzeigen. Plausibilitätsprüfungen und Rückkopplungsschleifen im Prozess erlauben es den Beteiligten dann, auf diese Zusammenhänge zu reagieren, indem sie z.B. Anforderungen neu priorisieren, um ein gewünschtes Ergebnis im Architekturentwurf zu erhalten. Es könnte argumentiert werden, dass diese Möglichkeit den gesamten Prozess ad absurdum führt, da es den Beteiligten möglich ist, die Eingabeelemente (Business Driver, Anforderungen etc.) so lange zu verändern, bis eine bereits zu Beginn gewünschte Zielarchitektur als die „bestmögliche“ bestätigt wird. Warum dann also ein mitunter aufwändiger Entwurfs- und Bewertungsprozess? Die Antwort: weil die im Rahmen des Prozesses erstellte Dokumentation diese Zusammenhänge festhält und somit später hinterfragbar machen. Gibt es also im Projekt

Beteiligte, die Cloud Computing kategorisch ausschließen und einen Architekturentwurf anstreben, der diese Form des Anwendungsbetriebs vermeidet, müssen diese Beteiligten diese Anforderung dokumentieren und so hoch priorisieren, dass sie alle anderen Anforderungen wie z.B. Skalierbarkeit, Elastizität und ähnliche (die in der Cloud tendenziell besser umgesetzt werden können) überstrahlen. Auch wird dokumentiert, zu welchem Preis diese Alternative gewählt wird, nämlich durch Verzicht auf eine möglicherweise kostengünstigere Umsetzung mit Cloud-Technologien.

Die im Rahmen dieser Arbeit verwendeten Evaluierungsszenarien zur Bewertung der Gesamtmethode wurden mit dem Ziel ausgewählt, die oben genannten Anforderungen an die Methode zu überprüfen. Sie waren von begrenzter Komplexität, um die Nachweise einfach und nachvollziehbar zu gestalten. Es kann die Frage gestellt werden, ob nicht auch ein komplexeres Beispiel zur Evaluierung herangezogen werden hätte sollen. Wenngleich die Frage berechtigt ist, kann aufgrund von Erfahrungen aus Diskussionen zu Cloud-basierten Softwarearchitekturen entgegnet werden, dass sich die Diskussionen um das Für und Wider von Cloud Technologien auf einer grundsätzlicheren Ebene und nicht in kleinen Details der Architektur abspielen. Häufig herrscht unter den Beteiligten durchaus Einvernehmen zur grundsätzlichen Architektur der Software, jedoch unterschiedliche Vorstellungen zur Verteilung einzelner Softwarekomponenten. Nachdem angenommen werden kann, dass die Methode auch bei komplexeren Vorhaben anwendbar ist (da sich die verwendeten Basis-Methoden ADD, ATAM und CBAM auch in größeren Softwareprojekten bewährt haben), wurde in dieser Arbeit auf die Durchführung eines komplizierteren Entwurfsvorhabens zugunsten der Übersichtlichkeit verzichtet.

Leider war es auch nicht möglich, ein Projekt rückwirkend zu untersuchen, in welchem nachweislich aufgrund des Fehlens eines integrierten Entwurfs- und Bewertungsprozesses eine sub-optimale Architektur gewählt wurde, damit ist es auch nicht möglich, zu beziffern, inwieweit bestehende Opportunitätskosten durch Wahl einer sub-optimalen Lösung durch die integrierte Methode beseitigt oder verkleinert würden. Der Mehrwert der Methode beschränkt sich damit auf die Vorteile einer Entwurfs- und Bewertungsdokumentation (Nachvollziehbarkeit von Entscheidungen, Diskussion von Zusammenhängen etc.).

8.3 Ausblick

Diese Arbeit bietet in einigen Bereichen Anknüpfungspunkte für weitere Forschungsarbeiten, die die hier erzielten Ergebnisse weiterführen können.

Der Erfolg der in dieser Arbeit hergeleiteten Gesamtmethodik hängt an mehreren Stellen vom Erfahrungsschatz des beteiligten Software-Architekten ab. Er steht beispielsweise maßgeblich

in der Verantwortung, Zusammenhänge zwischen Architekturentscheidungen und qualitativen Ausprägungen sowie Aufwände für die Umsetzung von Alternative abzuschätzen. Derartige Abschätzungen sind, je früher im Architekturentwurf sie erfolgen, jedoch zumeist mit Unsicherheiten belegt. Die angesetzten Werte für Sensitivitäten, Nutzen, Kosten etc. sind in der Regel mit einem Risiko behaftet (z.B. Eintrittswahrscheinlichkeit für Nutzen, Erreichbarkeit von Kostenzielen). Es fehlen derzeit in der Beschreibungs- und Bewertungsmethodik die Möglichkeiten, Risiken oder Wertebereiche für bestimmte Attribute in den Modellen und Berechnungsvorschriften zu berücksichtigen. Die Modelle könnten also in diese Richtung hin erweitert werden. Damit würde es ggf. möglich, für die berechneten Return-on-Investments Wahrscheinlichkeiten anzugeben. Somit könnte für Architekturalternativen angegeben werden, mit welcher Wahrscheinlichkeit der Return-on-Investment in einem bestimmten Wertebereich liegt. Dies würde den Prozessbeteiligten die Möglichkeit geben, das Risiko bei der Entscheidung für eine Alternative zu berücksichtigen. Sie könnten sich also bewusst gegen eine Alternative mit einem hohen Return-on-Investment entscheiden, wenn diese mit einem hohen Risiko behaftet ist, und sich dafür für eine Alternative mit einem schlechteren Wert entscheiden, bei der das Risiko aber niedriger bewertet wird. Die könnte zum einen dazu führen, dass gerade bei risikoscheuen Entscheidungsträgern Alternativen wegen des tendenziell geringeren Risikos in bekannten Ausführungsumgebungen den Vorzug erhalten (Cloud Computing also einen eher schweren Stand hätte). Zum anderen würde dies aber auch aufzeigen, an welchen Stellen Machbarkeitsstudien und prototypische Umsetzungen das Risiko senken und damit den Weg für weitere Alternativen ebnen könnten.

Ein weiteres Betätigungsfeld für weiterführende Arbeiten ist die Ergänzung der Methodik um Werkzeugunterstützung. Mit den Möglichkeiten zur formalen Beschreibung der verschiedenen Ein- und Ausgabeartefakte der einzelnen Bearbeitungsschritte sind die Voraussetzungen geschaffen, ebendiese Artefakte durch ein Entwurfs- und Entwicklungswerkzeug zu erzeugen, zu bearbeiten und zu validieren. Damit würde es möglich, Regeln für die einzelnen Modelle werkzeuggesteuert zu überwachen, Konsistenz automatisch zu prüfen (z.B. zu prüfen, ob es zu jeder Anforderung einen Business Driver gibt), erstellte Dokumentation zu verwalten und den Entwurfs- und Bewertungsprozess mit den folgenden Entwicklungsschritten zu koppeln, für die es bereits Werkzeugunterstützung gibt. Damit könnte die in dieser Arbeit erstellte Methode in den gesamten Lebenszyklus einer Softwareanwendung eingebettet werden.

Literaturverzeichnis

[AA06] Arsanjani, Ali; Allam, Abdul (2006): *Service-Oriented Modeling and Architecture for Realization of an SOA*, in: IEEE Computer Society (Hrsg.): IEEE International Conference on Services Computing, 2006. SCC '06., S.521

[ABB+09] Arsanjani, Ali; Booch, Grady; Boubez, Toufic u.a. (2009): *SOA Manifesto*, in: SOA Manifesto Working Group (Hrsg.): 2nd International SOA Symposium, Rotterdam, abrufbar unter: http://www.soa-manifesto.org/ (17.01.2015)

[ABK10] Abrahamsson, Pekka; Babar, Muhammad Ali; Kruchten, Philippe (2010): *Agility and Architecture: Can They Coexist?*, in: IEEE Computer Society (Hrsg.): IEEE Software 27(2), S.16-22

[AFG09] Armbrust, Michael; Fox, Armando; Griffith, Rean et al. (2009): *Above the Clouds: A Berkeley View of Cloud Computing*, in: UC Berkeley RAD Laboratory (Hrsg.): Technical Report UCB/EECS-2009-28, Berkeley, abrufbar unter: http://www.eecs.berkeley.edu/Pubs/TechRpts/2009/EECS-2009-28.pdf (17.01.2015)

[App15] AppNexus (2015): *AppNexus Cloud*, abrufbar unter: http://www.appnexus.com/ (17.01.2015)

[Ari08] Ariely, Dan (2008): *Denken hilft zwar, nützt aber nichts: Warum wir immer wieder unvernünftige Entscheidungen treffen*, 1. Auflage, Droemer/Knaur, München

[Aws] Amazon Web Services, Inc. (2015): *Amazon Web Services*, Seattle, abrufbar unter: http://aws.amazon.com/de/ (17.01.2015)

[Aws14-1] Amazon Web Services (2014): *AWS Case Study: Hess Corporation*, Seattle, abrufbar unter: http://aws.amazon.com/de/solutions/case-studies/hess-corporation/ (17.01.2015)

[Aws14-2] Amazon Web Services (2014): *AWS Case Study: Airbnb*, Seattle, abrufbar unter: http://aws.amazon.com/de/solutions/case-studies/airbnb/ (17.01.2015)

[AZJ04] Ali Babar, Muhammad; Zhu, Liming; Jeffery, Ross (2004): *A Framework for Classifying and Comparing Software Architecture Evaluation Methods*, National ICT Australia Ltd., New South Wales, abrufbar unter: http://www.cse.unsw.edu.au/~limingz/publication/ASWEC2004_Zhu.pdf (17.01.2015)

[BAA10] Benestad, Hans Christian; Anda, Bente; Arisholm, Erik (2010): *Understanding cost drivers of software evolution: a quantitative and qualitative investigation of change effort in two evolving software systems*, in: Springer-Verlag (Hrsg.): Empirical Software Engineering, 2010(15), Issue 2, S.166-203

[BBK02] Backmann, Felix; Bass, Len; Klein, Mark (2002): *Illuminating the Fundamental Contributors to Software Architecture Quality*, Carnegie Mellon, Software Engineering Institute, Pittsburgh, PA, abrufbar unter: http://www.sei.cmu.edu/reports/02tr025.pdf (17.01.2015)

[BCK03] Bass, Len; Clements, Paul; Kazman, Rick (2003): *Software Architecture in Practice*, 2. Auflage, Addison-Wesley, Boston

[BEL+03] Barbacci, Mario; Ellison Wobert; Lattanze, Anthony u.a. (2003): *Quality Attribute Workshops, Third Edition*, 3. Auflage, Carnegie Mellon, Software Engineering Institute, Pittsburgh, PA, abrufbar unter: http://www.sei.cmu.edu/reports/03tr016.pdf (17.01.2015)

[BEL+08] Buckl, Sabine; Ernst, Alexander; Lankes, Josef u.a. (2008): *Enterprise Architecture Management Patterns -- Exemplifying the Approach*, in: IEEE Computer Society (Hrsg.): 12th International IEEE Enterprise Distributed Object Computing Conference, 2008. EDOC '08., S.393-402, München

[BG04] Babar, Muhammad Ali; Gorton, Ian (2004): *Comparison of Scenario-Based Software Architecture Evaluation Methods*, in: IEEE Computer Society (Hrsg.): 11th Asia-Pacific Software Engineering Conference, 2004, S.600-607, New South Wales

[BIT09] Münzl, Gerald; Przywara, Bernhard; Reti, Martin u.a. (2009): *BITKOM-Leitfaden: Cloud Computing - Evolution in der Technik, Revolution im Business*, Bundesverband Informationswirtschaft, Telekommunikation und neue Medien e.V., Berlin

[BKL+95] Barbacci, Mario; Klein, Mark; Longstaff, Thomas u.a. (1995): *Quality Attributes*, Carnegie Mellon, Software Engineering Institute, Pittsburgh, PA, abrufbar unter: http://www.sei.cmu.edu/reports/95tr021.pdf (17.01.2015)

[BKN+10] Baun, Christian; Kunze, Marcel; Nimis, Jens; u.a. (2010): *Cloud Computing - Webbasierte dynamische IT-Services*, Springer-Verlag, Berlin

[BN12] Bass, Len; Nord, Robert L. (2012): *Understanding the Context of Architecture Evaluation Methods*, in: IEEE Computer Society (Hrsg.): 2012 Joint Working Conference on Software Architecture & 6th European Conference on Software Architecture, S.277-281, Helsinki

[Boo06] Booch, Grady (2006): *On Architecture*, in: IEEE Computer Society (Hrsg.): IEEE Software 23(2), S.16-18

[Boo07] Booch, Grady (2007): *The Economics of Architecture-First*, in: IEEE Computer Society (Hrsg.): IEEE Software 24(5), S.18-20

[Car08] Carr, Nicholas (2008): *The Big Switch - Rewiring the World, From Edison to Google*, 1. Auflage, W.W. Norton & Company, New York

[CDG+10] Chou, David; deVadoss, John; Ganghi, Nitin u.a. (2010): *SOA with .NET & Windows Azure*, 1. Auflage, Prentice Hall, Upper Saddle River, NJ

[CKK02] Clements, Paul; Kazman, Rick; Klein, Mark (2011): *Evaluating Software Architectures*, 9. Auflage, Addison-Wesley, Boston

[CN12] Clements, Paul; Northrop, Linda (2012): *Software Product Lines*, 8. Auflage, Addison-Wesley, Boston

[Dar46] Darwin, Charles (1946): *Automatic Computing Engine (ACE)*, National Physical Laboratory, 17 April 1946; PRO document reference DSIR 10/275, London, digitale Kopie abrufbar über "The Turing Archive for the History of Computing" unter: http://www.alanturing.net/darwin_ace/ (31.01.2015)

[DEF+08] Jürgen Dunkel et. al. (2008): *Systemarchitekturen für Verteilte Anwendungen*, 1. Auflage, Carl Hanser Verlag, München

[DN02] Dobrica, Liliana; Niemelä (2002): *A Survey on Software Architecture Analysis Methods*, in: IEEE Computer Society (Hrsg.): IEEE Transactions on Software Engineering (Volume: 28, Issue: 7, S. 638-653), Bukarest

[DoD11] Department of Defense (2011): *The DoDAF Architecture Framework Version 2.02*, Department of Defense, abrufbar unter: http://dodcio.defense.gov/Portals/0/Documents/DODAF/DoDAF_v2-02_web.pdf (17.01.2015)

[EHM07] Eicker, Stefan; Hegmanns, Christian; Malich Stefan (2007): *Auswahl von Bewertungsmethoden für Softwarearchitekturen*, in: Institut für Informatik und Wirtschaftsinformatik, Universität Duisburg/Essen (Hrsg.): ICB-Research Report No.14, Essen, abrufbar unter: http://www.icb.uni-due.de/fileadmin/ICB/research/research_reports/ICBReport14.pdf (17.01.2015)

[EHM08] Eicker, Stefan; Hegmanns, Christian; Malich Stefan (2008): *Projektbezogene Auswahl von Bewertungsmethoden für Softwarearchitekturen*, Lehrstuhl für Wirtschaftsinformatik und Softwaretechnik, Universität Duisburg-Essen, Essen, abrufbar unter: http://ibis.in.tum.de/mkwi08/24_Software-Produktmanagement_fuer_flexible_Anwendungssysteme/04_Eicker.pdf (17.01.2015)

[Erl06] Erl, Thomas (2006): *Service-Oriented Architecture*, 6. Auflage, Prentice Hall, Upper Saddle River, NJ

[FBH+13] Farwick, Matthias; Breu, Ruth; Hauder, Matheus (2013): *Enterprise Architecture Documentation: Empirical Analysis of Information Sources for Automation*, in: IEEE Computer Society (Hrsg.): 46th Hawaii International Conference on System Sciences (HICSS), 2013, S.3868-3877, Wailea

[Fos02] Foster, Ian (2002): *What is the Grid? A Three Point Checklist*, Argonne National Laboratory, Argonne, abrufbar unter: http://www.mcs.anl.gov/~itf/Articles/WhatIsTheGrid.pdf (17.01.2015)

[Fow03] Fowler, Martin (2003): *Patterns of Enterprise Application Architecture*, 6. Auflage, Pearson Education, Boston

[FUS+09] Franke, Ulrik; Ullberg, Johan; Sommerstad, Teodor u.a. (2009): *Decision Support oriented Enterprise Architecture Metamodel Management using Classification Trees*, KTH Royal Institute of Technology, Stockholm

[Gar09] Pettey, Christy (2009): *Gartner Highlights Five Attributes of Cloud Computing*, Gartner Inc., Stamford, Connecticut, abrufbar unter: http://www.gartner.com/it/page.jsp?id=1035013 (17.01.2015)

[Gee09] Geelan, Jeremy (2009): *Twenty-One Experts Define Cloud Computing*, Sys-Con Media Inc., abrufbar unter: http://virtualization.sys-con.com/node/612375 (17.01.2015)

[Goo14] Google, Inc. (2014): *Google App Engine*, Mountain View, abrufbar unter: https://cloud.google.com/appengine/docs (17.01.2015)

[Goo15] Google, Inc. (2015): *Google Apps for Business*, Mountain View, abrufbar unter: https://www.google.de/intx/de/work/apps/business/ (17.01.2015)

[Gra03] Gray, Jim (2003): *Distributed Computing Economics*, Microsoft Research, Redmond, abrufbar unter: http://research.microsoft.com/pubs/70001/tr-2003-24.pdf (17.01.2015)

[GV02] Giesen, Joachim; Völker, Axel (2002): *Requirements Interdependencies and Stakeholders Preferences*, in: IEEE Computer Society (Hrsg.): Proceedings of the IEEE Joint International Conference on Requirements Engineering, 2002, S. 206-209

[HAH12] Heesch, U. van; Avgeriou, P.; Hilliard, R. (2012): *A Documentation Framework for Architecture Decisions*, in: Elsevier B.V. (Hrsg.): The Journal of Systems and Software, Vol. 85, No. 4, pp. 795-820, Amsterdam

[HAH12-1] Heesch, Uwe van; Avgeriou, Paris; Hilliard, Rich (2012): *Forces on Architecture Decisions - A Viewpoint*, in: IEEE Computer Society (Hrsg.): Joint Working IEEE/IFIP Conference on Software Architecture (WICSA) and European Conference on Software Architecture (ECSA), 2012, S.101-110, Helsinki

[HEA14] van Heesch, Uwe; Eloranta, Veli-Pekka; Avgeriou, Paris (2013): *Decision-Centric Architecture Reviews*, in: IEEE Computer Society (Hrsg.): IEEE Software (Volume: 31, Issue: 1, S. 69-76)

[HP13] Hesse, Tom-Michael; Paech, Barbara (2013): *Supporting the Collaborative Development of Requirements and Architecture Documentation*, in: IEEE Computer Society (Hrsg.): 3rd International Workshop on the Twin Peaks of Requirements and Architecture (TwinPeaks), 2013, S.22-26, Rio de Janeiro

[HP15] Hewlett-Packard (2015): *HP Helion CloudSystem*, abrufbar unter: http://www8.hp.com/us/en/cloud/cloudsystem.html (17.01.2015)

[IAH+04] Ionita, Mugurel; America, Pierre; Hammer, Dieter u.a. (2004): *A scenario-driven approach for value, risk, and cost analysis in system architecting for innovation*, in: IEEE Computer Society (Hrsg.): Fourth Working IEEE/IFIP Conference on Software Architecture, 2004. WICSA 2004. Proceedings., S.277-280

[IAH05] Ionita, Mugurel; America, Pierre; Hammer, Dieter (2005): *A Method for Strategic Scenario-Based Architecting*, in: IEEE Computer Society (Hrsg.): Proceedings of the 38th Annual Hawaii International Conference on System Sciences, 2005. HICSS '05., S.312b, Hawaii

[Iee07] IEEE (2007): *IEEE Std. 1471-2000: IEEE Recommended Practice for Architectural Description of Software-Intensive Systems*, The Institute of Electrical and Electronics Engineers, Inc., New York

[IHO02] Ionita, Mugurel; Hammer, Dieter; Obbink, Henk (2002): *Scenario-Based Software Architecture Evaluation Methods: An Overview*, in: Workshop on Methods and Techniques for Software Architecture Review and Assessment at the International Conference on Software Engineering, 2002, Orlando

[IM11] Iacobucci, Joseph; Mavris, Dimitri (2011): *A Method for the Generation and Evaluation of Architecture Alternatives on the Cloud*, in: Georgia Institute of Technology (Hrsg.): 6th International Conferecne on System of Systems Engineering, Atlanta

[Inv15] Investorpedia (2015): *Return On Investment - ROI*, abrufbar unter: http://www.investopedia.com/terms/r/returnoninvestment.asp (17.01.2015)

[Jai11] Jain, Prem (2011): *Architecture Evolution and Evaluation (ArchEE) Capability*, in: The MITRE Corporation (Hrsg.): 6th International Conferecne on System of Systems Engineering, McLean

[JJS+07] Johnson, Pontus; Johansson, Erik; Sommestad, Teodor u.a. (2007): *A Tool for Enterprise Architecture Analysis*, in: IEEE Computer Society (Hrsg.): 11th IEEE International Enterprise Distributed Object Computing Conference, 2007, S.142-153, Annapolis, MD

[JLQ+11] Jonkers, Henk; Lankhorst, Marc; Quartel, Dick u.a. (2011): *ArchiMate for Integrated Modelling Throughout the Architecture Development and Implementation Cycle*, in: IEEE Computer Society (Hrsg.): 2011 IEEE 13th Conference on Commerce and Enterprise Computing (CEC), S. 294-301, Luxembourg

[KAK02] Kazman, Rick; Asundi, Jai; Klein, Mark (2002): *Making Architecture Design Decisions: An Economic Approach*, Carnegie Mellon, Software Engineering Institute, Pittsburgh, PA, abrufbar unter: http://www.sei.cmu.edu/reports/02tr035.pdf (17.01.2015)

[KKB+98] Kazman, Rick; Klein, Mark; Barbacci, Mario (1998): *The Architecture Tradeoff Analysis Method*, Carnegie Mellon, Software Engineering Institute, Pittsburgh, PA, abrufbar unter: http://www.sei.cmu.edu/library/assets/icse98-2.pdf (17.01.2015)

[KKC00] Kazman, Rick; Klein, Mark; Clements, Paul (2000): *ATAM: Method for Architecture Evaluation*, Carnegie Mellon, Software Engineering Institute, Pittsburgh, PA, abrufbar unter: http://www.sei.cmu.edu/reports/00tr004.pdf (17.01.2015)

[KKL10] Kossmann, Donald; Kraska, Tim; Loesing, Simon (2010): *An Evaluation of Alternative Architectures for Transaction Processing in the Cloud*, ETH Zürich, Zürich, abrufbar unter: http://systems.ethz.pubzone.org/servlet/Attachment?attachmentId=76&versionId=1363456 (17.01.2015)

[KKN11] Keuler, Thorsten; Knodel, Jens; Naab, Matthias (2011): *Architecture-Centric Software and Systems Engineering*, Fraunhofer IESE, , abrufbar unter: http://publica.fraunhofer.de/eprints/urn:nbn:de:0011-n-1863619.pdf (17.01.2015)

[KN14] Knodel, Jens; Naab, Matthias (2014): *Software Architecture Evaluation in Practice*, in: IEEE Computer Society (Hrsg.): IEEE/IFIP Conference on Software Architecture (WICSA), 2014, S.115-124, Sydney, NSW

[KN14-1] Knodel, Jens; Naab, Matthias (2014): *Mitigating the Risk of Software Change in Practice*, in: IEEE Computer Society (Hrsg.): IEEE Conference on Software Maintenance, Reengineering and Reverse Engineering (CSMR-WCRE), 2014 Software Evolution Week, Antwerpen

[KNT08] Klems, Markus; Nimis, Jens; Tai, Stepan (2008): *Do Clouds Compute? A Framework for Estimating the Value of Cloud Computing*, in: Springer-Verlag (Hrsg.): Designing E-Business Systems. Markets, Services, and Networks, 7th Workshop on E-Business, WEB 2008, S.110-123, Paris

[KNK03] Kazman, Rick; Nord, Robert L.; Klein, Mark (2003): *A Life-Cycle View of Architecture Analysis and Design Methods*, Carnegie Mellon, Software Engineering Institute, Pittsburgh, PA, abrufbar unter: http://www.sei.cmu.edu/reports/03tn026.pdf (17.01.2015)

[Kru95] Kruchten, Philippe (1995): *Architectural Blueprints - The ""4+1"" View Model of Software Architecture*, in: IEEE Computer Society (Hrsg.): IEEE Software 1995, 12(6), S.42-50

[Lin11] Lin, Tom C.W. (2011): *A Behavioural Framework for Securities Risk*, The Seattle University Law Review, Seattle

[LKN+09] Lenk, Alexander; Klems, Markus; Nimis, Jens u.a. (2009): *What's Inside the Cloud? An Architectural Map of the Cloud Landscape*, in: IEEE Computer Society (Hrsg.): ICSE Workshop on Software Engineering Challenges of Cloud Computing, 2009. CLOUD '09, S.23-31, Vancouver, BC

[LRV99] Lassing, Nico; Rijsenbrij, Daan; van Vliet, Hans (1999): *The Goal of Software Architecture Analysis: Confidence Building or Risk Assessment*, in: Freie Universität Amsterdam (Hrsg.): Proceedings of the First BeNeLux conference on Software Architecture, abrufbar unter: http://www.few.vu.nl/~hans/publications/y1999/benelux99.pdf (17.01.2015)

[Mic13-1] Microsoft Corp. (2013): *Case Study: HarperCollins Publishers*, Redmond, abrufbar unter: http://www.microsoft.com/casestudies/Case_Study_Detail.aspx?CaseStudyID=710000003318 (17.01.2015)

[Mic13-2] Microsoft Corp. (2013): *Case Study: Lufthansa Systems*, Redmond, abrufbar unter: http://www.microsoft.com/casestudies/Windows-Server-2012-R2/Lufthansa-Systems-AG/Lufthansa-Systems-Uses-Hybrid-Cloud-to-Trim-IT-Delivery-to-Hours-and-Reduce-Costs/710000003533 (17.01.2015)

[Mic15] Microsoft Corp. (2015): *Microsoft Azure*, Redmond, abrufbar unter: http://www.azure.com (17.01.2015)

[Mic15-1] Microsoft Corp. (2015): *Microsoft Office 365 für zu Hause oder für Unternehmen – Office Online*, Redmond, abrufbar unter: http://office.microsoft.com/de-de/products/ (17.01.2015)

[Mic15-2] Microsoft Corp. (2015): *Microsoft Dynamics CRM-Lösungen*, Redmond, abrufbar unter: http://www.microsoft.com/de-de/dynamics/crm.aspx (17.01.2015)

[Mie10] Mietzner, Ralph (2010): *A Method and Implementation to Define and Provision Variable Composite Applications, and ist Usage in Cloud Computing*, Institut für Architektur von Anwendungssystemen, Universität Stuttgart, Stuttgart, abrufbar unter: http://elib.uni-stuttgart.de/opus/volltexte/2010/5614/pdf/Diss.pdf (17.01.2015)

[Mye09] Myerson, Judith M. (2009): *Cloud computing versus grid computing*, IBM Corporation, New York, abrufbar unter: http://download.boulder.ibm.com/ibmdl/pub/software/dw/web/wa-cloudgrid/wa-cloudgrid-pdf.pdf (17.01.2015)

[Naa12] Naab, Matthias (2012): *Enhancing Architecture Design Methods for Improved Flexibility in Long-Living Information Systems*, Fraunhofer Verlag, Kaiserslautern

[NBC+03] Nord, Robert L. et.al. (2003): *Integrating the Architecture Tradeoff Analysis Method (ATAM) with the Cost Benefit Analysis Method (CBAM)*, Carnegie Mellon, Software Engineering Institute, Pittsburgh, PA, abrufbar unter: http://www.sei.cmu.edu/reports/03tn038.pdf (17.01.2015)

[NSJ+08] Närman, Per; Schönherr, Marten; Johnson, Pontus et. al. (2008): *Using Enterprise Architecture Models for System Quality Analysis*, in: IEEE Computer Society (Hrsg.): Enterprise Distributed Object Computing Conference, 2008. EDOC '08, S.14-23, München

[NSS+09] Närman, Per; Sommestad, Teodor; Sandgren, Sofia et. al. (2009): *A Framework for Assessing the Cost of IT Investments*, KTH Royal Institute of Technology, Stockholm

[OG09] The Open Group (2009): *TOGAF Version 9 - Management Overview*, The Open Group, abrufbar unter: http://www.togaf.info/togafSlides91/TOGAF-V91-M1-Management-Overview.pdf (17.01.2015)

[OJK+13] Österlind, Magnus; Johnson, Pontus; Karnati, Kiran et. al. (2013): *Enterprise Architecture Evaluation Using Utility Theory*, in: 17th IEEE International Enterprise Distributed Object Computing Conference , KTH Royal Institute of Technology, Stockholm

[Ora15] Oracle, Inc. (2015): *Oracle IaaS*, abrufbar unter: http://www.oracle.com/de/cloud/iaas/cloud-iaas-main-2225449-de.html (17.01.2015)

[PM07] Pohl, Klaus; Metzger, Andreas (2007): *Variability Management in Software Product Line Engineering*, in: IEEE Computer Society (Hrsg.): Proceedings of the 28th International Converence on Software Engineering (ICSE'07 Companion), S.186-187, Minneapolis, MN, USA

[Pol09] Prof. Dr. Andreas Polze (2009): *A Comparative Analysis of Cloud Computing Environments*, Hasso-Plattner-Institute for Software Engineering, Potsdam, abrufbar unter: http://drngpasclibrary.pbworks.com/f/cloud-study.2.pdf (17.01.2015)

[Pra13] Praw, Jason (2013): *Barriers to Cloud Adoption study*, comScore Inc., Reston, Virginia, USA

[RG08] Roy, Banani; Graham, Nicholas (2008): *Methods for Evaluating Software Architecture: A Survey*, Queen's University at Kingston, Ontario, abrufbar unter: http://research.cs.queensu.ca/TechReports/Reports/2008-545.pdf (17.01.2015)

[RW07] Riebisch, Matthias; Wohlfarth, Sven (2007): *Introducing Impact Analysis for Architectural Decisions*, in: IEEE Computer Society (Hrsg.): 14th Annual IEEE International Conference and Workshops on the Engineering of Computer-Based Systems, 2007. ECBS '07., S.381-392

[Sal15] Salesforce.com (2015): *Force.com*, abrufbar unter: http://www.salesforce.com/platform/solutions/employee-apps/ (17.01.2015)

[Sal15-1] Salesforce.com (2015): *CRM-Lösungen und Cloud-Plattform - Salesforce.com Deutschland*, abrufbar unter: http://www.salesforce.com/de/crm/products.jsp (17.01.2015)

[SC06] Shaw, Mary; Clements, Paul (2006): *The Golden Age of Software Architecture*, in: IEEE Computer Society (Hrsg.): IEEE Software 2006, 23(2), S.31-39

[Sch10] Schlichter, Johann (2010): *Distributed Applications - Verteilte Anwendungen*, TU München, München, abrufbar unter: http://wwwschlichter.informatik.tu-muenchen.de/dokument.php?id_dokument=611 (17.01.2015)

[SMW+14] Schmidt, Rainer; Möhring, Michael; Wißotzki, Matthias u.a. (2014): *Towards a Framework for Enterprise Architecture Analytics*, in: IEEE Computer Society (Hrsg.): 18th International Enterprise Distributed Object Computing Conference Workshops and Demonstrations (EDOCW), S.266-275, Ulm

[Sta02] Starke, Gernot (2002): *Effektive Software-Architekturen - ein praktischer Leitfaden*, Carl Hanser Verlag, München

[Tan13] Tang, David (2013): *Why People Won't Buy Your Product Even Though It's Awesome*, The Flevy Blog, abrufbar unter: http://flevy.com/blog/why-people-wont-buy-your-product-even-though-its-awesome/ (17.01.2015)

[VM15] VMware (2015): *vCloud Suite*, abrufbar unter: http://www.vmware.com/de/products/vcloud-suite (17.01.2015)

[VRC+09] Vaquero, Luis; Rodero-Merino, Luis; Caceres, Juan u.a. (2009): *A Break in the Clouds: Towards a Cloud Definition*, in: ACM (Hrsg.): ACM SIGCOMM Computer Communication Review (Volume 39, Number 1, S.50-55), New York, abrufbar unter: http://ccr.sigcomm.org/online/files/p50-v39n1l-vaqueroA.pdf (17.01.2015)

[WBB+06] Wojcik, Rob; Bachmann, Felix; Bass, Len u.a. (2006): *Attribute-Driven Design (ADD)*, Carnegie Mellon, Software Engineering Institute, Pittsburgh, PA, abrufbar unter: http://www.sei.cmu.edu/reports/06tr023.pdf (17.01.2015)

[Wex15] WebEx (2015): *Online Meetings & Web Conferencing - WebEx*, abrufbar unter: http://www.webex.com/products/web-conferencing.html (17.01.2015)

[Zac87] Zachman, John (1987): *A framework for information systems architecture*, in: IBM Corporation (Hrsg.): IBM Systems Journal. 26, Nr. 3, 1987, S. 277-293

[ZCB10] Zhang, Qi; Cheng, Lu; Boutaba, Raouf (2010): *Cloud computing: state-of-the-art and research challenges*, in: Springer-Verlag (Hrsg.): Journal of Internet Services and Applications, Volume 1, Issue 1, S.7-18, Waterloo, Ontario, CA

[ZT05] Zayaraz, Godandapani; Thambidurai, P. (2005): *Software Architecture Selection Framework Based on Quality Attributes*, in: IEEE Computer Society (Hrsg.): IEEE Indicon 2005 Conference, Chennai

Anhang A: Glossar wichtiger Begriffe

ADD
Attribute Driven Design Method

Architectural Driver
Sammlung funktionaler, qualitativer und ökonomischer Anforderungen an eine Software-Achitektur (siehe [BCK03, S.154]).

Artifact
Artefakt eines Systems auf das ein Reiz von außen ausgeübt wird. Dies kann das Gesamtsystem oder Teile davon sein (siehe [BCK03, S.75]).

ATAM
Architecture Tradeoff Analysis Method

Business Driver
Sammlung aller Faktoren, die die Entwicklung eines Softwaresystems motivieren. Dazu gehören alle funktionalen, qualitativen und ökonomischen Anforderungen (Architecture Drivers), etwaige Constraints und alle vom System betroffenen Interessensgruppen (siehe [BCK03, S.277]

CBAM
Cost Benefit Analysis Method

Constraint
siehe Design Constraint

Design Constraint
Entwurfseinschränkungen, d.h. Entscheidungen zum Systementwurf, die (bereits bei der Formulierung der Anforderungen an das fertige System) vorweggenommen werden, d.h. genauso im finalen Entwurf des Systems enthalten sein müssen (siehe [WBB+06, S.7]).

Environment
Umweltbedingung, unter der ein System bei Eintreffen einen Reizes von außen arbeitet (siehe [BCK03, S.75]).

Funktionale Anforderung
Spezifikation einer Funktion, die ein System bereitstellen muss, um die Bedürfnisse der Nutzer zu befriedigen, wenn das System unter bestimmten Rahmenbedingungen eingesetzt wird (siehe [WBB+06, S.7]).

Non-Risk
Architekturrelevante Entscheidung in einem Architekturentwurf, die bereits getroffen

wurde, und deren Konsequenzen vergleichsweise gut abgeschätzt werden können (siehe [BCK03, S.275]).

Quality Attribute
engl. Qualitätsattribut; Aussage zur Qualität eines Softwaresystems (siehe [BCK03, S.72]). Diese wird häufig in der „Stimulus-Response"-Notation beschrieben (siehe [WBB+06, S.11]).

Response Measure
Maßeinheit für die Reaktion des Systems. Diese sollte messbar sein, um das System in Bezug auf seine Anforderungen testen zu können (siehe [BCK03, S.75]).

Response
Reaktion des Systems auf ein von außen eintreffenden Reiz (siehe [BCK03, S.75]).

Risk
Architekturrelevante Entscheidungen in einem Architekturentwurf, die entweder noch nicht getroffen wurden oder bereits getroffen wurden, deren Konsequenzen aber noch nicht abgeschätzt werden bzw. in Zukunft Probleme einiger Qualitätsmerkmale bereiten können (siehe [KKC00, S.3]).

Sensitivity Point
Element einer Softwarearchitektur, dessen Ausgestaltung hohe Korrelation zu einem messbaren Qualitätsattribut aufweist (siehe [KKC00, S.3]).

Stimulus Source
engl. Reizquelle; eine systemexterne Einheit (z.B. ein Anwender, Computersystem oder anderer Akteur), der einen Reiz auf das System generiert (siehe [BCK03, S.75]).

Stimulus
engl. Reiz; Einwirkung auf ein Softwaresystem von außen, auf die durch das System eine Reaktion generiert werden soll (siehe [BCK03, S.75]).

Tradeoff Point
Element einer Softwarearchitektur, das mehrere Sensitivity Points vereint, bei dem die Ausgestaltung aber unterschiedliche Auswirkungen auf die betroffenen Qualitätsattribute besitzt, d.h. die bessere Erfüllung eines Attributs hat negative Auswirkungen auf ein anderes Attribut (siehe [KKC00, S.3]).

Anhang B: Bewertungsmethoden für Softwarearchitekturen

Kriterien für die Auswahl von Bewertungsmethoden

Es gibt derzeit am Markt eine Vielzahl an Bewertungsmethoden für Software-Architekturen – jede von ihnen mit bestimmten Zielen entwickelt: beispielsweise die Bestimmung von Risiken im Entwurf (SAAM, ATAM, …), des Return-on-Investment von Entwurfsentscheidungen (CBAM, …), allgemeine Vorhersage der Systemqualität (SAEM, …). Auch unterscheiden sie sich im Zeitpunkt für deren Anwendung, den beteiligten Personen und vielen weiteren Attributen. Es gibt deshalb einige Vergleichsstudien (z.B. [BN12], [EHM07], [DN02], [AZJ04]), die Methoden gegenüberstellen und somit bei der Auswahl der bestgeeigneten Methoden helfen können. Tabelle B-1 und Tabelle B-2 zeigen einen Ausschnitt aus den Vergleichstabellen der genannten Studien.

Im Rahmen dieser Arbeit wurden mit ATAM und CBAM zwei Methoden als Basis genutzt, weiterentwickelt und um eine formale Beschreibungsmethode für die Entwurfs- und Bewertungsartefakte erweitert. Für die Auswahl und Kombination genau dieser Methoden aus der Vielzahl an Alternativen gab es mehrere Gründe:

- Die Methoden beschränken sich nicht auf bestimmte Qualitätsanforderungen; insbesondere Kosten und Nutzen werden bewertet.
- Beziehungen zwischen Merkmalen werden analysiert und in Form von Risiken, Sensitivity Points und Tradeoff Points aufbereitet.
- Für die Anwendung der Methode wird nur die Beteiligung der betreffenden Stakeholder vorausgesetzt; für CBAM wird ohnehin eine teilweise Durchführung der ATAM benötigt.
- Die Ziele der Methode decken sich mit den Zielen dieser Arbeit, d.h. für Architekturalternativen wird der ROI ermittelt und als Vergleichsmaßstab verwendet.
- Die Methoden werden in einer frühen Projektphase, d.h. unmittelbar nach dem Entwurf der Architektur durchgeführt.
- Die Evaluierungstechnik setzt in beiden Methoden auf Szenarien.
- Beteiligung der Stakeholder ist explizit gewünscht; dies ist eine zentrale Anforderung der in dieser Arbeit entwickelten Gesamtmethode um Offenheit, Transparenz und Diskussionskultur unter den Beteiligten zu fördern.
- Die Anforderungen an die Expertisen des Evaluierungsteams sind, obwohl sie nicht gering sind, vertretbar.

Überblick über Bewertungsmethoden nach [BN12], [EHM07], [DN02] und [AZJ04]

Kriterium	SAAM	ATAM	SBAR	CBAM
Qualitäts-anforderungen	Modifizierbarkeit	Verschiedene	Entwurfs-tauglichkeit	Kosten und Nutzen
Beziehungen zwischen Merkmalen	Nein	Ja	Ja	Nein
Voraussetzungen für die Anwendung	Keine	Bereitstellung von Ressourcen für Interviews und Analysen	Implementierung der Architektur	Zum Teil Vorab-ausführung der ATAM
Ziele der Methode	Tauglichkeit, Risiken	Sensitivity Points, Tradeoff Points, Risks, Non-Risks	Bestimmung des Potenzials der Erreichung der Anforderungen	Bestimmung der ROIs von Architekturalternativen
Projektphase	Nach Abschluss des Architektur-entwurfs	Nach Abschluss des Architektur-entwurfs	Spät, im Rahmen eines Re-engineerings	Nach Abschluss des Architektur-entwurfs (bzw. nach ATAM)
Evaluierungs-technik	Szenarios	Utility-Baum, Szenarios	Abhängig von der Anforderung, Szenarios, math. Modelle, Simulation, Interpretation	Nutzenbewertung von Szenarien, Kostenmodell
Beteiligung Stakeholder	Ja	Ja	Nein, nur Architekten	Ja
Anforderung an das Evaluierungsteam	Gering, leicht anzuwenden	Hoch, Expertise im Erzeugen eines Utility-Baums, Identifikation von Architektur-alternativen	Hoch, durch Auswahl des anforderungsabhängigen Bewertungsansatzes	Relativ hoch, durch Abschätzung von Nutzen und Kosten

Tabelle B-1: Vergleich verschiedener Bewertungsmethoden (Teil 1)

Kriterium	ARID	SACAM	SAEM	AQA
Qualitäts-merkmale	Entwurfs-tauglichkeit	Abgeleitet von Geschäftszielen	Externe und interne Qualitäts-anfoderungen	Verständlichkeit, Machbarkeit, Offenheit, Wartbarkeit, Erweiterbarkeit und Kunden-zu-friedenheit
Beziehungen zwischen Merkmalen	Nein	Nein	Nein	Nein
Voraussetzun-gen für die Anwendung	Keine	Identifikation der Geschäftsziele, Auswahl Archi-tekturkandidaten	Keine	Anforderungsana-lyse
Ziele der Me-thode	Bestimmung der Entwurfs-tauglichkeit	Vergleich von Ar-chitekturkandida-ten	Vorhersage Sys-temqualität	Unterstützung für Entscheidungs-träger
Projektphase	Früh, beim Ent-wurf der Kompo-nenten	Früh	Spät, da metrische Methode	Spät, da metrische Methode
Evaluierungs-technik	Szenarien	Kriterien, Szenarios, Direktiven, Scoringmodell	Metriken, GQM	Befragungs-technik
Beteiligung Stakeholder	Ja	Ja	Nein	Nein
Anforderung an das Evalu-ierungsteam	Hohe Anforde-rungen, Erfah-rung zur Bewer-tung der Ansätze erforderlich	Wirtschaftliches Verständnis erfor-derlich, da Ablei-tung der Metriken von den Ge-schäftszielen	Expertenwissen zur Abbildung von Metriken auf interne Attribute	Hohe Anforde-rungen an Archi-tekten, da Befra-gung in Bezug auf die Maße

Tabelle B-2: Vergleich verschiedener Bewertungsmethoden (Teil 2)

Auswahlkriterien nach [EHM08]

Eicker u.a. stellen in [EHM08] einen Entscheidungsbaum (siehe Abb. B-1) auf, der anhand des oder der zu untersuchenden Qualitätsmerkmale eine geeignete Analysemethode empfiehlt. Auch dieser führt zu den in dieser Arbeit verwendeten Methoden. Zunächst sollen mehrere Qualitätsmerkmale und deren Beziehungen untereinander untersucht werden. Hier kommt

die ATAM in die engere Wahl. Für den Return-on-Investment-Vergleich werden Kosten und Nutzen untersucht, was wiederum zur CBAM führt.

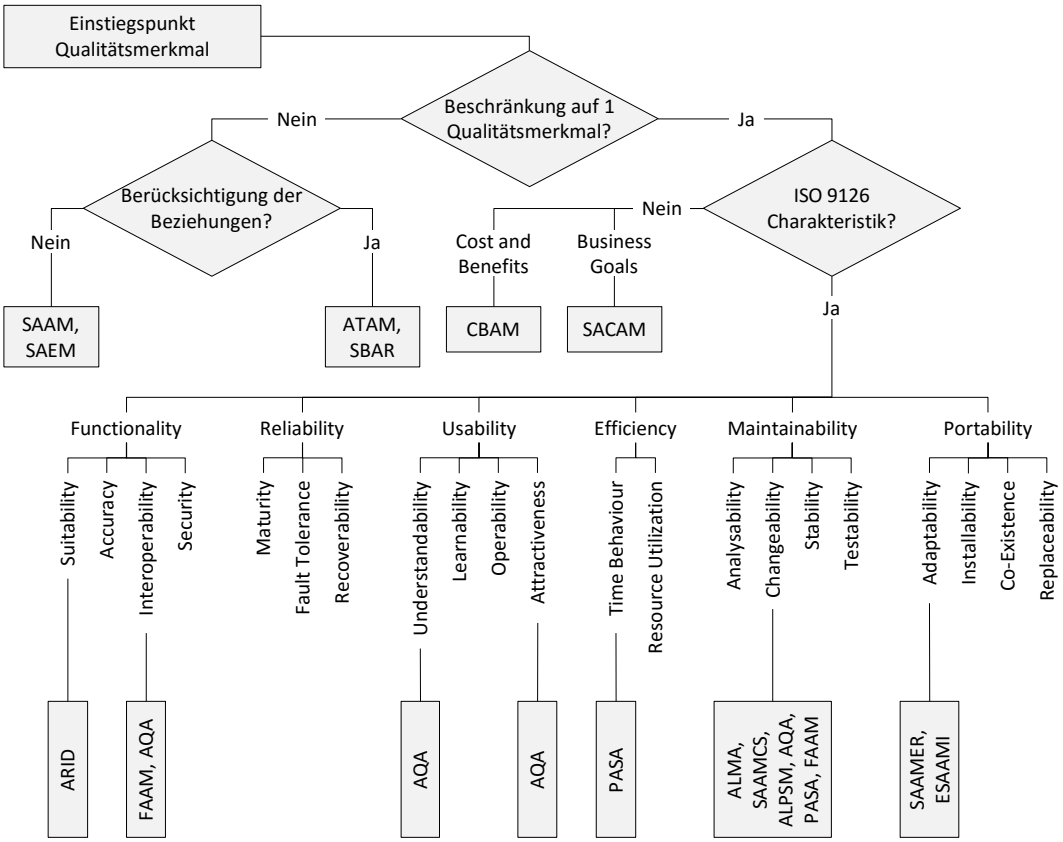

Abb. B-1 Auswahl von Bewertungsmethoden nach [EHM08]

Anhang C: Cloud Angebote (PaaS) im Vergleich

	Amazon	Microsoft	Google
Rechenleistung			
Runtime	Elastic Compute Cloud (EC2)	Virtual Machines, Cloud Services, Web Apps	Compute Engine, App Engine, Container Engine
Big Data und Analytics	Redshift, Elastic MapReduce, Kinesis	HDInsight, Machine Learning, Stream Analytics	BigQuery, Dataflow, Prediction API
Netzwerk- und Integrationsdienste			
VPN	Virtual Private Cloud, Direct Connect	Virtual Network	Direct Peering, VPN
Internet **Service Bus**	AWS Lambda	Service Bus, BizTalk Services, Event Hub	Cloud Pub/Sub
Speicherdienste			
Speicher	Simple Storage Service, Glacier	Page Blobs, Block Blobs	Blobstore API
Cache	Cloud Front, ElastiCache	CDN, Redis Cache	
Persistent Drive	Elastic Block Store (EBS)	Azure Disks	
Datenbankdienste			
Datenbank (SQL)	Relational Database Service (RDS), DynamoDB	SQL Database	
Datenbank (No-SQL)	SimpleDB	DocumentDB, Table Service	Datastore, Storage
Anwendungsdienste			
Messaging	Simple Queue Service (SQS)	Queue Service	Task Queue API
Notification	Simple Notification Service (SNS)	Notification Hub	
Identity Service	Directory Service	Active Directory	Google Accounts
E-Mail	Simple Email Service	SendGrid	Mail API
Application Service APIs	Elastic Transcoder	Media Services, Bing Search, Bing Translator	Google Search, Translate API
Managementdienste			
	CloudWatch, Beanstalk, OpsWorks, Cloud Formation	Application Insights, Automation, Operational Insights	

Tabelle C-1: Cloud-Angebote (PaaS) im Vergleich (Ausschnitt)