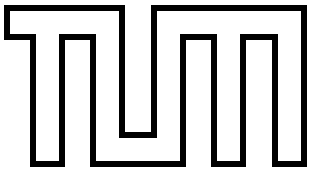


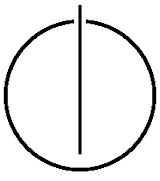
FAKULTÄT FÜR INFORMATIK

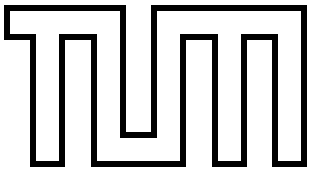
DER TECHNISCHEN UNIVERSITÄT MÜNCHEN

Bachelorarbeit in Wirtschaftsinformatik

Entwicklung einer Methode zur Bewertung der Transformierbarkeit von On-Premise Anwendungssystemen in Software as a Service Lösungen

Julian Merkl





FAKULTÄT FÜR INFORMATIK

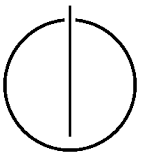
DER TECHNISCHEN UNIVERSITÄT MÜNCHEN

Bachelorarbeit in Wirtschaftsinformatik

Entwicklung einer Methode zur Bewertung der Transformierbarkeit von On-Premise Anwendungssystemen in Software as a Service Lösungen

Developing a method to assess transformability of on-premise application systems into software as a service solutions

Bearbeiter:	Julian Merkl
Aufgabensteller:	Prof. Dr. rer. nat. Florian Matthes
Betreuer:	M. Sc. Alexander Steinhoff
Abgabedatum:	15. Oktober 2012



Ich versichere, dass ich diese Bachelorarbeit selbständig verfasst und nur die angegebenen Quellen und Hilfsmittel verwendet habe.

München, den 15. Oktober 2012 Julian Merkl

Danksagung

Die folgende Arbeit wurde in Kooperation mit der msg systems AG erstellt. Besonderer Dank gilt daher den Mitarbeitern, die ihr Expertenwissen und ihre Arbeitszeit bei der Durchführung des Bachelorprojekts zur Verfügung gestellt haben. Vorweg bedanke ich mich herzlichst bei meinen Betreuern Jürgen Biebl, Peter Huber und Ralf Engelschall für die kollegiale Unterstützung.

Auch für die hervorragende Betreuung seitens des Lehrstuhls durch Alexander Steinhoff möchte ich mich ausdrücklich bedanken. Ihm wünsche ich weiterhin viel Durchhaltevermögen in den letzten Monaten seiner Promotion und noch mehr Erfolg für seinen weiteren Werdegang.

Kurzreferat

Längst ist *Cloud Computing* nicht mehr nur ein Hype, sondern steht auf der Agenda vieler Unternehmen. Zahlreiche Anwendungssysteme sind als *Software as a Service* Lösung erhältlich und dabei sind es nicht nur Neuentwicklungen, die den Markt beflügeln. Dennoch stehen Hersteller traditioneller Software häufig vor dem Problem, die vorhandene Lösung in die *Cloud* zu portieren. Um den Prozess zur Transformation vorab analysieren und bewerten zu können, bedarf es einer strukturierten Vorgehensweise. Voraussetzung für eine aussagekräftige Analyse ist ein ganzheitliches Wissensmanagement, das die Expertise zur Entwicklung einer *SaaS*-Lösung bündelt und einen zielgerichteten Zugang zum Wissen ermöglicht.

Der Fokus dieser Arbeit liegt auf der Entwicklung einer Methode, die als Werkzeug zum Wissensmanagement herangezogen werden kann, um die Transformierbarkeit eines *On-Premise* Anwendungssystems zu bewerten. Zentrales Element der Methode, die in Kooperation mit der *msg systems AG* entwickelt wurde, ist eine Daten- und Filterstruktur. Damit lassen sich Lösungsansätze zur Erfüllung der *SaaS*-Kriterien, wie der elastischen Bereitstellung oder dem IP-basierten Zugriff, in einer Matrix klassifizieren. Die Dimensionen der Matrix werden durch gerichtete Graphen aufgespannt, die die essentiellen Kriterien zu konkreten Anforderungen, beispielsweise die Skalierbarkeit unter einem bestimmten Lastprofil, spezialisieren. Bei der Anforderungsspezifikation filtert der Anwender relevante Pfade.

Da *Cloud Computing* ein interdisziplinäres Modell zur Bereitstellung von IT-Ressourcen darstellt, ermöglicht die Methode eine Betrachtung der Transformation aus unterschiedlichen Blickwinkeln. Dazu liefert sie konkrete Handlungsempfehlungen, deren Realisierbarkeit im Rahmen der Projektstudie zu bewerten ist.

Bei der Identifikation von Lösungsansätzen konnten auch elementare Charakteristiken zur Spezifikation einer *SaaS*-Lösung ermittelt werden. Neben klassischen Merkmalen, wie Lastprofilen oder Metriken zur Abrechnung, ist auch der angestrebte Reifegrad von Skalierbarkeit, Erweiterbarkeit oder die gemeinsame Nutzung der Ressourcen mit Dritten, von entscheidender Bedeutung.

Die Methode wurde beim Kooperationspartner exemplarisch eingeführt und soll dort weiterentwickelt und validiert werden. Bereits bei dieser ersten Implementierung konnte eine interdisziplinäre Anwendbarkeit demonstriert werden.

Schlagwörter: Methode, Software as a Service, Cloud Computing, Bewertung, Transformation, Adaption, Portierung, Anwendungssystem, Machbarkeitsstudie, Wissensmanagement

Abstract

Cloud computing can be no longer considered just a hype, as nowadays it is part of the agenda of many companies. Numerous application systems are available as software as a service solutions, however, not only are new developments spurring the market. Yet, traditional software vendors are often confronted with the problem of porting their existing solution into the cloud. In order to analyze and evaluate the process of transformation in advance, a structured approach is required. Holistic knowledge management, combining expertise to develop a SaaS solution and enablement of a targeted access to knowledge, is a basic prerequisite for a meaningful analysis.

The focus of this work is to develop a method, which can be used as a knowledge management tool to evaluate the transformability of on-premise application systems. The basic element of the method, which was developed in cooperation with the msg systems AG, is a data- and filtering structure. In this way solution approaches to fulfill the SaaS criteria, such as the elastic provision or IP-based access, can be classified in a matrix. The dimensions of the matrix can be spanned by directed graphs that specialize the essential criteria to specific requirements, for example scalability under a certain load profile. By specifying his requirements, the user filters relevant paths.

Since cloud computing is an interdisciplinary model to provide IT resources, the method allows considerations of the transformation from different perspectives. Moreover, it

provides specific recommendations for action, the feasibility of which must be assessed in the study.

By identifying possible solution approaches, fundamental characteristics for the specification of a SaaS solution could be determined as well. Beside traditional attributes such as load profiles or billing metrics, also the maturity level of scalability, extensibility, or the sharing of resources with others, is crucial.

The method was exemplarily introduced at the cooperation partner. There it shall be further developed and validated. In this first implementation a interdisciplinary applicability could be demonstrated already.

Keywords: method, software as a service, cloud computing, assessment, transformation, adaption, porting, application system, feasibility study, knowledge management

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Abkürzungsverzeichnis

AJAX	Asynchronous JavaScript and XML
API	Application Programming Interface
ASP	Application Service Providing
AWS	Amazon Web Services
BITKOM	Bundesverband Informationswirtschaft, Telekommunikation und neue Medien
BSI	Bundesamt für Sicherheit in der Informationstechnik
CaaS	Communication as a Service
CEO	Chief Executive Officer
CIO	Chief Information Officer
CRM	Customer Relationship Management
CTO	Chief Technical Officer
DaaS	Desktop as a Service
DBMS	Datenbank Management System
EC2	Elastic Compute Cloud
EDI	Electronic Data Interchange
ENISA	European Network and Information Security Agency
ERP	Enterprise Resource Planning
HR	Human Resources
HTML	Hypertext Markup Language
IaaS	Infrastructure as a Service
IBM	International Business Machines Corporation)
IDC	International Data Corporation
IKT	Informations- und Kommunikationstechnologie
IP	Internet Protokoll
ISO	International Organization for Standardization
ISV	Independant Software Vendor
IT	Informationstechnologie
msg	msg systems AG
NIST	National Institute of Standards and Technology
PaaS	Platform as a Service
SaaS	Software as a Service
SAP	Systemanalyse und Programmentwicklung AG
SecaaS	Security as a Service
SLA	Service Level Agreement
SOA	Service-orientierte Architektur
SQL	Simple Query Language
S3	Simple Storage Service
UI	User Interface
XaaS	Everything as a Service

1. Einleitung

Das erste Kapitel beschreibt die aktuelle Entwicklung des Trends *Cloud Computing* und zeigt welche Problemstellungen sich in Wirtschaft und Wissenschaft daraus ergeben. Aus dieser Motivation heraus entstand auch die Aufgabenstellung dieser Arbeit.

1.1. Motivation

Schenkt man dem jährlich erscheinenden „Hype Cycle of Emerging Technologies“ des Analysten und Marktforschers Gartner glauben, so durchläuft jeder Technologie-Trend nach dem großen Hype hinsichtlich seiner Marktpräsenz zunächst ein tiefes Tal der Desillusionierung bis er sich schließlich etabliert und die Technologie auch produktiv eingesetzt wird.

Seit dem Jahr 2008 ist *Cloud Computing* ein solcher Trend. Skeptikern war der Begriff ein weiteres *Buzzword*, das der Marketing-Maschinerie entsprungen war. Für Befürworter der Technologie war es der Beginn einer neuen Ära, die das *Computing* gänzlich revolutionieren würde. Auch hier zu Lande wurde *Cloud Computing* – gerade im Mittelstand – oft als Wiederauflage längst bekannter Geschäftsmodelle und somit als alter Wein in neuen Schläuchen betitelt [Hau11]. Selbst „Fortune Global 500“-CEOs wie Larry Ellison belächelten den Hype. Dieser sogar mit den Worten es sei nur ein idiotischer Begriff für etwas, was man schon Jahre mache [Eva10].

Heute im Jahr 2012 steht Cloud Computing kurz vor dem zu erwartenden Tal [Hü10]. Die einstigen Skeptiker sind größtenteils verschwunden. Der Begriff ist dennoch in aller Munde, allerdings nicht mehr in derer die selbst der IT-Branche angehören, um ihn zu preisen oder eben an den Pranger zu stellen. Vielmehr verschwinden die CTOs und CIOs dieser Welt im stillen Kämmerchen und diskutieren wie und wo man nun davon profitieren könnte [kpm12]. Nahezu alle *Global Player* der Industrie haben umfassende *Cloud*-Strategien entwickelt. Marktforscher schärfen ihre Umsatzprognosen für den Wachstumsmarkt. Für die Pioniere des *Cloud Computings* wie *Salesforce.com*, *Amazon* oder *Google* hat sich der einstige Trend längst zum nicht mehr weg zu denkenden Tagesgeschäft entwickelt. Doch sind es vor allem zahlreiche Startups, die die Gunst der Stunde nutzten und sich mit neuartigen Geschäftsmodellen aus der *Cloud* einen Namen machen und schon bald den ein oder anderen Branchen-Primus vom Thron stoßen könnten [for11]. Selbst Marktführer wie die SAP hatten und haben noch immer ihre Schwierigkeiten ihre vorhandenen Lösungen in die *Cloud* zu portieren [Rep12]. So scheint es als würde das ersehnte Kundenversprechen, nämlich mehr Agilität im Tagesgeschäft durch flexible Nutzung von Informationstechnologie (IT)-Ressourcen zu erreichen, Entwickler traditioneller Software-Lösungen beim Wandel zum Cloud-Anbieter in den eigenen Strukturen selbst vorzeitig disqualifizieren oder zumindest vor eine große Hürde stellen.

Doch nicht nur aus technologischer Sicht zeigen sich die Unternehmen beim Einsatz von

Cloud-Technologien teilweise noch immer sehr verhalten. Gerade in den Ländern in denen Datenschutz groß geschrieben wird. Schließlich geht es ja darum über das Internet auf seine Daten und Anwendungen zuzugreifen. Noch dazu sollen die Infrastrukturen mit Dritten gemeinsam genutzt werden. Das zeigt, dass *Cloud Computing* in vielen Bereichen ein Umdenken und zahlreiche Anpassungen erfordert und somit ein interdisziplinäres Konzept zur Bereitstellung von Informations- und Kommunikationstechnologie (IKT) statt nur eine neuartige Basis-Technologie darstellt. Letzteres macht die Idee, seine IT-Services in die *Cloud* zu verlagern, um von den bekannten Vorteilen zu profitieren, zu einem komplexen Unterfangen. Und das gilt für alle Marktteilnehmer – für Produzenten wie auch Konsumenten.

Dennoch möchte man möglichst sofort von der vielversprechenden Technologie profitieren. Was mangelnde Datenschutzrichtlinien, fehlende Sicherheitsmaßnahmen oder die unzureichende Gewährleistung von *SLAs* im Internet vorenthalten wird einfach – als private *Cloud* – im eigenen Unternehmen aufgebaut. Und so stehen plötzlich auch interne Anwendungen auf dem Prüfstand der „Cloud-Readiness“ [Pau10].

1.2. Aufgabenstellung und Zielsetzung

Die *msg systems AG* mit Hauptsitz in Ismaning bei München berät als international agierender Dienstleister Kunden unterschiedlicher Branchen bei der Umsetzung von derartigen *Cloud*-Projekten. Dabei sind alle Konstellationen denkbar. So unterstützten die Berater *ISVs* auf dem Weg selbst *Software as a Service*-Anbieter zu werden, Kunden, die deren Angebote in Anspruch nehmen wollen oder interne IT-Abteilungen beim Aufbau von *Private Clouds*. Und auch selbst möchte *msg* zukünftig eigene Lösungen als *Cloud-Service* anbieten.

In der Querschnittsabteilung *Applied Technology Research* bündelt das Unternehmen technologische und methodische Expertise. So sollen Expertenwissen und Projekterfahrungen aus allen Geschäftsbereichen zusammenfließen und zukünftigen Projekten zur Verfügung gestellt werden [msg]. Zudem unterliegt die Abteilung dem Forschungsauftrag innovative Technologien frühzeitig zu erproben und methodische Vorgehensweisen für Projekte zu entwickeln.

In Kooperation mit dem Lehrstuhl für *Software Engineering betrieblicher Informationssysteme* am Institut für Informatik der Technischen Universität München beschäftigt man sich seit längerer Zeit mit der Problemstellung, bestehende Anwendungssysteme in einen *Cloud-Service*, also in *Software as a Service*, umzuwandeln. Im Unternehmen entstand zudem der Bedarf, eine strukturierte und nachvollziehbare Bewertungsmöglichkeit für derartige *Cloud*-Projekte heranziehen zu können.

Im Rahmen dieser Abschlussarbeit im Studiengang Wirtschaftsinformatik soll eine Methode entwickelt werden, die als Hilfsmittel zur Entscheidungsfindung die Bewertung der Transformierbarkeit eines bestehenden Anwendungssystems in eine *Software as a Service (SaaS)* Lösung unterstützt. Bei *msg* soll die Methode außerdem genutzt werden, um den Entscheidungsprozess über die Durchführung einer solchen Transformation zu führen und zu standardisieren, um vergleichbare Entscheidungsgrundlagen für kommende Projekte zu schaffen und den Lernkurveneffekt zu verstärken.

Da die *msg systems AG* in zahlreichen Branchen vertreten ist, soll die Methode möglichst generisch, dass heißt allgemein anwendbar hinsichtlich der Branche und der Art des An-

wendungssystems sein. Teil dieser Bachelorarbeit ist auch die Methode als Werkzeug des Wissensmanagements im Unternehmen exemplarisch einzuführen. Die Ergebnisse sollen zudem in Form einer vereinfachten Tool-Unterstützung anwendbar sein.

Der Fokus dieser Arbeit liegt jedoch auf der Methodik selbst. Es soll eine formale Vorgehensweise zur Lösung der genannten Problemstellung erarbeitet werden. Eine ganzheitliches Projekt zur Einführung und Etablierung im Unternehmen wird nicht durchgeführt und so kann die Methode auch nur exemplarisch angewendet und validiert werden. Ein Fallbeispiel ist ebenso nicht Teil der Arbeit.

Bevor die Vorgehensweise zur Methodenentwicklung und die Ergebnisse vorgestellt werden, wird zunächst eine gemeinsame Terminologie eingeführt sowie die nötigen Grundlagen vermittelt.

2. Theoretischer Hintergrund

Im folgenden Kapitel werden die Hintergründe und Grundlagen zu *Cloud Computing* veranschaulicht sowie dazugehörige Begriffe erläutert. Anschließend werden die Service-Modelle und Bereitstellungsarten vorgestellt.

2.1. Terminologie und Grundlagen zu Cloud Computing

Wie schon zu Beginn der Arbeit erwähnt, handelt es sich beim *Cloud Computing* nicht nur um eine neue oder gar revolutionäre Technologie. Vielmehr handelt es sich um eine Evolution des *Computings*, also der Art und Weise wie wir Informationen speichern, verarbeiten oder Dritten zur Verfügung stellen [bsia]. Der technologische Fortschritt beschreibt die erste Säule der Entwicklung. Dabei lassen sich die Konzepte des *Cloud Computings* allesamt auf wohlbekannte Entwicklungslinien der IKT zurückführen. So hat sich beispielsweise die Verteilung von Daten, Anwendungen und ihre Darstellung im Vergleich zum *Client/Server-Computing* erneut geändert [KS93]. In den siebziger Jahren wurden die Daten noch auf zentralen Großrechnern verarbeitet. Die Ein- und Ausgabe von Informationen erfolgte beim *Mainframe Computing* an Terminals ohne jegliche Intelligenz. Die stetige Kapazitätssteigerung und der damit verbundene Einzug des *Personal Computings* führten dazu, dass sich Anwendungen hin zum Client verlagerten. Auch Netzwerke unterlagen dieser Kapazitätssteigerung wodurch die zuvor genannte Verlagerung überhaupt erst möglich wurde.

Eine weitere Entwicklungslinie, nämlich die zunehmende Mobilität getrieben durch die Globalisierung, sowie der Ausbau des Internets führte zu Beginn des 21. Jahrhunderts wieder zu einer erneuten Zentralisierung. Die ersten webbasierten Anwendungen, wie Portale und *E-Business* Plattformen entstanden. Die Datenverarbeitung erfolgte serverseitig. Auf dem Client, hier dem Web-Browser, erfolgte lediglich die Präsentation. In der Geschichte der Informationstechnik lassen sich also bereits mehrere Evolutionszyklen dieser Art beobachten [BLRK10].

Parallel zu diesen technologischen Entwicklungen änderten sich über die letzten 15 bis 20 Jahre aber auch die Konzepte zur Bereitstellung von IT-Infrastrukturen und der sich darauf befindlichen Anwendungen sowie deren Beschaffung (engl. sourcing). Dies stellt die zweite Säule dar. So formte der Bedarf an hohen Rechenkapazitäten bei mangelnder Verfügbarkeit von *Super Computern* – vorwiegend im wissenschaftlichen Umfeld – die Idee des *Grid Computings*. Dabei werden geographisch verteilte Kapazitäten in einem Pool vereint und bedarfsgerecht in virtuelle Organisationen partitioniert. Ungenutzte Rechenleistung kann so Dritten zur Verfügung gestellt werden [ZCZH10]. Die Vorstellung des *Utility Computings*, dem das Nachhaltigkeitsprinzip unterliegt, wurde bereits Mitte der 60er-Jahre geboren. IT-Ressourcen sollen demnach nur im tatsächlich benötigten Maße als öffentlich verwaltetes Gut bereitgestellt und auch vergütet werden wie man es von Versorgungsein-

heiten (engl. utilities) wie Wasser, Strom und Gas gewöhnt ist [LMHL06].

Auch aus wirtschaftlicher Sicht entwickelte sich der Betrieb von Anwendungen im letzten Jahrzehnt weg vom reinen „Do-It-Yourself"-Geschäft. Der globale Trend des Outsourcings [garb], mit dem Hintergrund sich auf die eigenen Kernkompetenzen der Unternehmenswertschöpfung konzentrieren zu können, brachte bereits einige Geschäftsmodelle zur Auslagerung des Anwendungsbetriebs hervor. Bereits Ende der 1990er-Jahre kursierte das Modell des *Application Service Providing (ASP)* [Lev07]. Dabei betreibt ein Dienstleister die Applikation auf eigenen Systemen im Rechenzentrum und stellt diese dem Kunden samt Lizenzen über das Internet bereit.

ASP konnte sich seiner Zeit nicht durchsetzen. Die *Giga Information Group* (mittlerweile *Forrester Research* [hei03]) nannte im Jahr 2001, nachdem das Scheitern offensichtlich war, vor allem die schlechte Skalierbarkeit des Geschäftsmodells bei hoher Kostenintensität und mangelnde Anpass- und Konfigurierbarkeit der Anwendungen als Ursache. [com01] Weitere Gründe waren die nicht-flächendeckende Verfügbarkeit von Breitband-Internet, sowie die Bereitschaft der Unternehmen, kritische Daten in fremde Hände zu geben. Dennoch gilt *ASP* als Vorreiter für die heutigen „as a Service"-Modelle.

Nicht zuletzt haben sich seitdem auch die für das *Cloud Computing* vorausgesetzten Basis-Technologien weiterentwickelt. So bietet die Virtualisierung nun die nötige Flexibilität bei der dynamischen Zuweisung von Ressourcen und ermöglicht eine deutliche Effizienzsteigerung bezüglich der Auslastung der Hardware und damit einen wirtschaftlicheren Betrieb. Service-orientierte Architekturen und der Einsatz von *Web-Services* ermöglichen die Wiederverwendung von Komponenten bei ausreichender Konfigurationsmöglichkeit. Web-basierte Anwendungen profitieren von der verbesserten Breitbandversorgung ihrer Nutzer. So ermöglichen Schlüssel-Technologien wie *AJAX* oder *HTML*5 den Aufbau anspruchsvoller Benutzeroberflächen.

Die dritte Säule bei der Evolution des *Cloud Computings* geht aus der demografischen Entwicklung hervor. Mit dem Einzug der so genannten *Net Generation* in die Unternehmen unterliegt die Arbeitswelt derzeit nicht nur einem Generationswechsel, sondern auch einem Paradigmenwechsel. Nach Tapscott lässt sich diese Generation anhand von acht Normen charakterisieren [Tap09]:

- Innovation
- Freiheit
- Individualität
- Kritikfähigkeit
- Integrität
- Zusammenarbeit
- Unterhaltung
- Geschwindigkeit

Demnach ist es unter anderem der Wunsch nach Freiheit und Individualität, der die Anwender dazu leitet Informationen immer und überall verarbeiten zu wollen. Die zunehmende Kritikfähigkeit fördert das Bewusstsein, persönliche Daten im Internet preis zu

geben. Technische Innovationen werden sofort angenommen. Die *Consumerization of IT* beschreibt genau diesen Wandel [Kra12]. Informations- und Kommunikationstechnologie wird einfach nur noch konsumiert. Wo, wie und wann die Daten verarbeitet werden ist für den Anwender nicht von Belang. Was im Privatleben Gang und Gebe ist wird mehr und mehr auch auf das Berufsleben und damit in die Unternehmen projiziert. Aktuelle Trends wie *Bring your own device* sind der Inbegriff für die fortschreitende Verschmelzung von Beruf und Privatleben.

Beim *Cloud Computing* treffen also technologische, ökonomische aber auch soziologische Entwicklungslinien aufeinander. Gerade aus diesem Grund handelt es sich auch um ein interdisziplinäres Konstrukt.

Seinen Namen hat die Rechnerwolke von der zuvor beschriebenen Erkenntnis, dass Ort und Stelle der tatsächlichen Datenverarbeitung in den stetig komplexer werdenden IT-Landschaften immer intransparenter werden und es letztendlich auch immer weniger im Fokus des Anwenders liegt. Wusste man beim *web-based Computing* noch genau, auf welchem Server und an welchem Standort die genutzte Anwendung läuft, so ist das beim *Cloud Computing* wohl nur noch selten der Fall. Zumal die flexible Allokation und Kapselung von Ressourcen in virtuellen, verteilten Infrastrukturen eine gewisse Dynamik ins Spiel bringen. Wie in der Einleitung beschrieben nicht immer zur Begeisterung der Datenschützer. Inspiriert vom Symbol der Wolke, das bei der grafischen Beschreibung von Informationssystemen seit je für grenzenlose Netzwerke wie dem Internet verwendet wird, wurde der Begriff *Cloud Computing* im heutigen Kontext eines *Computing*-Modells im Jahr 2006 präsent. Über die Urheberschaft der Namensgebung ist man sich nicht immer einig [Wil08].

Eric Schmidt, damaliger CEO von *Google*, nutzte den Begriff zur Beschreibung des *Computing*-Modells, dem die Suchmaschine und die damals schon gewachsene Anzahl an *SaaS*-Anwendungen, den *Google Apps*, zu Grunde liegt [SS06].

Salesforce.com gilt mit seinem CRM als Pionier für *Software as a Service*. Im Jahr der Markteinführung 1999 tauchten erste wissenschaftliche Publikationen zur neuen Form der internetbasierten Bereitstellung von Software auf. Zunächst wurde das Software-Konzept unter dem Titel „*Service-based software*“ geführt [BLB+00]. Im Jahr 2001 veröffentlichte die *Software & Information Industry Association* dann die Studie „Software As A Service: Strategic Backgrounder“. Populär wurde der Begriff mit der SD Forum Konferenz im Jahr 2005, bei der auch Marc Benioff, Gründer von *Salesforce.com*, teilnahm.

Im Jahr 2002 startete der Internet-Marktplatz *Amazon* seine *Amazon Web Services (AWS)*. Die damit zur Verfügung gestellten Dienste konnten Entwickler in eigene Anwendungen integrieren, ohne dass Endanwender direkt darauf zugriffen. Die Abrechnung erfolgte nach Nutzung. Aus diesen Anfängen entwickelte sich später unter anderem die heutige *Amazon Elastic Compute Cloud (EC2)* mit dem *Simple Storage Service (S3)*, die 2006 kurz nachdem Eric Schmidt sein Interview gab, präsentiert wurden [ama12]. Das Modell *Infrastructure as a Service (IaaS)* etablierte sich.

In den Jahren 2007 und 2008 folgte die logische Konsequenz der *SaaS*-Anbieter, die eigene Plattform für Dritte zu öffnen [Ban11]. *Salesforce.com* wurde mit *Force.com* nun auch zum *Platform as a Service*-Anbieter. *Google* zog mit der *Google App Engine* nach. Entwickler konnten nun eigene Anwendungen auf den großen, scheinbar endlos skalierbaren Lauf-

zeitumgebungen in der Wolke betreiben.

Bis zu dieser Zeit fiel es schwer die verschiedenen Modelle einzuordnen. Jeder Anbieter hatte seine eigene Auffassung von *Cloud Computing*. Auch Marktforscher wie *Gartner*, die *International Data Corporation (IDC)* oder *Forrester* veröffentlichten ihre – zumindest augenscheinlich herstellerunabhängigen aber dafür recht allgemeinen – Definitionen. Mit dem Erscheinen des Reports „Above the Clouds: A Berkeley View of Cloud Computing" wurde im Jahr 2009 an der *UC Berkeley* eine der ersten, mittlerweile viel zitierten, wissenschaftlichen Betrachtungen herausgegeben. Dabei wurden grundlegende Charakteristiken, wie das *Pay per Use*-Geschäftsmodell oder die elastische Skalierbarkeit konkret erläutert und die damit verbundenen Chancen aufgezeigt [AFG+09].

In der Branche haben sich mit dem großen Hype um *Cloud Computing* zahlreiche Gremien und Allianzen gebildet, in denen Hersteller, Dienstleister aber auch Behörden und Forschungsgesellschaften versuchen, gemeinsame Definitionen und Standards festzulegen [Fal11]. „Bisher konnte sich für den Begriff *Cloud Computing* keine Definition als allgemeingültig durchsetzen", so das Bundesamt für Sicherheit in der Informationstechnik (BSI) [bsib]. Im Report „A Break in the Clouds: Towards a Cloud Definition" wurde versucht die kursierenden Definitionen zusammenzutragen und Funktionalitäten von *Cloud* und *Grid Computing* zu vergleichen [VRMCL09]. Diese Arbeit legte wohl einen weiteren Grundstein.

In Fachkreisen stößt seit dem vergangenen Jahr eine Definition auf besonders breite Akzeptanz. Das *National Institute of Standards and Technology (NIST)*, eine Behörde die dem *U.S. Department of Commerce* angehört, veröffentlichte im September 2011 die „NIST Definition of Cloud Computing". Diese Definition wird auch vom europäischen Pendant, der *European Network and Information Security Agency (ENISA)* verwendet [bsib], die unter anderem auch für die EU-Kommission tätig ist. Diese arbeitet gerade an einer europaweiten *Cloud*-Strategie [eni11]. Das *NIST* definiert *Cloud Computing* wie folgt [MG]:

> *Cloud Computing* ist ein Modell, das einen universellen, unkomplizierten, netzwerkbasierten Zugriff auf einen gemeinsamen Pool von konfigurierbaren Rechenkapazitäten (z.B. Netzwerke, Server, Speicher, Anwendungen und Dienste) nach Bedarf ermöglicht, die in kürzester Zeit und mit minimalem Verwaltungsaufwand und Mitwirken des Anbieters bereitgestellt und zugänglich gemacht werden können.

Die Definition beinhaltet drei Service-Modelle und vier Bereitstellungsarten.

***Software as a Service* (*SaaS*).** Dem Nutzer wird die Möglichkeit gegeben, die Anwendung des Anbieters, die auf einer *Cloud*-Infrastruktur betrieben wird, zu nutzen. Auf die Anwendung kann über eine Vielzahl an Endgeräten zugegriffen werden, entweder über einen *Thin Client* wie einem Web-Browser (z.B. webbasiertes E-Mail), oder über eine Programm-Schnittstelle. Der Endverbraucher verwaltet oder überwacht weder die zugrunde liegende *Cloud*-Infrastruktur, einschließlich Netzwerken, Servern, Betriebssystemen oder Speicher noch individueller Anpassungsmöglichkeiten, mit der Ausnahme von eingeschränkten, benutzerspezifischen Konfigurationseinstellungen der Anwendung.

***Platform as a Service* (*PaaS*).** Dem Nutzer wird die Möglichkeit gegeben, eigene oder erworbene Anwendungen, die mittels Programmiersprachen, Bibliotheken, Diensten und Werkzeugen, die vom Anbieter unterstützt werden, erstellt wurden, auf der

Cloud-Infrastruktur bereitzustellen. Der Endverbraucher verwaltet oder überwacht nicht die zugrunde liegende *Cloud*-Infrastruktur einschließlich Netzwerken, Servern, Betriebssystemen oder Speicher, steuert jedoch die bereitgestellte Software und mögliche Konfigurationseinstellungen der Umgebung für den Anwendungsbetrieb.

***Infrastructure as a Service* (*IaaS*).** Dem Nutzer wird die Möglichkeit gegeben, Rechenleistung, Speicher, Netzwerke und andere grundlegende Rechen-Ressourcen zu verwalten, auf denen der Endverbraucher in der Lage ist beliebige Software, darunter Betriebssysteme und Anwendungen bereitzustellen. Der Endverbraucher verwaltet oder überwacht nicht die zugrunde liegende *Cloud*-Infrastruktur jedoch Betriebssysteme, Speicher und bereitgestellte Anwendungen und steuert möglicherweise ausgewählte Netzwerkkomponenten (z.B. *Host-Firewalls*).

Die Service-Modelle beschreiben demnach nicht nur den Einsatzzweck von *Cloud Computing*, sondern auch die Tiefe der Eigenverantwortung und damit verbunden natürlich auch die Möglichkeiten der Individualisierung. Aus Sicht eines Anbieters stellen sie die Wertschöpfungstiefe dar [bit]. Die Service-Modelle bauen also aufeinander auf und bilden die Schichten des *Cloud Computings*.

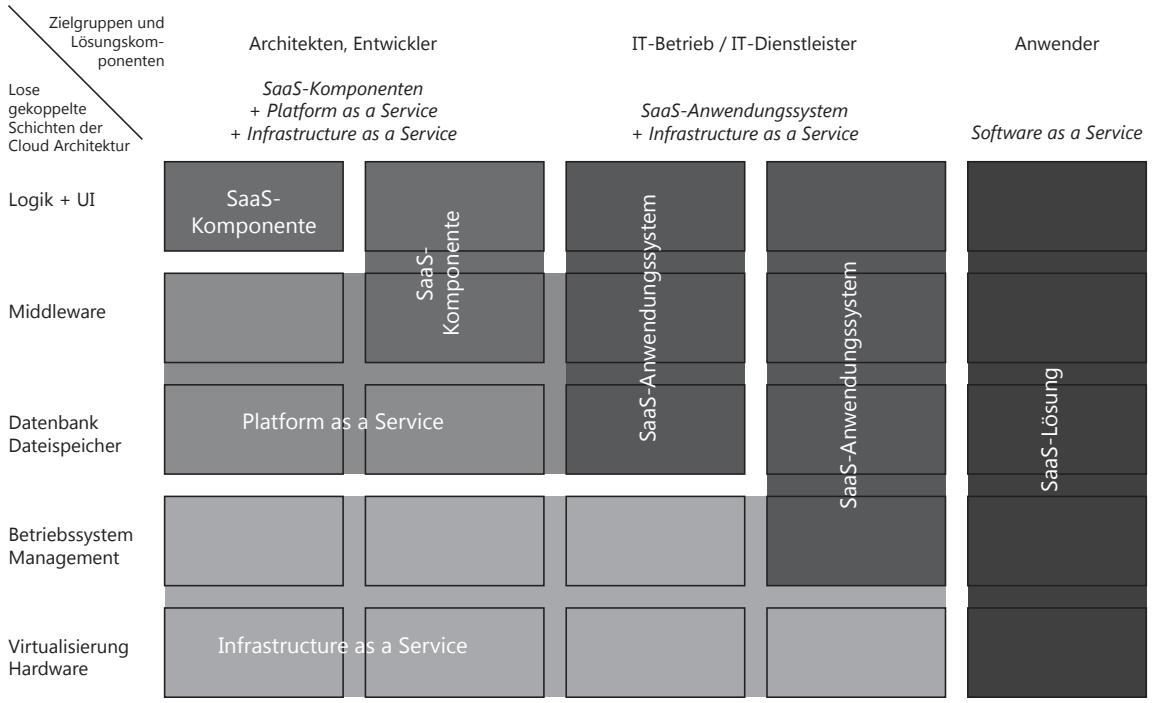


Abbildung 2.1.: Schichten einer Cloud-Architektur und Wertschöpfungstiefe von SaaS nach Zielgruppen in Anlehnung an BITKOM

Neben diesen drei fundamentalen Modellen, haben sich im Laufe der Zeit weitere Ableger gebildet wie zum Beispiel *(SecaaS)*, *(DaaS)* oder *(CaaS)* [MPR+]. So lässt sich zusammengefasst unter dem Begriff *(XaaS)* natürlich beliebigen Diensten der Stempel *Cloud Computing* aufdrücken obwohl oft nur das Geschäftsmodell (z.B. *Pay per Use*) auf einen *Mana-*

ged Service übertragen wird. Letztendlich können aber auch bei *Cloud*-basierten Diensten die meisten dieser Spezialisierungen auf die drei grundsätzlichen Modelle zurückgeführt werden, da es sich dabei natürlich wieder um Infrastruktur-Komponenten, Laufzeitumgebungen oder Software handelt.

Die zweite Dimension des *Cloud Computings* beschreibt die Art der Bereitstellung bezüglich ihrer Leistungsempfänger. So gibt es heute nicht die eine *Cloud* im Internet. Es existiert eine Vielzahl von Infrastrukturen, die oft sogar auf proprietären Technologien aufgebaut sind oder sich zumindest in den Basis-Technologien (z.B. der Virtualisierung) unterscheiden. Zusätzlich steigt die Zahl der Anbieter wobei die *Clouds* bis dato in den seltensten Fällen integriert sind. *Cloud Computing* wird also nicht nur von den *Global Playern* der IKT-Branche angeboten. Die Technologie bleibt auch nicht nur Großunternehmen und Konzernen vorbehalten. So ist – gerade in Deutschland – der Trend zu beobachten, dass Unternehmen aller Größen und Branchen eigene *Cloud*-Infrastrukturen aufbauen anstatt die Angebote der zahlreichen Anbieter auf dem Markt in Anspruch zu nehmen [kpm12].

Man unterscheidet daher prinzipiell in öffentliche und private *Clouds* sowie Mischformen. Das *NIST* spricht von vier grundlegenden Bereitstellungsarten:

Public Cloud. Die *Cloud*-Infrastruktur steht der breiten Öffentlichkeit offen. Sie ist im Besitz von einem Unternehmen, einer akademischen oder behördlichen Einrichtung oder einer Kombination und wird von dieser verwaltet und betrieben. Sie steht am Standort des *Cloud*-Anbieters.

Private Cloud. Die *Cloud*-Infrastruktur steht nur für eine einzelne Organisation, bestehend aus mehreren Nutzergruppen (z.B. Organisationseinheiten) zur exklusiven Nutzung bereit. Sie wird von dieser Organisation, von Dritten oder einer Kombination verwaltet und betrieben, ist in deren Besitz und kann vor Ort oder ausgelagert stehen.

Community Cloud. Die *Cloud*-Infrastruktur steht nur für einen bestimmten Verbund aus Nutzergruppen von Organisationen mit gemeinsamen Interessen (z.B. Ziele, Sicherheitsanforderungen, Richtlinien und *Compliance*-Aspekte) zur exklusiven Nutzung bereit. Sie wird von einer oder mehreren der Organisationen des Verbunds, von Dritten oder einer Kombination verwaltet und betrieben, ist in deren Besitz und kann vor Ort oder ausgelagert stehen.

Hybrid Cloud. Die *Cloud*-Infrastruktur besteht aus einer Zusammenlegung von zwei oder mehr getrennten *Cloud*-Infrastrukturen (*Private*, *Community* oder *Public Cloud*), die zwar autark bleiben aber mittels standardisierter oder proprietärer Technologie angebunden werden, so dass eine Portabilität von Daten und Anwendungen ermöglicht wird (z.B. Lasten-Ausgleich zwischen den *Clouds* bei Lastspitzen).

Auch hier findet man, wie auch bei den Service-Modellen, zahlreiche Spezialisierungen. Technologie-Berater wie die *IBM Global Technology Services* unterscheiden diese beispielsweise auch hinsichtlich ihrer Entwicklungsstufe bei der Einführung von *Cloud Computing* oder der Anwendungsbreite in Unternehmen (z.B. *Exploratory Cloud*, *Departmental Cloud*,

uvm.) [ibm]. Auf die diversen Subtypen wird an dieser Stelle jedoch nicht genauer eingegangen.

2.2. Kriterien für Software as a Service

Neben der konzeptionellen Einordnung beschreiben die diversen Definitionen auch die prägenden Charakteristiken der vorgestellten Modelle. Was die Terminologie, die grundlegenden Bereitstellungsarten oder auch das Schichtenmodell angeht, scheint man sich mittlerweile einig geworden zu sein. Und so liegen die größten Unterschiede zwischen den Definitionen meist nur noch in deren Beschreibung. Einige charakterisieren *Cloud Computing* sehr detailliert andere dagegen nur oberflächlich. So scheint es als wäre die größte Schwierigkeit eine gute Definition zu geben, den richtigen Grad an Präzision zu finden. Laut *Gartner* sollte auch die *NIST* Definition nicht als das Maß der Dinge sondern als weitere *Working Definition* gesehen werden. Der Marktforscher warnt vor Unklarheiten bei ihrer Anwendung [DM09]. Letztendlich wurden viele der frühen Definitionen in ihrer ersten Fassung mittlerweile überarbeitet. So waren die Service-Modelle nicht immer ausreichend konkretisiert und damit ließ man den Anbietern bei Erfüllung von Kriterien viel Interpretationsfreiheit. Als Folge wurde aus so mancher Web-Anwendung kurzer Hand *SaaS* und einfache *Hosting*-Dienste wurden als *IaaS* deklariert.

Da die Kriterien für *Cloud Computing* und *SaaS* bei der Methodenentwicklung eine tragende Rolle spielen wurden neben der des *NIST* drei weitere Definitionen herangezogen. Darunter die von zwei internationalen Analysten und die des deutschen Bundesverband Informationswirtschaft, Telekommunikation und neue Medien (BITKOM).

Die folgende Tabelle 2.1 zeigt eine Gegenüberstellung der identifizierten Kriterien. Dabei wurden die übergeordneten Charakteristiken von *Cloud Computing*, die demnach für alle Service-Modelle und Bereitstellungsarten gleichermaßen gelten, von den *SaaS*-spezifischen getrennt betrachtet. Um mögliche Überdeckungen identifizieren zu können wurden gleichgesinnte Ausdrücke in einer Zeile aufgetragen.

Dadurch lässt sich bereits auf den ersten Blick erkennen wo die *NIST* Definition ihre vermeintlichen Schwächen hat. So werden die Ausprägungen der Service-Modelle nur elementar beschrieben.

Bei den Kriterien zu *Cloud Computing* kann man in fünf Punkten eine nahezu vollständige Übereinstimmung feststellen:

- Eigenständige Bereitstellung nach Bedarf und in Echtzeit
- netzwerkbasierter Zugriff über Internet-Technologien
- Messbarkeit der Nutzung zur Kontierung des Verbrauchs
- Elastische Ressourcen-Zuweisung
- Gemeinsame Nutzung von IT-Ressourcen

Diese werden vom *NIST* auch als essentiell bezeichnet [MG]. Die Definitionen von *Gartner* und *IDC* fordern zusätzlich für alle Service-Modelle standardisierte Application Program-

	NIST	IDC	Gartner	BITKOM
Cloud Computing	On Demand Self-Service	Self-Service	„Ready to use“-service based on service levels	Bereitstellung in Echtzeit als Self-Service
	Broad Network Access	Access via Internet Protokoll (IP)	Uses Internet Technologies	Zugriff über Internet-Technologien
	Measured Service	Used-based Pricing	Metered by Use	Abrechnung nach Nutzung
	Rapid Elasticity	Elastic Scaling	Scalable and Elastic	Flexible Skalierbarkeit
	Ressource Pooling	Shared Standard Service	Shared pools of ressources	Gemeinsame Nutzung von IT-Ressourcen
		Published Service Interface/API	Well-defined service interfaces with automated response	
Software as a Service	Various Client Devices / User Interface (UI)s	Standard UI Technologies		
		Managed by an external Service Provider	Owned, delivered and managed remotely	Anwendung und Infrastruktur befinden sich beim Dienstleister
		Ongoing support and maintenance, daily technical operation	Includes maintenance and upgrade services	
		One-to-many model	Single set of common code and data definitions	Gemeinsame Infrastruktur für alle Kunden (1:N)
	Limited user-specific configuration settings			Eingeschränkte Anpassungs- und Integrationsmöglichkeiten
			Extendable data model without altering the source code	Leichte Erweiterbarkeit
	Runs on cloud infrastrcture	Solution-packaged		Gesamtbündel aus Infrastruktur und Applikation)

Tabelle 2.1.: Gegenüberstellung gängiger Definitionen für Cloud Computing und SaaS

ming Interface (API)s zur Verwaltung und Steuerung des Cloud-Angebots [gar09][idca].

Bei den Kriterien für *Software as a Service* lässt sich eine höhere Diversifikation feststellen. Die *NIST* Definition fordert lediglich die Möglichkeit über verschiedene Endgeräte und Schnittstellen auf das *SaaS*-Angebot zugreifen sowie eingeschränkte Konfigurationsmöglichkeiten vornehmen zu können. Zudem soll die Anwendung selbst natürlich auf einer *Cloud*-Infrastruktur, die die essentiellen Kriterien erfüllt, betrieben werden. *IDC*, *Gartner* und auch der *BITKOM* führen jedoch zusätzliche Aspekte mit auf [idcb][gara][MPR$^+$]. So definieren diese *SaaS* als eine Anwendung, die in jedem Fall von einem externen Dienstleister bereitgestellt werden muss wobei sich die Infrastruktur ebenfalls außer Haus befinden muss. Betreibt ein Unternehmen eine interne Anwendung in einer *Private Cloud*, so stellt es nach dieser Auffassung kein *SaaS* dar. Auch müssen Wartung, Pflege und Betrieb, ohne Mitwirkungspflichten des Anwenders, durch die Anbieter ausgeführt werden. Beide Punkte betreffen weniger die Implementierung der Anwendung als das Geschäftsmodell und den Betrieb.

Aus Sicht der Entwicklung eines *SaaS*-basierten Angebots werden jedoch zwei weitere interessante Kriterien genannt, die durchaus komplexe Anforderungen an eine Software-Architektur darstellen können. *Software as a Service* bedeutet demnach auch, die Anwendung auf einer gemeinsamen Infrastruktur in einer 1:N-Beziehung an Kunden auszuliefern. Das heißt, dass alle Kunden vom selben Quellcode und den selben Datenstrukturen bedient werden. Der *BITKOM* schärft diesen Aspekt mit dem Zusatz, dass neue Versionen der Anwendung nur einmalig und zentral eingespielt werden, so dass Kunden umgehend davon profitieren können [MPR$^+$]. Daraus lässt sich schließen, dass gemäß dieser Auffassung, alle Anwender in einer einzigen Laufzeitumgebung operieren.

Ein weiteres Kriterium stellt die Erweiterbarkeit über die grundlegende, kundenspezifische Anpassung hinaus dar. Datenmodelle müssen nach *Gartner* auf einfache Weise individualisiert werden können, ohne dabei den Quellcode zu verändern.

Die beiden letztgenannten Kriterien können als Komplexitätstreiber gesehen werden. Folglich steigt die Komplexität der Software-Architektur mit dem Grad der gemeinsamen Nutzung von IT-Ressourcen und dem Grad der Erweiterbarkeit durch den Anwender.

Zuletzt nennen die Definitionsgeber noch eine weitere Charakteristik: eine *SaaS*-Lösung bündelt stets die gewünschte Anwendung mit den zum Betrieb benötigten Infrastruktur-Diensten (vgl. 2.1). *SaaS*-Anwender sind gemäß dieser Definition auch an die Plattform (*PaaS*) und Infrastruktur (*IaaS*) des Anbieters gebunden. Erwirbt man demnach eine Anwendung und betreibt diese auf einer *PaaS* von Dritten oder der eigenen, so handelt es sich dennoch nicht um *SaaS*, auch wenn die Anwendung selbst die Kriterien erfüllt. Entwickler von *SaaS*-Lösungen können hingegen schon auf bestehende Angebote von *PaaS* und *IaaS* zurückgreifen oder proprietäre *Cloud*-Infrastrukturen aufbauen.

Zusammengefasst lassen sich unter Berücksichtigung aller vier Definitionen folgende *SaaS*-spezifischen Kriterien festhalten:

- Zugriff über verschiedene Endgeräte und Benutzerschnittstellen
- Betrieb der Anwendung durch einen externen Dienstleister außer Haus
- Wartung, Pflege und Support sind im Angebot enthalten

- Bereitstellung der Anwendung im 1:N-Modell
- Anpassungs- und Integrationsmöglichkeiten
- Einfache Erweiterung durch den Anwender
- Bündelung von Anwendung, Plattform und Infrastruktur in einem Angebot

Eine Anwendung die, alle *SaaS*-spezifischen Kriterien wie auch die essentiellen Kriterien des *Cloud Computings* erfüllt, wird fortan als vollwertige *Software as a Service*-Lösung bezeichnet.

2.3. Klassifikation von Anwendungssystemen

Diese Arbeit beschäftigt sich mit Anwendungssystemen, die als *Software as a Service* bereit gestellt werden sollen. Um die Thematik und die Anwendung der Methode besser verstehen zu können, werden im folgenden Abschnitt die Begriffe eingeordnet sowie die grundlegenden Klassen von Anwendungssystemen vorgestellt.

Bei einem Anwendungssystem handelt es sich um ein Teilsystem des betrieblichen Objektsystems nach Ferstl/Sinz. Darin wird grundsätzlich zwischen automatisiert und nichtautomatisiert sowie zwischen Informations- und Basissystemen (den Systemen, die keine Informationen verarbeiten) unterschieden. Als Anwendungssystem wird der automatisierte Teil eines Informationssystems bezeichnet [FS95].

Nach Hansen/Neumann unterstützt ein betriebliches Informationssystem die Leistungsprozesse und Austauschbeziehungen innerhalb des Betriebs sowie zwischen dem Betrieb und seiner Umwelt [HN05]. Bei einem betrieblichen Anwendungssystem handelt es sich demnach um die Gesamtheit aller Anwendungen, die für ein konkretes betriebliches Anwendungsgebiet eingesetzt werden sowie um die dazugehörigen Daten.

Demnach können Anwendungssysteme hinsichtlich ihrer betrieblichen Funktion entlang der horizontalen Wertschöpfungskette klassifiziert werden. Weitere Möglichkeiten sind die Klassifikation nach Branche, nach Verwendungszweck entlang der vertikalen Lenkungsebene oder auch eine Klassifikation nach dem Grad der Spezialisierung, Leistungsumfang oder der Standardisierung [Amb99].

Die nachfolgende Abbildung 2.2 zeigt die Klassifizierung von Anwendungssystemen nach ihrem Verwendungszweck. Anwendungen für Personal-, Finanz- und Rechnungswesen zählen zu den branchenneutralen Anwendungssystemen. Beispiele für zwischenbetriebliche Anwendungssysteme sind zum Beispiel *E-Commerce* Portale (elektronische Marktplätze) oder so genannte *EDI*-Systeme zum elektronischen Datenaustausch mit der betrieblichen Umwelt. Aber auch Bürosysteme zur Kommunikation und Kooperationsunterstützung oder ein Dokumententen-Management-System stellen betriebliche Anwendungssysteme dar [SH04].

Der Begriff Anwendungssystem ist also auf zahlreiche Anwendungen der IKT anzuwenden und somit sehr breit gefächert. Bislang hat sich *SaaS* vorwiegend bei System zur Administration und Disposition sowie Bürosystemen durchgesetzt.

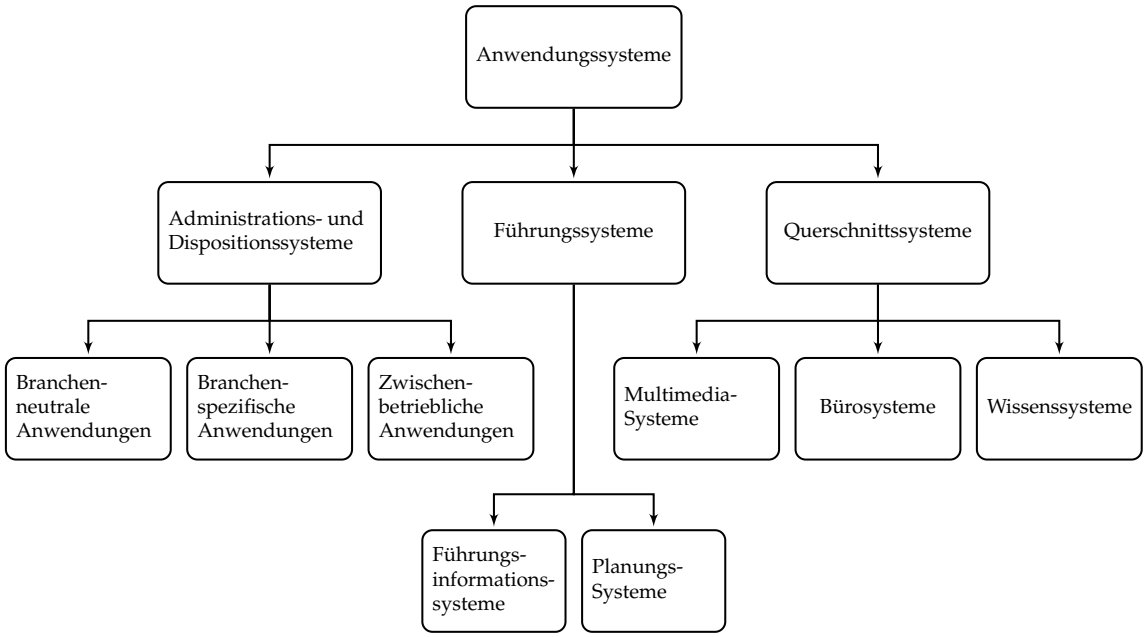


Abbildung 2.2.: Klassifikation von Anwendungssystemen nach Verwendungszweck in Anlehnung an Stahlknecht/Hasenkamp [SH04]

Klassische Beispiele sind *CRM-*, *HR-* oder *ERP*-Systeme sowie Anwendungen zur Kollaboration wie Internet-Telefonie, Webkonferenzen, E-Mail Dienste oder Portale (z.B. soziale Netzwerke) [MPR$^+$].

Die in dieser Arbeit entwickelte Methode soll prinzipiell auch auf andere Klassen anwendbar sein.

3. Vorgehensweise

Die Entwicklung der Methode erfolgte in enger Kooperation mit der *msg systems AG*, da im Unternehmen bereits ein konkreter Bedarf und damit auch mögliche Anwendungsszenarien gegeben waren. Zudem konnte auf das Expertenwissen der Mitarbeiter zurückgegriffen werden. Die explorative Vorgehensweise erfolgte in Anlehnung an das Einführungskonzept eines Wissensmanagements in Unternehmen nach Allweyer/Jost [AJ99].

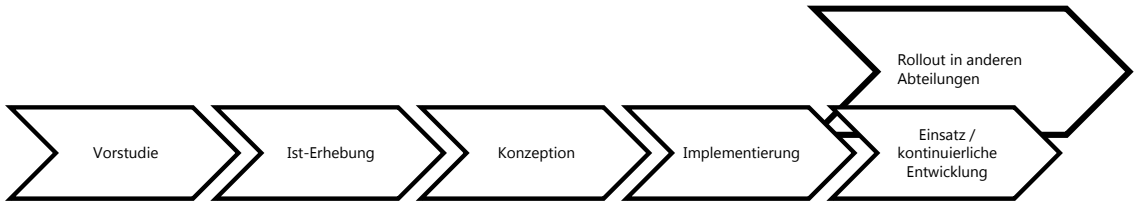


Abbildung 3.1.: Prozess zur Vorgehensweise nach Allweyer/Jost; Darstellung nach Prof. Gronau, Universität Potsdam

In der Vorstudie wurde zunächst die Aufgabenstellung geschärft sowie die Ziele, Rahmen und Ergebnistypen des Projekts formuliert. Anschließend wurde eine umfangreiche Recherche – vorwiegend im Internet – durchgeführt, bei der zunächst geprüft wurde, ob bereits Lösungen für diese oder ähnliche Problemstellungen bestehen. Zwar existieren einige Leitfäden oder Checklisten zum Aufbau von *SaaS*-Lösungen für Hersteller allerdings betrachten diese die Materie oft nur sehr oberflächlich und weniger technisch [LMR+]. Neben Leitfäden und Checkliste konnten diverse *Tools* zur Bewertung der *Cloud Readiness* von Unternehmen ausgemacht werden, die jedoch nicht die Transformation einer Anwendung, sondern die Fähigkeit und Bereitschaft eines Unternehmens *Cloud Computing* zu nutzen, bewerten [vmw].

Eine Basis, die für eine Bewertung der Transformierbarkeit – auch mit Blick auf die konkrete Implementierung, herangezogen werden kann, konnte nicht gefunden werden. Dennoch waren die vorhandenen Werkzeuge eine gute Grundlage zur Inspiration und um das Facettenreichtum von *Cloud Computing* zu überschauen.

Schließlich folgte eine Bestandsaufnahme der von *msg* im Vorfeld gesammelten Ideen und Beiträge zur Thematik. Dort hatte man sich in der Vergangenheit bereits intensiv zum Thema Sicherheit in der *Cloud* auseinandergesetzt und so wurden relevante Kriterien tabellarisch zusammengetragen.

Mit dem Wissen, dass es sich bei *Cloud Computing* um ein ganzheitliches Konzept und nicht nur eine Informationstechnologie handelt, wurden mögliche Disziplinen identifiziert, die zur Einführung einer *SaaS*-Lösung gefordert sind. So wurde der Forschungsschwerpunkt nicht von vornherein eingegrenzt und es konnte ein geeigneter Einstiegspunkt in die Materie gefunden werden.

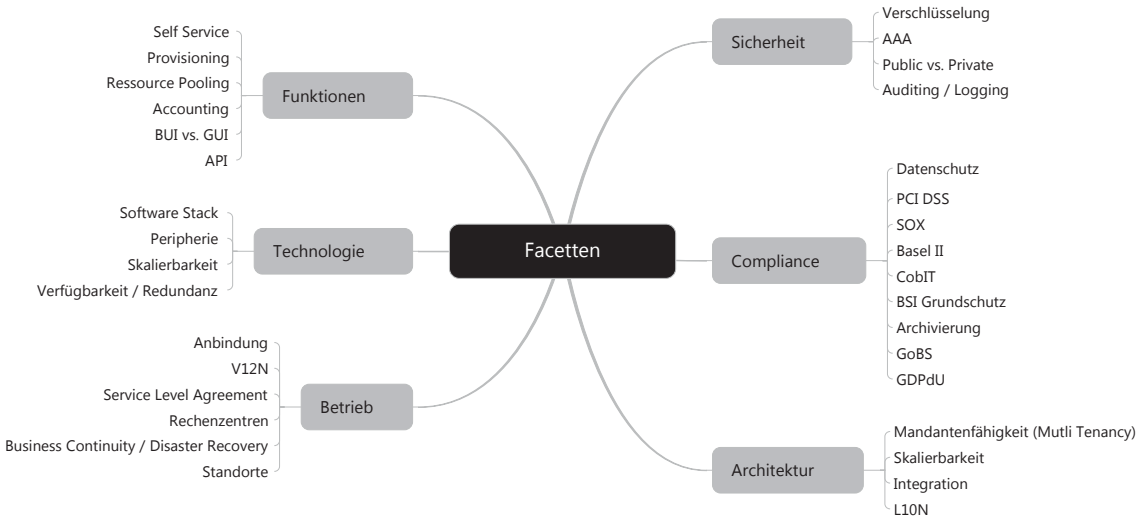


Abbildung 3.2.: Studie zum Facettenreichtum des Cloud Computings

Schließlich wurde der Fokus dieser Arbeit auf die Software-Architektur gesetzt, da beim Kooperationspartner *msg* hierfür viele Experten eingebunden werden konnten und gleichzeitig das Forschungsschwerpunkt des Lehrstuhls für *Software Engineering betrieblicher Informationssysteme* gewahrt blieb. Sicherlich sind auch andere Disziplinen wie Sicherheit oder *IT-Compliance* ein spannendes Gebiet des *Cloud Computings*. Dennoch erfordern Sie detailliertes Fachwissen, zum Beispiel bei juristische Belangen.

Die Ergebnisse zur Software-Architektur legen den Grundstein für die Weiterentwicklung der Methode innerhalb der *msg* – nicht zuletzt auch in anderen Disziplinen.

Der erste Schritt zur Konzeption war eine Anforderungsermittlung. So wurden anhand eines fiktiven Anwendungsbeispiels verschiedene Szenarien zur späteren Anwendung der Methode ausgearbeitet. Daran konnte auch abgeleitet werden, welchen Output die Methode liefern muss, um als Hilfsmittel in der Beratung herangezogen werden zu können. Grundlegende Überlegungen zur Methodik wurden angestellt.

Mit dem Ziel, eine möglichst generische Methodik zu entwickeln, die nur projektspezifische Aspekte zur Transformation einer *SaaS*-Lösung behandelt, wurden die bisherigen Überlegungen zur Methodik zunächst mit einer einzelnen Anforderung durch exerziert. Die daraus abgeleitete Filterstruktur konnte an einem zweiten Beispiel validiert werden.

Aus den beiden Beispielen wurde schließlich eine Datenstruktur entwickelt und diese in einem Prototyp implementiert, der die gewünschten Ergebnisse als Grundlage zur Entscheidungsfindung liefert. Die Datenstruktur wurde mit der Durchführung von Workshops in der Querschnittsabteilung *Applied Technology Research* weiter evaluiert und ausgebaut.

Im letzten Schritt wurden die Ergebnisse formalisiert. Somit ist die entwickelte Methode nicht nur für die spezielle Anwendung innerhalb der *msg* geeignet sondern kann auch auf andere Bedarfsfälle und Szenarien zugeschnitten werden.

4. Konzeption und Formalisierung der Methode

Dieser Teil der Arbeit beschäftigt sich mit der Konzeption der Methode. Dabei werden zunächst Anforderungen ermittelt und schließlich die Vorgehensweise zur Lösung der Problemstellung aufgezeigt. Im letzten Abschnitt wird die entwickelte Methode formalisiert und in ein abstraktes Modell überführt.

4.1. Anforderungsermittlung

Zu Beginn wurde gemeinsam mit den Auftraggebern der konkrete Bedarf ermittelt, indem verschiedene Szenarien zur Anwendung der Methode entwickelt wurden. Die Basis dazu lieferte eine vergangene Anfrage einer Fachabteilung der *msg systems AG*, die für ein laufendes Projekt entsprechende Unterstützung angefordert hatte. Die wesentlichen Szenarien sind dabei die Unterstützung eines Kunden – intern oder extern – durch einen Berater und der interne Wissenstransfer durch eine Fachkraft. Das heißt die Methode und das ihr zugrunde liegende Fachwissen sollen zur Prozessunterstützung und als Informationsquelle gleichermaßen angewendet werden können. In beiden Fällen gilt es, die für eine Transformation relevanten Aspekte zu identifizieren.

Beispielszenario: Ein Automobilhersteller stellt seinen Vertragswerkstätten einen elektronischen Ersatzteilkatalog zur *On-Premise* Installation auf einem Server zur Verfügung. Die Endanwender greifen über das lokale Netzwerk und einem Web-Browser auf das Anwendungssystem zu. Updates werden per *Download* oder Datenträger durch einen Dienstleister eingespielt. Der Hersteller plant den Ersatzteilkatalog zukünftig auch als *Software as a Service*-Lösung anzubieten, um Händlern den Betrieb zu erleichtern und um Kosten für den *Support* einzusparen. Zudem führt die stetig anwachsende Produktpalette zu kürzeren Update-Intervallen. Die Lösung wurde als Individualsoftware von der internen IT-Abteilung entwickelt und somit verfügt der Hersteller über detaillierte Kenntnisse über die Anwendung und ist auch im Besitz des Quellcodes.

Da jedoch keine Kompetenzen zur Entwicklung einer *Cloud*-basierten Anwendung vorhanden sind wendet sich die Fachabteilung an ein Beratungsunternehmen, wie der *msg*. Bereits in einer frühen Phase der Projektentwicklung soll die grundsätzliche Machbarkeit einer Transformation untersucht werden.

Wird die Methode zur Beratung eines Kunden eingesetzt, so ist dieser in der Lage, sein Vorhaben grundlegend zu spezifizieren. Beispielsweise kennt er die Architektur der Anwendung und weiß welche Technologien eingesetzt werden. Die Rolle des Kunden ist

deshalb idealerweise durch Fachkräfte mit detaillierten Kenntnissen über die vorhandene *On-Premise* Lösung besetzt (z.B. Software-Architekt, Entwickler, Datenschutzbeauftragter). Darüber hinaus sind die projektspezifischen Ausprägungen der zu erfüllenden *SaaS*-Kriterien bekannt, so zum Beispiel anhand welcher Größen eine Abrechnung mit den Nutzern erfolgt oder Wachstumsprognosen und Nutzerverhalten.

Der Berater (z.B. Vertriebsmitarbeiter, Projektleiter) kennt die zu transferierende Anwendung bezüglich ihrer Implementierung nicht zwangsläufig im Detail. Er benötigt die Methode hauptsächlich zur Komplexitätsreduzierung und zur Strukturierung wie auch zur Dokumentation der Angaben des Kunden. Die eigentliche Bewertung erfolgt durch die Akteure möglichst nur hinsichtlich der relevanten Kriterien und ihren Charakteristiken sowie für die geforderten Disziplinen.

Im zweiten Szenario wird die Methode unternehmensintern angewendet. Mitarbeiter verwenden sie zum Wissenstransfer. Dabei muss zwischen der Beschaffung und der Weitergabe von Wissen unterschieden werden. Fachkräfte wenden die Methode an, um konkrete Anforderungen zu spezifizieren, etwa im Rahmen des *Requirement Engineerings*, oder um Projekte zu verifizieren oder zu vergleichen. Ein weiterer Anwendungsfall ist die Rückkopplung von angewandtem Wissen und Erfahrungen aus abgeschlossenen Projekten. Lösungsansätze oder weitere Anforderungen werden in die Methode eingespeist.

Anhand dieser beiden Szenarien konnten folgende Anforderungen an die zu entwickelnde Methode identifiziert werden:

- Hilfsmittel zur Bewertung der Transformierbarkeit
- Leitfaden zur Spezifikation des Endprodukts
- Identifikation relevanter Aspekte durch Filterfunktionalität
- Ableitung von Handlungsempfehlungen
- Nachvollziehbarkeit durch Strukturierung und Dokumentation der Angaben
- Reproduzierbarkeit und Verlässlichkeit des Bewertungsergebnis

Sie dienen sowohl der Prozessunterstützung als auch dem Wissensmanagement.

4.2. Methodische Überlegungen

Um eine Bewertung der Transformierbarkeit, das heißt der Machbarkeit der Portierung einer bestehenden Implementierung auf eine *Cloud*-Infrastruktur, unter Berücksichtigung der in Abschnitt 2.2 zu erfüllenden *SaaS*-Kriterien durchführen zu können, müssen Berater und Kunde wissen, welche Anpassungen unter den gegeben Voraussetzungen durchführbar oder überhaupt nötig sind. Dazu müssen die *SaaS Essentials* in konkrete Anforderungen zergliedert werden. Beispielsweise eignet sich möglicherweise die verwendete Datenbank nicht zur Massenskalierung oder das verwendete *Framework* erlaubt keine Zustandslosigkeit. Manche Anforderungen sind vielleicht bereits in der *On-Premise* Lösung implementiert, wie zum Beispiel ein webbasierter Zugriff oder Mehrsprachigkeit in der Benutzeroberfläche. Für andere Anforderungen gilt es unter den gegeben Voraussetzungen vielleicht eine individuelle Lösung zu entwickeln. Dabei gibt es Kunden die einen

Public Cloud-Service anbieten wollen, der kritische Kundendaten verarbeiten muss und welche, die eine interne Applikation in der *Private Cloud* betreiben wollen und deshalb gewisse Anforderungen (z.B. gesetzliche Auflagen) nicht erfüllen müssen.

Mit der Zielsetzung eine Methode zu entwickeln, die generisch anwendbar ist und beliebige Szenarien unterstützt, wurden folgende Prämissen aufgestellt:

1. Um die Machbarkeit der Transformation vor ihrer Durchführung bewerten zu können müssen sowohl der ursprüngliche Zustand als auch der Zielzustand bekannt sein.

2. Das Filtern von relevanten Aspekten erfordert eine Strukturierung von Anforderungen und Lösungsansätzen sowie die Abbildung von Abhängigkeiten.

3. Konkrete Handlungsempfehlungen können nur ausgesprochen werden wenn auch konkrete Lösungsansätze zum Erreichen des Zielzustands existieren.

4. Die Bewertung muss hinsichtlich der Realisierbarkeit der Handlungsempfehlungen erfolgen.

Daraus konnte schließlich eine Methode hergeleitet werden, die sich wie folgt beschreiben lässt. Zur Bewertung müssen konkrete Lösungsansätze identifiziert werden. Diese Identifikation erfolgt durch die Spezifikation von Anforderungen an das Endprodukt (also der *SaaS*-Lösung) denen Lösungsansätze zugeordnet sind. Ein Projekt stellt demnach eine Teilmenge dar (vgl. Abb. 4.1). Sowohl Anforderungen als auch Lösungsansätze können Abhängigkeiten unterliegen wie zum Beispiel Bedingung oder Ausschluss. Als Resultat erhält man eine Teilmenge von Lösungsansätzen aus denen sich Handlungsanweisungen ableiten lassen. Ist für eine bestimmte Anforderung kein Lösungsansatz zugeordnet, so muss dieser außerordentlich identifiziert werden. Beide Mengen sind demnach beliebig erweiterbar. Die Zuordnung von Anforderungen auf Lösungsansätze stellt eine beliebige Abbildung dar. So kann ein einzelner Lösungsansatz mehrere Anforderungen erfüllen, oder eine Anforderung mehrere Lösungsansätze erfordern.

Die Methode ermöglicht, durch die Herleitung von Handlungsanweisungen eine Bewertung durch den Anwender, das heißt sie stellt eine Bewertungsmöglichkeit zur Verfügung ohne selbst eine Wertung durchzuführen. Folglich benötigen Anwender sowohl ausreichende Kenntnis über den Ist- als auch den Soll-Zustand des zu transformierenden Anwendungssystems. Voraussetzung dafür ist eine Datenbasis, die Anforderungen und Lösungsansätze strukturiert. Die Methode besteht also aus drei Komponenten:

- Datenstruktur für Lösungsansätze

- Filterstruktur zur Identifikation relevanter Aspekte

- Bewertungsverfahren

Diese Komponenten werden in den folgenden Abschnitten genauer erläutert.

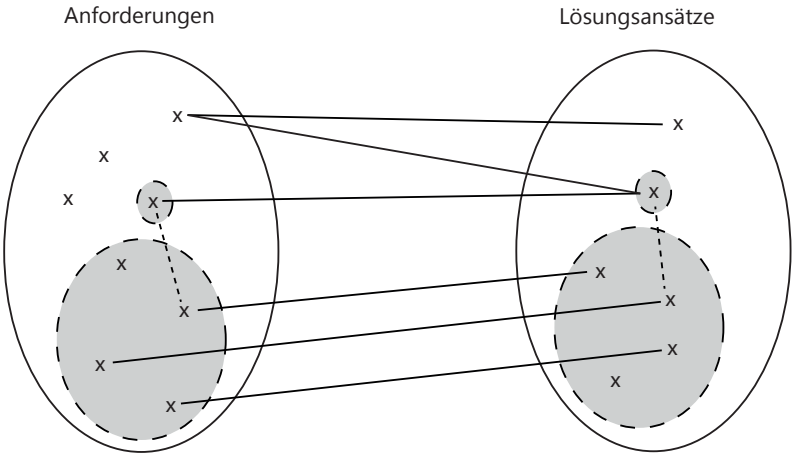


Abbildung 4.1.: Filterung von Lösungsansätzen durch Spezifikation der Anforderungen

4.3. Matrix zur Katalogisierung von Lösungsansätzen

Die zentrale Komponente der Methode ist ein Katalog von Lösungsansätzen in Form einer mehrdimensionalen Matrix. In der Matrix wird das gesammelte Wissen zur Implementierung von *SaaS*-Lösungen in Raster eingeordnet. Auf diese Weise lassen sich die Lösungsansätze, wie zum Beispiel zur Mandantenfähigkeit, über verschiedene Aspekte hinweg klassifizieren, so dass eine granulare Filterung ermöglicht wird. Für das genannte Beispiel existieren verschiedene Herangehensweisen, die in den Komponenten einer Architektur, über spezifische Konzepte oder Praktiken umgesetzt werden können. Auch die Anforderung selbst kann unterschiedliche Ausprägungsformen haben.

Bei der Entwicklung der Methode wurden drei Kerndimensionen festgelegt. Die erste Dimension dient der Zuordnung der Lösungsansätze zu Anforderungen. Die zweite Dimension charakterisiert die Lösungsansätze, um projektspezifische Ausprägungen der Anforderungen berücksichtigen zu können. Die Dritte ermöglicht eine Einordnung der Lösungsansätze bezüglich ihrer Ordnung im Gesamtsystem, so zum Beispiel nach der Komponente zur Implementierung in einer Systemarchitektur. Eine Erweiterung der Matrix um zusätzliche Dimensionen ist möglich. Wie im vorherigen Kapitel 4.2 beschrieben ist eine mehrfache Zuordnung nicht auszuschließen. In der grafischen Darstellung einer Matrix führt dies zu redundanten Einträgen. Software-technisch ist stellt eine solche 1:N-Beziehung bekanntlich kein Problem dar.

Neben der Einordnung kann der Katalog zusätzliche Informationen zu den Lösungsansätzen in Form von Attributen enthalten. So können beispielsweise Beschreibungen hinzugefügt oder Typen bestimmt werden. Im Gegensatz zu den Dimensionen der Matrix erlauben Attribute keine strukturierte Filterung wie sie im nächsten Abschnitt eingeführt wird.

Da Matrizen beim Wissensmanagement häufig als Instrument zur Kategorisierung von Wissen herangezogen werden ist es durchaus denkbar den Katalog auch für andere Zwecke zu nutzen oder vorhandene Datenquellen, zum Beispiel Anwendungen zur Anforderungsanalyse oder Aufwandsschätzung, an die Methode anzubinden.

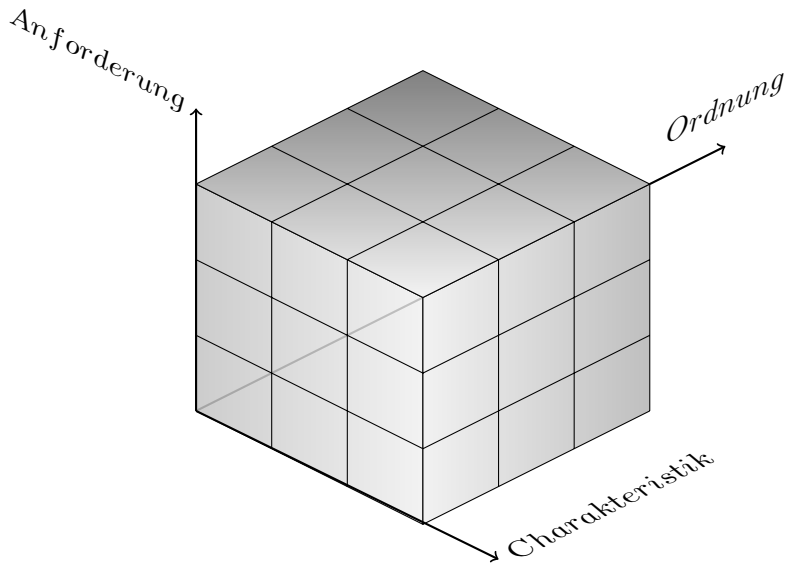


Abbildung 4.2.: Dreidimensionale Matrix zur Strukturierung des Katalogs

4.4. Bäume zur strukturierten Filterung

Um nur noch relevante Aspekte betrachten und bewerten zu müssen, bedarf es einer Filterstruktur, die aus der Matrix irrelevante Spalten (oder Zeilen) einer Dimension herausfiltert. Diese Struktur dient gleichzeitig auch als Leitfaden zur Anforderungsspezifizierung.

Die grundlegende Idee dazu basiert auf gerichteten Bäumen (vgl. Graphentheorie). Anforderungen, Charakteristiken, Ordnungen oder weitere Dimensionen werden also hierarchisch angeordnet. Dazu muss für jede Dimension eine individuelle Struktur entwickelt werden, die die Artefakte einer Dimension Ebene für Ebene gruppiert. Die Herleitung einer solchen Baumstruktur kann durch Generalisierung oder durch Spezialisierung erfolgen. In beiden Fällen entsteht eine gewurzelte Baumstruktur, die eine Navigation zur Spezifikation der Anforderungen und das Setzen von Filtern über die Dimensionen hinweg ermöglicht.

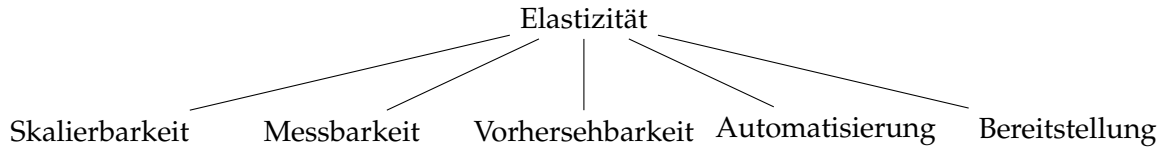


Abbildung 4.3.: Beispiel einer Baumstruktur zur Spezialisierung einer Anforderung

Bei der Anforderungsspezifikation durchläuft der Anwender den Baum beginnend mit der Wurzel. Dabei können einzelne Knoten gefiltert werden, so dass deren Folgeknoten in der nächsten Ebene nicht mehr berücksichtigt werden. Auf diese Weise können ganze Pfade exkludiert werden. Die Anzahl der Blätter des Baumes entspricht der Anzahl der

Spalten (oder Zeilen) der jeweiligen Dimension der Matrix.

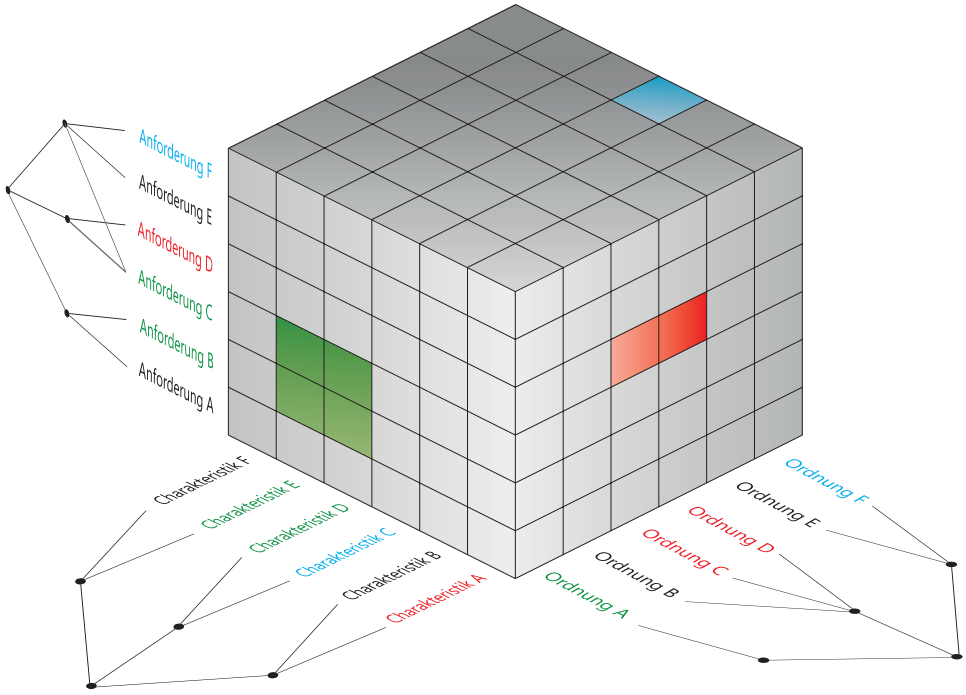


Abbildung 4.4.: Grafische Darstellung eines dreidimensionalen Katalogs mit seinen Filterstrukturen

Erweitert man eine solche Baumstruktur um Abhängigkeiten oder Mehrfachzuordnungen, so entartet der Baum in eine Polyhierarchie und stellt nur mehr einen gerichteten Graphen dar. Das Setzen der Filter kann weiterhin analog zur Breiten- oder, falls erforderlich, auch zur Tiefensuche erfolgen.

4.5. Ableitung von Handlungsanweisungen

Die oben beschriebene Matrix aus Lösungsansätzen kann nun mit der Filterstruktur auf projektrelevante Aspekte reduziert werden. Um jedoch eine Bewertung vornehmen zu können müssen aus diesen Lösungsansätzen konkrete Handlungsanweisungen abgeleitet werden. Dazu werden die beschreibenden Attribute sowie die Informationen aus der Einordnung in die Matrix herangezogen.

Ein zunächst allgemeiner Lösungsansatz wird also unter Berücksichtigung der spezifizierten Charakteristik in den jeweiligen Dimensionen in einen spezifischen Kontext gesetzt. Je mehr Dimensionen der Katalog umfasst und umso mehr beschreibende Attribute enthalten sind desto präziser können die Handlungsanweisungen formuliert werden.

Eine solche Handlungsanweisung lässt sich in einem dreidimensionalem Katalog wie folgt zusammensetzen:

> „Erfülle die Anforderung «*Anforderung*» mit dem/der «*Charakteristik*» durch Implementierung von «*Lösungsansatz*» in dem/der «*Ordnung*».“

Beispiel:

> „Erfülle die Anforderung *Skalierbarkeit* mit dem *Lastprofil An/Aus* durch Implementierung von *Clustering* in dem *Container Anwendungsserver*.“

Die daraus gewonnene Checkliste dient als Basis zur Bewertung der Transformierbarkeit durch die Anwender.

4.6. Bewertungsverfahren

Das wesentliche Ziel der Methode ist eine Bewertung durchzuführen, die als Entscheidungsgrundlage für ein Transformationsprojekt dient. Die Methode wurde von Beginn an mit dem Ziel entwickelt möglichst flexibel und erweiterbar zu sein. Daher wird keine selbständige Bewertung durchgeführt sondern der *Input* für ein integriertes Bewertungsverfahren geliefert. Somit lassen sich je nach Anforderung verschiedenste Verfahren heranziehen und auch der Kontext der Bewertung kann bei der Implementierung der Methode in ein anderes Licht gerückt werden.

Analysiert man den Prozess der Transformation, so ist neben der Machbarkeit zum Beispiel auch eine Bewertung der Komplexität oder der Schwierigkeit denkbar. Erweitert man den Katalog um individuelle, fachliche, funktionale Anforderungen, so könnte Letzteres die Grundlage für eine Aufwandsschätzung, wie zum Beispiel mit der *Function Point*-Methode, liefern. Die Bewertung oder auch eine Gewichtung von Kriterien erfolgt dabei stets durch den Anwender. Auf die alternativen Anwendungsgebiete wird jedoch nicht weiter eingegangen.

Bei der Bewertung der Transformierbarkeit bewertet der Anwender den Prozess der Transformation bezüglich der Machbarkeit. Folglich wird ein Bewertungsverfahren benötigt, bei dem die Güte der Durchführung – also wie gut oder schlecht etwas durchführbar ist – bewertet werden kann. Im weiteren Sinne beschreibt die Machbarkeit, ob Bedingungen existieren, die dem Vorhaben entgegen stehen und wie komplex die Erfüllung dieser Bedingungen anzusehen ist.[Ang]

Folglich könnten bei der Anwendung der Methode zahlreiche Informationen gewonnen werden, die für das weitere Vorgehen im Projekt eine Rolle spielen. Eine Möglichkeit ist daher, die Handlungsanweisungen als offene Fragen zu formulieren. Die Interpretation, Strukturierung und Auswertung der Antworten stellt aber vermutlich ein ähnlich komplexes Verfahren dar wie der Aufbau eines Lösungskatalogs.

Um standardisierte Antworten zu erhalten, die nicht zuletzt auch die Vergleichbarkeit von Projekten ermöglichen, eignen sich geschlossene Fragen oder Rating-Skalen. Erfolgt die Anwendung der Methode zur Beratung, so ist der zeitliche Faktor bei der Durchführung eines Interviews nicht zu vernachlässigen. Durch vorgegebene Antworten lässt sich die Durchführung eines Interviews zur Bewertung verkürzen.

Schließlich erfolgt die eigentliche Bewertung immer pro Handlungsanweisung. Soll eine Aussage über die grundsätzliche Transformierbarkeit des gesamten Anwendungssystems getätigt werden, so müssen die Wertungen über die einzelnen Items der Checkliste ausgewertet und komprimiert werden. Dazu empfiehlt es sich die Antworten mit einem ordinalen Skalenniveau zu versehen, so dass eine mathematische Auswertung, zum Beispiel die Kalkulation eines Indexes, möglich ist.

4.7. Formalisierung der Methode durch Meta-Modell

Im letzten Schritt der Konzeptionsphase wurde die entwickelte Methode formalisiert und in ein abstraktes Meta-Modell überführt, da sowohl der Aufbau des Katalogs wie auf die Strukturen zum Setzen der Filter sowie das Bewertungsverfahren stark individualisiert werden können. So lässt sich der Katalog um zusätzliche Dimensionen und Attribute erweitern und erlaubt beliebig viele Ausprägungsformen. Die Graphen zur strukturierten Filterung werden demnach auch beliebig breit. Je nach Anzahl der Ausprägungen einer Dimension empfiehlt sich eine mehr oder weniger granulare Struktur zur Filterung mit entsprechender Tiefe.

Letztendlich lässt sich vermutlich keine universelle Datenstruktur und damit auch kein statischer Graph finden, um beliebige Aspekte abbilden zu können. Um die Methode einführen und anwenden zu können muss der Anwender also immer erst die Struktur des Katalogs definieren.

Formalisierte Methodenbeschreibung: Zur Bewertung der Transformierbarkeit werden Lösungsansätze in einer Matrix in verschiedenen Dimensionen eingeordnet. Jede Dimension wird von einem Baum aufgespannt, der die jeweilige Dimension von der Wurzel ausgehend spezialisiert. Die Lösungsansätze werden dabei den Knoten mit dem höchsten Spezialisierungsgrad, also den Blättern, zugeordnet. Um nur relevante Aspekte einer Transformation zu betrachten, durchläuft der Anwender die Bäume und markiert Projektrelevante Knoten, so dass nur Folgeknoten von zuvor markierten Knoten betrachtet werden. Durch diese strukturierte Filterung wird die Matrix auf das nötige Minimum reduziert. Durch die Modellierung von Abhängigkeiten im Baum entartet dieser zu einem gerichtetem Graphen. Die Knoten der Bäume (oder Graphen) wie auch die Lösungsansätze enthalten beschreibende Attribute. Diese ermöglichen die Ableitung von konkreten Handlungsanweisungen. Die Bewertung erfolgt durch ein geeignetes Bewertungsverfahren auf Basis dieser Handlungsanweisungen durch den Anwender.

Die Abbildung 4.5 zeigt das Meta-Modell in seiner Urform, also mit den drei Kerndimensionen und ohne die Berücksichtigung von Abhängigkeiten innerhalb einer Dimension oder zwischen den Lösungsansätzen, als konzeptuelles Klassendiagramm. Die Repräsentation der Graphen erfolgt durch Adjazenz, das heißt durch Abbildung der Kanten mit ihren Endknoten (vgl. Graphentheorie).

Durch das Meta-Modell lässt sich die Methode hinsichtlich der Struktur des Lösungskatalogs sowie der Bewertung individuell ausgestalten und ist damit nicht nur auf die Anforderungen der *msg systems AG* zugeschnitten, sondern auch für andere Anwendungen geeignet.

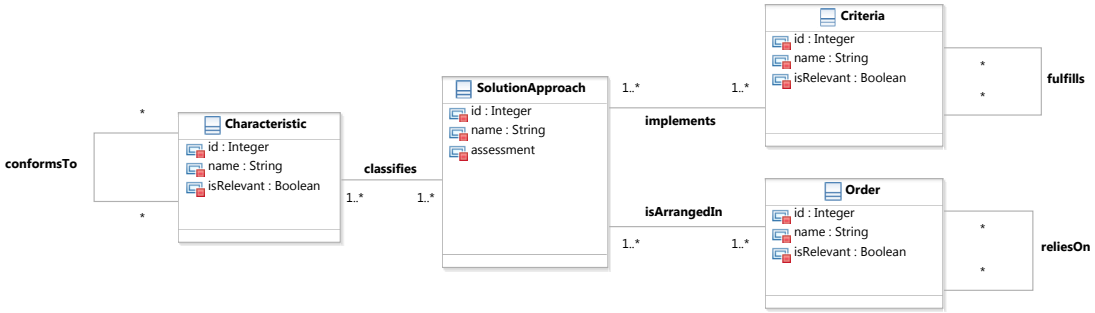


Abbildung 4.5.: Vereinfachtes Meta-Modell der Methode

Ausgehend von der Urform lassen sich auch spezialisierte Formen der Methode ableiten. So etwa für spezielle Disziplinen, die fachliche Attribute zur Kategorisierung von Lösungsansätzen benötigen. So könnte die Methode zum Beispiel in einer Rechtsabteilung implementiert werden wobei Lösungsansätze zur Erfüllung von *Compliance*-Richtlinien explizit nach Branchen, Unternehmensgrößen oder Ländern eingeordnet werden.

Weitere Dimension und Abhängigkeiten können durch zusätzliche Klassen und Assoziationen an das Meta-Modell eingefügt werden. Dies wird im nächsten Kapitel zur Implementierung der Methode verdeutlicht.

5. Implementierung bei der msg systems AG

Im folgenden Kapitel werden die Schritte zur Implementierung des formalen Meta-Modells in der Praxis aufgezeigt. Dazu werden zunächst die Ziele unter dem Gesichtspunkt des konkreten Einsatzzwecks festgelegt. Ausgehend von der Urform werden die individuellen Anpassungen vorgestellt.

5.1. Zielsetzung

Die Methode wurde, wie bereits im Kapitel zur Einführung erwähnt, in enger Zusammenarbeit mit der *msg systems AG* entwickelt. Dort soll sie zukünftig zur Wissensentwicklung, Wissensstrukturierung und zur Wissensverteilung über alle Fachabteilungen (z.B. auch branchenspezifische Abteilungen) hinweg eingeführt und kontinuierlich weiterentwickelt werden. Primäres Ziel war daher eine interdisziplinäre und generische Ausrichtung der Methode in ihren Strukturen.

Die Implementierung erfolgte im Rahmen dieser wissenschaftlichen Arbeit beginnend mit der Disziplin Software-Architektur. Wie in Abschnitt 4.3 beschrieben ist der Aufbau eines Lösungskatalogs mit seiner Filterstruktur fundamentale Voraussetzung zur Anwendung der Methode. Der Grundstein dazu sollte innerhalb der Querschnittsabteilung *Applied Technology Research* gelegt werden.

Anhand des initialen Katalogs, mit hinreichender Vollständigkeit um die Funktionsweise zu demonstrieren, wurden die Prozesse zur Identifikation und Strukturierung von *SaaS*-spezifischen Lösungsansätzen evaluiert. Auf dieser Basis werden die Experten im Umgang mit der Methode geschult, so dass diese in kommenden Projekten wie etwa internen Innovationsprojekten bereits begleitend zum weiteren Aufbau des Lösungskatalogs eingesetzt werden kann. Die Lösungsansätze stellen dabei keine konkreten Anweisungen zur Implementierung dar, sondern benennen lediglich Konzepte, Prinzipien oder Vorlagen zur Realisation.

Die *msg* plant die Methode auch als Anwendungssystem zu implementieren, welches Beratern im Kundengespräch als elektronisches Hilfsmittel dient. Dazu soll ein funktionaler Prototyp erstellt werden, der die benötigten Funktionalitäten zum Filtern und zur Bewertung von Handlungsanweisungen bereitstellt.

5.2. Entwicklung der Katalogstruktur

Um die Methode an die Anforderungen der *msg* anzupassen, wurde das Meta-Modell entsprechend angepasst und erweitert.

Erweiterung um Disziplinen. Mit der Zielsetzung die Methode in mehreren Disziplinen und Fachabteilungen einzusetzen, wurde schließlich eine zusätzliche Dimension einge-

führt, die es erlaubt, diese weitere Zuordnung durchzuführen. Der Katalog baut also auf einer vierdimensionalen Matrix auf. Da die Anzahl der Disziplinen überschaubar bleibt wurde auf die Entwicklung einer Filterstruktur verzichtet. Die Dimension der Disziplinen bildet also keine Hierarchie.

Ebenenmodell zur Positionierung von Lösungsansätzen. Zur Einordnung der Lösungsansätze bezüglich ihrer Position im Gesamtsystem wurde ein individuelles Ebenenmodell in Anlehnung an die klassische Schichten-Architektur nach Sun Microsystems entwickelt. Schichten-Architekturen gelten zwar als gängiges Konzept des *Software Engineerings* doch bieten sie keine ausreichenden Möglichkeiten Lösungsansätze aus anderen Disziplinen wie zum Beispiel der Systemintegration oder *IT-Compliance* einzuordnen. Folglich wurde das Konzept der Schichten-Architekturen um weitere Ebenen erweitert, so dass auch nicht-architektonische Lösungsansätze, wie etwa zum Betrieb von IT-Infrastrukturen oder zur Einhaltung gesetzlicher Rahmenbedingungen in den Katalog mit aufgenommen werden können. Die Vorlage zur Erweiterung lieferte das Konzept der *Technology Stacks*, das neben der Software auch physikalische Komponenten umfasst.

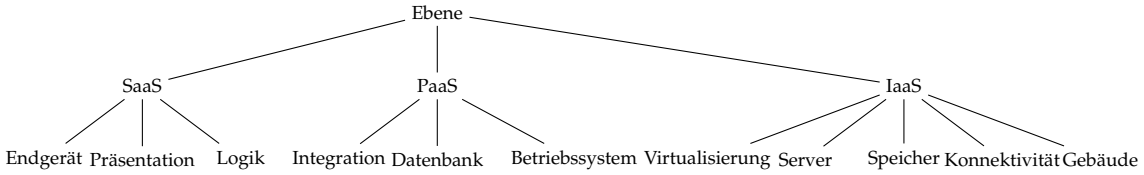


Abbildung 5.1.: Filterstruktur für den Cloud-Stack

Die durch die Vereinigung der beiden Modelle erzeugten Ebenen lassen sich zudem auch auf die bekannten Service-Modelle des *Cloud Computings* zurückführen. Somit können auch *SaaS*-spezifische Anforderungen an die Plattform sowie an die *Cloud*-Infrastruktur gestellt werden. Abbildung 5.1 zeigt die Baumstruktur zur Filterung.

Identifikation von Anforderungen und Charakteristiken. Die Basis für die Identifikation von Anforderungen stellen die in Abschnitt 2.2 vorgestellten Kriterien von *Cloud Computing* und *SaaS* dar. Mit dem Fokus auf Anforderungen an die Software-Architektur wurden daraus die wesentlichen, architektonisch relevanten Kriterien extrahiert:

- IP-basierter Zugriff
- Elastizität der Ressourcen
- Messung der Nutzung
- gemeinsame Nutzung der Ressourcen mit Dritten
- Verwaltung der bereitgestellten Ressourcen durch den Nutzer
- Erweiterbarkeit

Kriterien die per Definition abgeleitet wurden, werden im folgenden als *SaaS Essentials* bezeichnet. Sie bilden stets die erste Ebene der Filterstruktur der Anforderungen.

Die diversen Definitionen beinhalten meist auch Merkmale der genannten Kriterien. So spricht man von rascher Elastizität, einer Bereitstellung in Echtzeit oder dem allgegenwärtigen Zugriff auf Ressourcen. Bei der methodeninternen Definition der Anforderungen an eine *SaaS*-Lösung wurde auf diese beschreibenden Zusätze bewusst verzichtet, da sie bereits erste Charakteristiken darstellen, die in der Methode eigens behandelt werden. Vor allem das häufig genannte Kriterium *On Demand* könnte so fälschlicherweise als Anforderung im Sinne der Methode verstanden werden. Um die nächsten Schritte zur Entwicklung der Katalogstruktur besser zu verstehen, werden die sechs *Essentials* kurz erläutert. Diese Beschreibungen sollen später auch zu einem gemeinsamen Verständnis unter den Anwendern beitragen.

Der IP-basierte Zugriff auf die *Cloud* beschreibt die Anforderung über standardisierte Internet-Technologien und über verteilte Netze wie dem Internet auf das Anwendungssystem zugreifen zu können. Der Einsatz von Standards wie dem IP-Protokoll ermöglicht, eine breite Nutzergruppe anzusprechen und zwar unabhängig von Endgeräten, Anbindungen oder Standorten. Zudem wird eine Integration mit Fremdsystemen, etwa zum Datenaustausch möglich (z.B. *EDI*).

Durch die Elastizität von Ressourcen können diese an den aktuellen Bedarf dynamisch angepasst werden. Nur so sind zum Beispiel flexible Abrechnungsmodelle wie *Pay per use* überhaupt erst realisierbar. Zudem ermöglicht diese Anforderung Wachstum ohne dass Nutzer Ressourcen vorhalten müssen. Elastizität heißt aber auch Rezessionen ausgleichen zu können. Die Elastizität bezieht sich nicht zwingend nur auf Kapazitäten sondern gegebenenfalls auch auf den Leistungsumfang.

Die Messung der Nutzung ist Voraussetzung zur verbrauchsbasierten Abrechnung. Gerade auf einer geteilten Infrastruktur ist es wichtig zu wissen, welcher Nutzer welche Ressourcen in welchem Ausmaß in Anspruch genommen hat. Doch nicht nur zu Abrechnung, sondern auch zur Einhaltung von *Service Level Agreements* und der Partitionierung der Ressourcen (z.B. Quotas) und auch zur Durchführung eines Lastenausgleichs müssen der Verbrauch und die Auslastung erfassbar sein.

Mit der gemeinsamen Nutzung von Ressourcen mit Dritten ist die Bereitstellung des Angebots in einer 1:N-Beziehung gemeint. Der Anbieter betreibt eine Infrastruktur, auf der mehrere Nutzer in der Lage sind gleichzeitig dem Zweck des Angebots nachzugehen. Dabei gibt es mehrere Varianten zur Realisierung dieses Kriteriums, die sich darin unterscheiden bis zu welcher Ebene sich Nutzer eine Komponente teilen.

Eine eigenständige Verwaltung der bereitgestellten Ressourcen durch den Anwender selbst erfordert zusätzliche Benutzerschnittstellen zur Durchführung und Überwachung von Aufgaben rund um die Verwaltung. Darunter fallen neben der Steuerung des Ressourcenbedarfs auch Möglichkeiten zur Anpassung und Konfiguration des Angebots in einem vom Anbieter eingeschränkten Maß.

Neben diesen eingeschränkten Anpassungs- und Konfigurationsmöglichkeiten fordert die Erweiterbarkeit zusätzliche Möglichkeiten zur Individualisierung. Der Nutzer adaptiert beispielsweise Datenstrukturen oder implementiert eigene Module (z.B. *Plugins*) ohne den gemeinsamen Quellcode zu ändern.

Beim Aufbau der Filterstruktur zu den Anforderungen ausgehend von diesen sechs *Essentials* war die grundsätzliche Idee die *Essentials* soweit zu spezialisieren bis sie auf bekannte, standardisierte Lösungsansätze zurückgeführt werden können. Dabei bieten gängige nicht-funktionale Anforderungen den idealen Spezialisierungsgrad, da sie nicht *Cloud*-spezifisch und unabhängig von der Klasse des Anwendungssystems zu finden sind und somit ein hohes Potenzial für die Verfügbarkeit von standardisierten Lösungsansätzen aufweisen. Der ISO-Standard 9126-1 fasst gängige nicht-funktionale Anforderungen zusammen. Auf dieser Grundlage konnte eine Zuordnung zu den *Essentials* erfolgen.

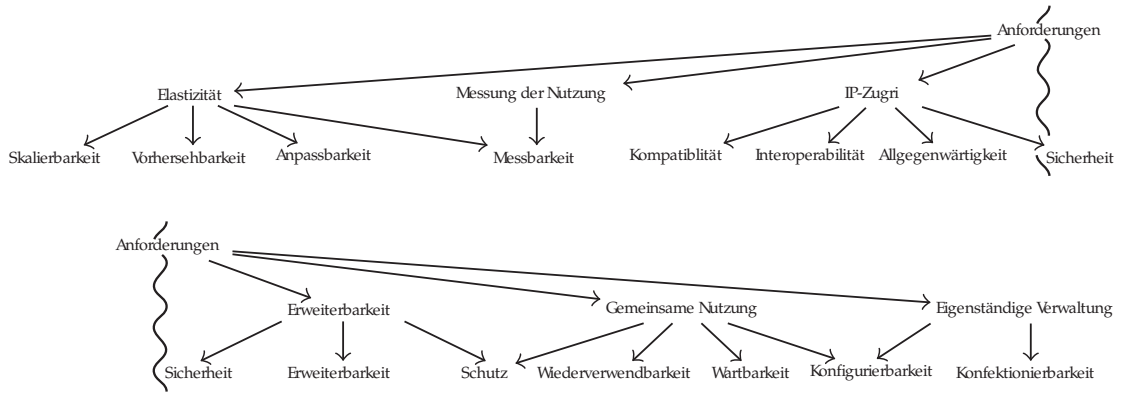


Abbildung 5.2.: Spezialisierung der SaaS-Kriterien zu nicht-funktionalen Anforderungen

Die Abbildung 5.2 zeigt eine Möglichkeit zur Spezialisierung der *Essentials* zu nicht-funktionalen Anforderungen. Beispielweise erfordert Elastizität eines Anwendungssystems prinzipiell eine Skalierbarkeit der Komponenten. Doch um Lastspitzen auszugleichen, muss die Infrastruktur auch in der Lage sein diese frühzeitig zu erkennen oder idealerweise vorherzusehen. Die gemeinsame Nutzung der Ressourcen erfordert eine Wiederverwendbarkeit von diversen Komponenten und den Schutz von Nutzern durch mutwillige oder unbeabsichtigte Angriffe von innerhalb des Anwendungssystems. Gerade in einer solchen Umgebung muss der Anbieter sicherstellen das System warten zu können ohne die nutzer-spezifischen Konfigurationen oder gar Daten zu gefährden. Die Messung der Nutzung erfordert die Erfassung, Speicherung und Aufbereitung von Nutzungsdaten, so wie sich auch dynamische Skalierbarkeit nur realisieren lässt wenn die aktuelle Auslastung der Komponenten bekannt ist.

Auf diese Weise lassen sich alle der wesentlichen Kriterien auf bekannte Problemstellungen des *Software Engineerings* zurückführen wodurch die Identifikation von Lösungsansätzen möglich wird. Die nicht-funktionalen Anforderungen stellen in diesem Zusammenhang Pole dar, an denen sich die Lösungsansätze stärker oder schwächer orientieren. Eine strikte und eindeutige Zuordnung ist auch nicht in allen Fällen möglich oder sinnvoll.

Bei der Spezialisierung der wesentlichen Kriterien wurde zudem folgender Zusammenhang festgestellt. Die eigenständige Verwaltung durch den Nutzer, die Elastizität der Ressourcen wie auch die gemeinsame Nutzung mit Dritten implizieren eine weitere essentielle Anforderung. So benötigen die drei genannten *Essentials* ein zentrales System zur

Verwaltung von Ressourcen das ermöglicht eine Bereitstellung oder Einstellung (also Minderung) von Kapazitäten und Leistungen durchzuführen und dabei jederzeit die Verfügbarkeit von Ressourcen überwacht und inventarisiert. Diese Verwaltung der Ressourcen wird auch als *Provisioning* bezeichnet. Die Verfügbarkeit eines solchen Systems zur Provisionierung stellt ein weiteres wesentliches Kriterium dar, welches zwar nicht explizit in den Definitionen genannt wird jedoch gerade in Bezug auf die Software-Architektur eine zentrale Rolle spielt [Zei12]. So ist eine Elastizität der Ressourcen nur möglich wenn Kapazitäten angepasst werden können ohne den Zustand des Systems zurückzusetzen. Nutzer können nur dann eigenständig Konfigurationsänderungen anstoßen, wenn diese auch selbstständig umgesetzt werden. Und schließlich ist eine gemeinsame Nutzung von Ressourcen nur möglich wenn sich die Infrastruktur partitionieren lässt und sich die Partitionen einzeln verwalten lassen. Da das System zur Provisionierung gleich für mehrere der *Essentials* als funktionale Anforderung gilt wird es als Schatten-Kriterium bezeichnet. Es beinhaltet alle Komponenten, Schnittstellen und Prozesse zur Verwaltung der Ressourcen und Zustände [msd]. Bei der Implementierung der Methode bei der *msg* wurde dieses Schatten-Kriterium jedoch nicht als *Essential* sondern als funktionale Anforderung in den Katalog aufgenommen.

Die Dimension Anforderung wird also zum Zeitpunkt der Implementierung der Methode von den *Essentials* mit ihren nicht-funktionalen Anforderungen sowie dem Schatten-Kriterium aufgespannt. Neben dieser und den bereits vorgestellten Dimensionen der Disziplinen und Ebenen fehlt noch die vierte Dimension der Charakteristiken zur projektspezifischen Filterung von Lösungsansätzen. Bei der Entwicklung einer Struktur für die Charakteristiken von Lösungsansätzen kann, wie im vorherigen Kapitel zur Entwicklung der Methode beschrieben, auf zwei Weisen vorgegangen werden.

So lassen sich manche bereits vor der Identifikation von Lösungsansätzen als typische Charakteristiken für die speziellen Anforderungen bestimmen (*Top-Down*). In anderen Fällen können die Charakteristiken erst von den Lösungsansätzen hergeleitet werden (*Bottom-Up*). Um Lösungsansätze für die Anforderung Skalierbarkeit zu filtern wurden sofort die Lastprofile einer Anwendung, wie zum Beispiel schnelles oder unvorhersehbares Wachstum, als entscheidende Charakteristik erkannt ohne Lösungsansätze parat zu haben. Im Fall der Anforderung Ubiquität als Spezialfall für den IP-basierten Zugriff wurden erst nach der Ermittlung von Lösungsansätzen geeignete Charakteristiken, wie beispielsweise die Verbindungsart oder die geographische Verteilung der Nutzer oder die Anzahl der Endgeräte und Plattformen bestimmt.

Typisierung der Entitäten. Die interdisziplinäre Anwendung der Methode erfordert nicht nur zur Filterung sondern auch zur Ableitung von Handlungsanweisungen zusätzliche Unterscheidungsmerkmale – auch bei den Lösungsansätzen. Aus diesem Grund wurde eine Typisierung der Entitäten eingeführt. So lassen sich in der Disziplin Software-Architektur Anforderungen, wie bereits erläutert, grundlegend in funktionale und nicht-funktionale Anforderungen einteilen. Die *Essentials* spielen dennoch eine übergeordnete Rolle. Andere Disziplinen wie *Compliance* erfordern unter Umständen eigene Typen, zum Beispiel Richtlinien oder Gesetze.

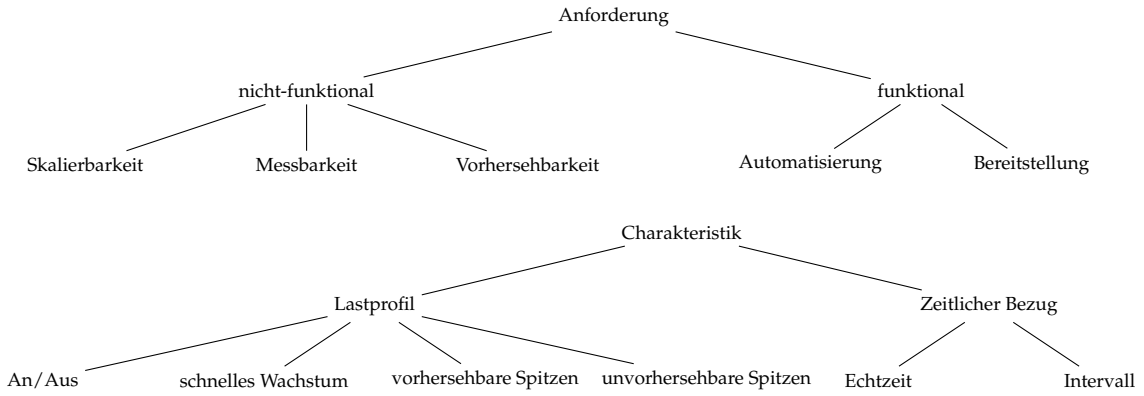


Abbildung 5.3.: Partitionierte Baumstrukturen durch Typisierung

Durch die Typisierung in den Filterstrukturen können die Baumstrukturen in sich gruppiert werden. Jeder Typ entspricht so einem Teilbaum. Auf diese Weise lassen sich Charakteristiken bestimmten Anforderungen zuordnen. In der folgenden Abbildung 5.3 können so die einzelnen Lastprofile der Anforderung Skalierbarkeit zugeschrieben werden.

Bei der Implementierung der Methode bei der *msg systems AG* wurden initial folgende Typen eingeführt:

Anforderung → {Essentielles Kriterium, funktionale Anforderung, nicht-funktionale Anforderung}

Charakteristik → {Lastprofil, Reifegrad, Verbindung, Geographische Verteilung, Zeitlicher Bezug}

Ebene → {Service-Modell, Containerschicht}

Lösungsansätze → {Konzept, Prinzip, Software-Pattern, Komponente, Schlüsseltechnologie}

Die Typenbildung unter den Lösungsansätzen ermöglicht in Verbindung mit den Disziplinen eine zweckgebundene Ausgabe der Handlungsanweisung mit Blick auf die Bewertung. Dadurch lassen sich beispielsweise Architektur-Konzepte von Software-Architekten und *Design Patterns* oder der Einsatz von Schlüsseltechnologien durch Entwickler bewerten.

5.3. Rating-Skala zur Bewertung der Transformierbarkeit

Die in Abschnitt 4.1 identifizierten Szenarien erfordern eine unkomplizierte Bewertung, die im Rahmen von Workshops oder Interviews zur Beratung durchgeführt werden kann. Gemäß der Zielsetzung soll durch den Einsatz der standardisierten Methode der Lernkurveneffekt gefördert werden. Dies erfordert eine Vergleichbarkeit von Projekten hinsichtlich ihrer Transformierbarkeit.

Folglich wurde ein Bewertungsverfahren gewählt, das eine vergleichbare aber dennoch graduelle Wertung ermöglicht. Die Anwender bewerten anhand einer Rating-Skala folgende Handlungsanweisungen:

> „Erfülle die/das *«Typ der Anforderung» «Anforderung»* mit dem/der *«Typ der Charakteristik» «Charakteristik»* durch Implementierung der/des *«Typ des Lösungsansatz» «Lösungsansatz»* in dem/der *«Typ der Ebene» «Ebene»*."

Beispiel:

> „Erfülle die/das *nicht-funktionale Anforderung Skalierbarkeit* mit dem/der *Lastprofil An/Aus* durch Implementierung der/des *Prinzips Stateless-Architektur* in dem/der *Containerschicht Präsentation*."

Dabei wird nicht bewertet wie gut oder wie schlecht – im Sinne des Aufwands, Schwierigkeitsgrads oder der Komplexität – die Anweisung umzusetzen ist sondern lediglich die generelle Machbarkeit. Die Skala enthält dazu drei Bewertungsmöglichkeiten:

- nicht implementierbar
- implementierbar
- bereits implementiert

Durch diese Bewertung lassen sich einerseits *K.O.*-Kriterien identifizieren, andererseits können durch weitere Auswertung erste Aussagen getätigt werden zu welchem Grad ein vorhandenes Anwendungssystem eventuell sogar schon *SaaS*-fähig ist.

5.4. Durchführung von Workshops und Ergebnisse zum Aufbau des Katalogs

Im Rahmen der Implementierung wurden Workshops zur Wissensentwicklung respektive zur Identifikation von Lösungsansätzen zum Aufbau des Katalogs durchgeführt, um eine hinreichende Vollständigkeit zur weiteren Einführung im Unternehmen zu erzielen. Dazu wurden ausgewählte Software-Architekten der Abteilung *Applied Technology Research* bezüglich der Möglichkeiten oder Ansatzpunkte zur Realisierung der *SaaS Essentials* befragt.

Wie auch die Charakteristiken lassen sich Lösungsansätze nicht immer sofort einer nichtfunktionalen Anforderung zuordnen und somit direkt identifizieren. Eine bewährte Vorgehensweise zur Wissensentwicklung war die schrittweise Vertiefung von zunächst oberflächlichen Ideen die direkt von den *Essentials* hergeleitet werden konnten. Dabei war auch die differente Betrachtung von allgemeinen Lösungsansätzen bezüglich ihrer Position im Gesamtsystem – als bei dieser Implementierung ihrer übergeordneten Containerschicht - hilfreich. Mit dieser Vorgehensweise können aus allgemeinen Konzepten oder Prinzipien spezielle Lösungsansätze erarbeitet werden, die auch nachträglich Anforderungen mit hohem Spezialisierungsgrad zugeordnet werden können.

Beispielsweise konnte so aus dem Prinzip der horizontalen Partitionierung als Ansatz für Skalierbarkeit in der Software-Architektur der datenbankspezifische Lösungsansatz *Sharding* erfasst werden und kategorisiert werden.

Während des Projekts wurden insgesamt fünf Workshops zur Identifikation von Lösungsansätzen durchgeführt. Neben zahlreichen Lösungsansätzen konnten dabei folgende grundsätzliche Erkenntnisse betreffend der Software-Architektur festgehalten werden:

Die Möglichkeiten zur Elastizität einer *SaaS*-Lösung werden maßgeblich durch das Lastprofil der Anwendung bestimmt. Dabei lassen sich IT-Lastprofile durch vier Parameter beschreiben: die initiale Last, die Dauer bis zum Erreichen der Spitzenlast sowie die minimal und maximal zu erwartende Spitzenlast [RN]. Somit ist für Auswahl von Lösungsansätzen zur Implementierung einer skalierbaren Architektur entscheidend in welchen Größenordnungen die Auslastung variiert und wie kurzfristig die Schwankungen auftreten. Zur Zuordnung von Lösungsansätzen wurden vier typische Lastprofile herangezogen:

- an/aus
- schnelles Wachstum
- unvorhersehbare Spitzen
- vorhersehbare Spitzen

Das Lastprofil „an/aus“ ist typisch für Administrations- und Dispositionssysteme in denen beispielsweise periodische Aufgaben wie Abrechnungen, Zahlungsläufe oder andere Transaktionen ablaufen. Schnelles Wachstum ist dagegen schwer zu planen. Anwendungssysteme die Netzwerkeffekten ausgesetzt sind wachsen ab einer kritischen Masse exponentiell. Dazu zählen unter anderem auch virale Dienste oder soziale Portale. Unvorhersehbare Lastspitzen werden durch unbekannte Größen meist aus Umwelt eines Informationssystems erzeugt. Folglich ist auch die Höhe der Lastspitzen unbekannt. Beispiele sind *E-Commerce*-Anwendungen oder Wissenssysteme wie Nachrichten-Portale oder Suchmaschinen. Dagegen sind bei vorhersehbaren Spitzen die Größen planbar. In diese Kategorie fallen Anwendungen die saisonalen Ausschlägen unterliegen wie Bestellsysteme oder Anwendungen die nur für ein bestimmten Ereignis betrieben werden [Eil].

In einer *SaaS*-Lösung können verschiedene Technologien und Konzepte angewandt werden um eine grundlegende Skalierbarkeit zu erreichen. Gerade in einer *Cloud*-Infrastruktur lassen sich einzelne Instanzen dank der Virtualisierung von Ressourcen vertikal – also durch Erweiterung der Kapazitäten der vorhandenen Instanz – sehr hoch skalieren ohne das dies Auswirkungen auf die Architektur hat. Diese einfache Möglichkeit stößt jedoch irgendwann an seine Grenzen, so dass horizontal – also durch Hinzufügen von Instanzen – skaliert werden muss. Auch dies ist bis zu einem gewissen Maße mit einfachen Mitteln, wie zum Beispiel zusätzlichen Cluster-Knoten an Last-intensiven Komponenten, möglich. Die augenscheinlich endlose Skalierbarkeit in der *Cloud* durch die beliebige Hinzu- oder Wegnahme von Ressourcen über alle Schichten der Anwendung hinweg erfordert jedoch die Umsetzung komplexer Architekturprinzipien wie zum Beispiel *Shared Nothing* oder *Stateless Clustering*.

Um Lösungsansätze zur Implementierung einer Nutzungsmessung zu identifizieren wurden drei grundlegende Metriken erkannt. Unabhängig davon ob die Daten zur Abrechnung oder nur intern verwendet werden können die folgenden Basisgrößen unterschieden werden:

- Zeit (z.B. Sitzungsdauer, Prozesslaufzeit)
- Ressource (z.B. Speicherplatz, Prozessorauslastung, Anzahl der Instanzen)
- fachliche Größe (z.B. Anzahl der Benutzer, Umsatzgrößen, Anzahl der Positionen)

Die Software-Architektur betreffend sind die beiden ersten von besonderer Bedeutung. Entscheidend ist, dass die Nutzung oft nur an ausgewählten Komponenten gemessen und vor allem einzelnen Nutzern zugeordnet werden kann. So ist die Erfassung von Übertragungsvolumen oft nur im *Frontend* möglich und sinnvoll. Die grundsätzliche Lösungsansatz zur Messung des Ressourcenverbrauchs ist dennoch die vorhandenen *Monitoring*-Funktionalitäten der jeweiligen Container über deren Schnittstellen ab zugreifen oder nach dem *Pipes and Filters*-Muster vorzugehen um individuelle Ressourcen zu messen. Je komplexer und individueller die zu messende Größe desto aufwendiger ist folglich die Implementierung. Die Erfassung von Zeiten klingt auf den ersten Blick harmlos. Anmeldevorgänge und Nutzungsdauer können bekanntlich in der Präsentationsschicht erfasst werden, doch selbst dieses Prinzip wird in einer *Stateless Architecture* zu einem komplexen Unterfangen da serverseitig kein Zustand und folglich auch keine Sitzung existiert. Die Messung von Zeit und Ressourcen ist also in einer hoch-skalierbaren Anwendung deutlich erschwert.

Die Kontierung von fachlichen Größen, die aus dem Datenbestand extrahiert oder sogar kalkuliert werden müssen, lässt sich über standardisierte Lösungen zur *Business Intelligence* oder durch eigene Abfragen an Datenbanken oder Verzeichnisdienste realisieren.

Auch für die Implementierung eines *Shared Service* lassen sich Lösungsansätze mit unterschiedlicher Komplexität ausmachen. Primär muss dazu eine Grundsatzentscheidung getroffen werden. Schließlich lässt sich die gemeinsame Nutzung von IT-Ressourcen grundsätzlich über zwei Arten realisieren:

- mehrere Mandanten in einer Instanz
- mehrere Instanzen auf einem System

In einem mandantenfähigen (engl. *multi-tenancy*) Anwendungssystem teilen sich Mandanten – im Sinne von Nutzergruppen – eine Laufzeitumgebung. Dabei erfolgt eine logische Isolation der Mandanten. Die Mandanten werden folglich stets auf Basis des selben Quellcodes beliefert. Fehler in der Laufzeitumgebung oder Wartungsarbeiten betreffen alle Mandanten gleichzeitig und im selben Ausmaß.

Laufen auf der *Cloud*-Infrastruktur mehrere Instanzen einer Anwendung (engl. *multi-instancy*) pro Nutzergruppen parallel so werden diese zum Beispiel durch die Partitionierung der Infrastruktur (virtuell oder physikalisch) getrennt. Jeder Kunde wird über eine dedizierte Laufzeitumgebung versorgt.

Gemäß den Kriterien für eine *SaaS*-Lösung aus Abschnitt 2.2 stellt die gemeinsame Nutzung von IT-Ressourcen auf Basis mehrfacher Instanzen kein vollwertiges *SaaS* dar. Dennoch lassen sich auch in der Praxis renommierte Anbieter finden, die ihre Anwendung auf dieser Basis bereitstellen oder sogar vor der logischen Isolation von Kundendaten warnen [Bab11]. Beim Aufbau eines *Shared Service* stellt sich also die Frage, auf welcher Ebene und bis zu welchem Grad sich die verschiedenen Nutzergruppen die Komponenten teilen.

Nutzer- oder kundenspezifische Anpassungs- und Konfigurationsmöglichkeiten sind in der heutigen Zeit bei *On-Premise* Anwendungssyteme häufig Standard. In einer *SaaS*-Lösung stellen vor allem Möglichkeiten zur Individualisierung, das heißt Anpassungen die über die Konfigurationsmöglichkeiten die vom Anbieter vorgegeben werden hinausgehen, besondere Anforderungen an die Architektur. Neben Basis-Anforderungen wie der Anpassung der Benutzeroberfläche, zum Beispiel durch Lokalisierung oder eigener *Designs* und Profile, konnten im Rahmen der Workshops folgende Komplexitätstreiber identifiziert werden.

- Adaption von Datenstrukturen
- Abbildung von Geschäftsprozessen
- Integration von Eigenentwicklungen

Wie auch die Messung der Nutzung steigt die Komplexität dieser Anforderungen im Bezug auf den Grad der Mandantenfähigkeit.

Sowohl bei der Elastizität, der gemeinsamen Nutzung der IT-Ressourcen als auch bei der Konfigurier- und Erweiterbarkeit können verschiedene Komplexitätsstufen erreicht werden. Je nachdem in welcher Komplexitätsstufe diese Anforderungen in der *SaaS*-Lösung implementiert werden spricht man vom Reifegrad der *SaaS*-Anwendung [for08].
Nach *Microsoft* existieren vier Reifegrade (engl. *Maturity Levels*):

- ad hoc/individuell
- konfigurierbar
- konfigurierbar & mandantenfähig
- skalierbar, konfigurierbar & mandantenfähig

Der erste Reifegrad „ad hoc/individuell“ beschreibt das ursprüngliche *ASP*-Modell. Jedem Kunden wird eine dedizierte Instanz bereitgestellt, die individuell angepasst werden kann. Beim zweiten Reifegrad „konfigurierbar“ greift jede Instanz auf den selben Quellcode zurück, jedoch sind die Instanzen pro Kunde konfigurierbar. Dennoch bleiben die Instanzen isoliert.
Der dritte Reifegrad wird durch effiziente Mandantenfähigkeit charakterisiert. Alle Kunden greifen auf ein und die selbe, konfigurierbare Instanz zu. Für den Anbieter ermöglicht dies eine Effizienzsteigerung bei gleichzeitiger Kostensenkung. Jedoch ist nur eine vertikale Skalierbarkeit gegeben.

Im vierten und höchsten Reifegrad ist auch horizontale Skalierbarkeit möglich. Dazu werden die Kunden über mehrere identische Instanzen balanciert verteilt wobei die Daten dabei selbst isoliert bleiben. Der Anbieter kann die Anzahl der Instanzen je nach Bedarf steuern. Die Zuordnung von Anwendern, Konfigurationseinstellungen und Daten erfolgt über Meta-Daten. [CC06]

Bei der Entwicklung einer *SaaS*-Lösung ist folglich ein *Trade Off* zwischen der Isolation von Kunden und ihren Daten sowie der gemeinsamen Nutzung von möglichst vielen Ressourcen und damit verbesserter Skalierbarkeit und einfacher Verwaltung zu wählen.

Neben diesen primären Charakteristiken und dazugehörigen Lösungsansätzen konnten in den Workshops eine weitere transformationrelevante Erkenntnis gewonnen werden. Wie schon in der Einführung beschrieben, hat der Trend zu serviceorientierten Architekturen maßgeblich die Entwicklung von *Software as a Service* beeinflusst.

In einer *SOA* sind Komponenten als wiederverwendbare *Web Services* ausgerichtet. Ein Dienst stellt eine eindeutige und wiederholbare Funktion über eine definierte Schnittstelle bereit. Die Abbildung von Prozessen erfolgt durch lose Kopplung der Dienste [Erl05].

Schließlich bietet ein *On-Premise* Anwendungssystem, dass als eine *Service-orientierte Architektur (SOA)* implementiert wurde, beste Voraussetzungen für eine Transformation. So ermöglicht die Wiederverwendbarkeit von Komponenten prinzipiell auch eine Mandantenfähigkeit. Die Kommunikation über Web-basierte Schnittstellen liefert die nötige Flexibilität beim ubiquitären Zugang über verschiedene Benutzerschnittstellen und Endgeräte. Die lose Kopplung der Dienste erlaubt granulare Skalierbarkeit. Dennoch müssen *SaaS*-Anwendungen nicht zwangsläufig als *SOA* aufgebaut werden.

5.5. Entwicklung eines Prototyps zur Tool-Unterstützung

Neben der Implementierung der Methode wurde ein Prototyp zur Tool-Unterstützung entwickelt. Dieser sollte die grundlegenden Funktionalitäten beinhalten, so dass die Methode exemplarisch angewendet werden kann.

Da die Methode auch in Abteilungen abseits der Disziplin *Software Engineering* erprobt und gegebenenfalls angepasst werden muss wurde der Prototyp mit der Anwendung *Microsoft Access* erstellt. Dabei handelt es sich um eine Desktop-Datenbankanwendung mit der auch ohne höhere Programmierkenntnisse durchaus auch anspruchsvolle Anwendungen und Datenbanken entwickelt werden können. Somit sind auch Experten von Softwarefernen Abteilungen in der Lage die Datenstrukturen weiter zu entwickeln. Zudem lässt sich eine mit *Access* erstellte Datenbank schnell und einfach verteilen ohne komplexe Laufzeitumgebungen samt Bibliotheken bereitstellen zu müssen.

Bevor die Entscheidung auf diese Entwicklungsumgebung fiel, wurde versucht die Matrix in einer herkömmlichen Tabellenkalkulation zu implementieren. Schnell war jedoch klar, dass damit die Mehrdimensionalität, vor allem was das Filtern betrifft, kaum abzubilden ist. Die Repräsentation der Baum- beziehungsweise Graphenstrukturen aus dem Meta-Modell konnten in der relationalen Datenbank problemlos umgesetzt werden.

Abbildung 5.4 zeigt das relationale Datenmodell des Prototyps. Dieses wurde zur Im-

plementierung des Tools aus dem konzeptuellen Klassendiagramm des Meta-Modells abgeleitet. So sind alle Klassen und Assoziationen als Relation angelegt. Die Datenbank des Prototyps entspricht somit mindestens der 2. Normalform (vgl. Relationale Entwurfstheorie). Dadurch bleibt auch Integrität des Katalogs erhalten, da durch das Löschen von Datensätzen einer Entität auch alle Assoziationen entfernt werden. Auch Änderungen müssen deshalb nicht redundant gepflegt werden.

Neben dem DBMS stellt *Access* auch die Werkzeuge zum Aufbau von Benutzerschnittstellen bereit. Der Zugriff auf die Tabellen kann dabei über *SQL*-Abfragen oder proprietäre Methoden der Programmiersprache *Visual Basic* erfolgen. Die Entwicklung einer graphischen Benutzeroberfläche ist über Formulare möglich.

Die zur Methodenanwendung erforderlichen Funktionalitäten werden über folgende Formulare bereitgestellt:

- Vorauswahl zur Definition der Einstiegspunkte (Disziplinen, Essentials, Ebenen)
- Spezifikation von Charakteristiken für die durch die Vorauswahl bedingten Anforderungen
- Ausgabe der Handlungsempfehlungen als Liste mit Bewertungsfeldern
- Berichtserstellung zur weiteren Auswertung

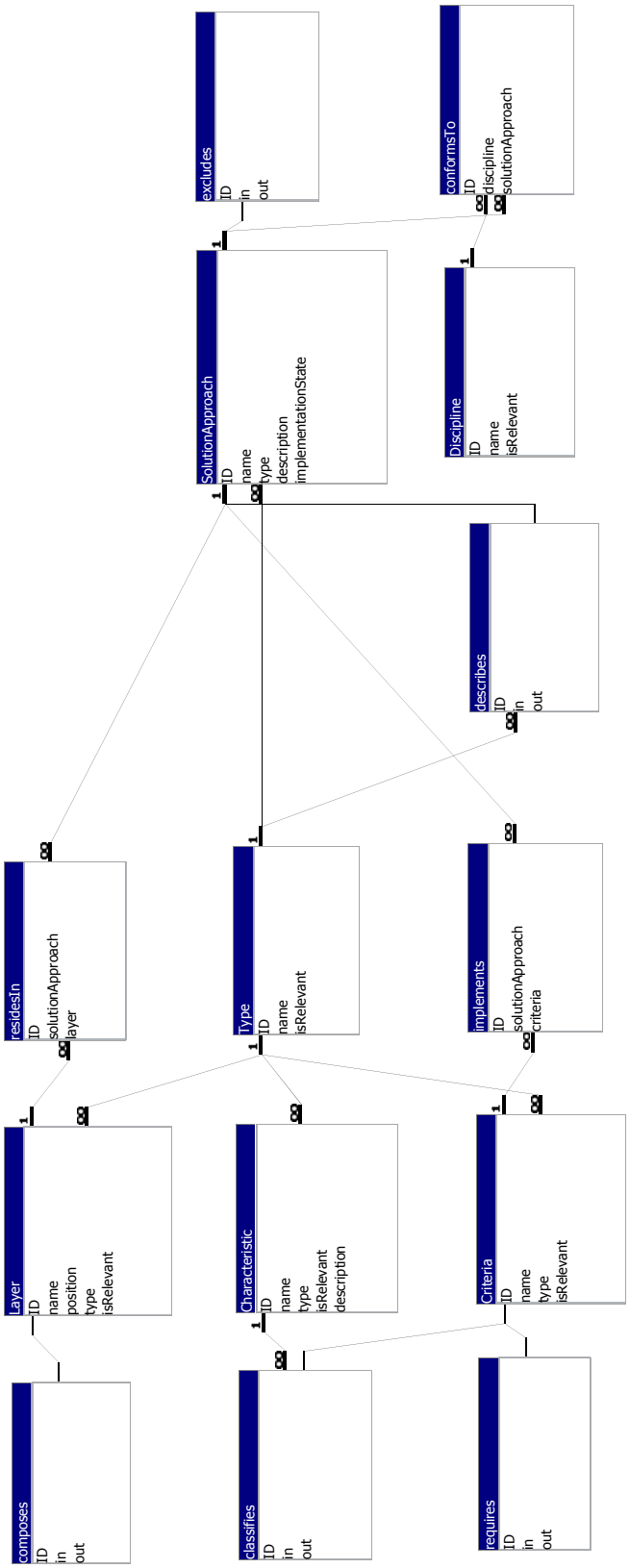


Abbildung 5.4.: Relationales Datenmodell der Methode für die msg

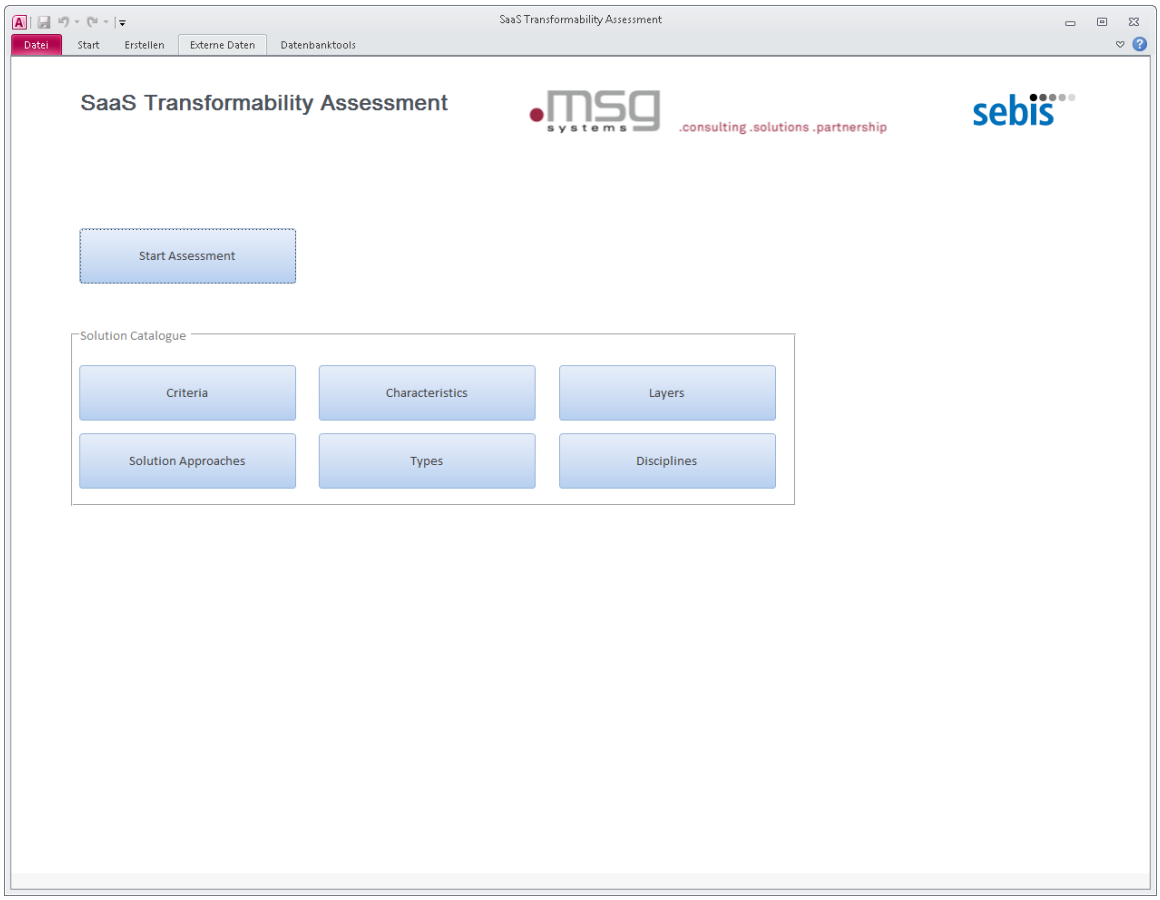


Abbildung 5.5.: Startansicht des Tools

Beim Starten des *Tools* gelangt der Anwender zunächst auf das Hauptformular. Hier hat der derzeit zwei Optionen: die Durchführung einer Bewertung oder die Pflege des Katalogs. Spezifische Formulare zum internen Wissenstransfer sind nicht enthalten, da der Fokus der Arbeit auf dem Szenario der Bewertung liegt. Zur Pflege des Katalogs können die diversen Entitäten, wie Kriterien, Ebenen des Schichtenmodells, Lösungsansätze und weitere bearbeitet und gelöscht werden oder neue hinzugefügt werden. Für jede Entität steht dazu ein eigenes Formular zur Verfügung, in dem auch die Assoziationen, also die Beziehungen untereinander, gepflegt werden können. Bei den Lösungsansätzen erfolgt zusätzlich eine Zuordnung zu den Dimensionen und Typen.

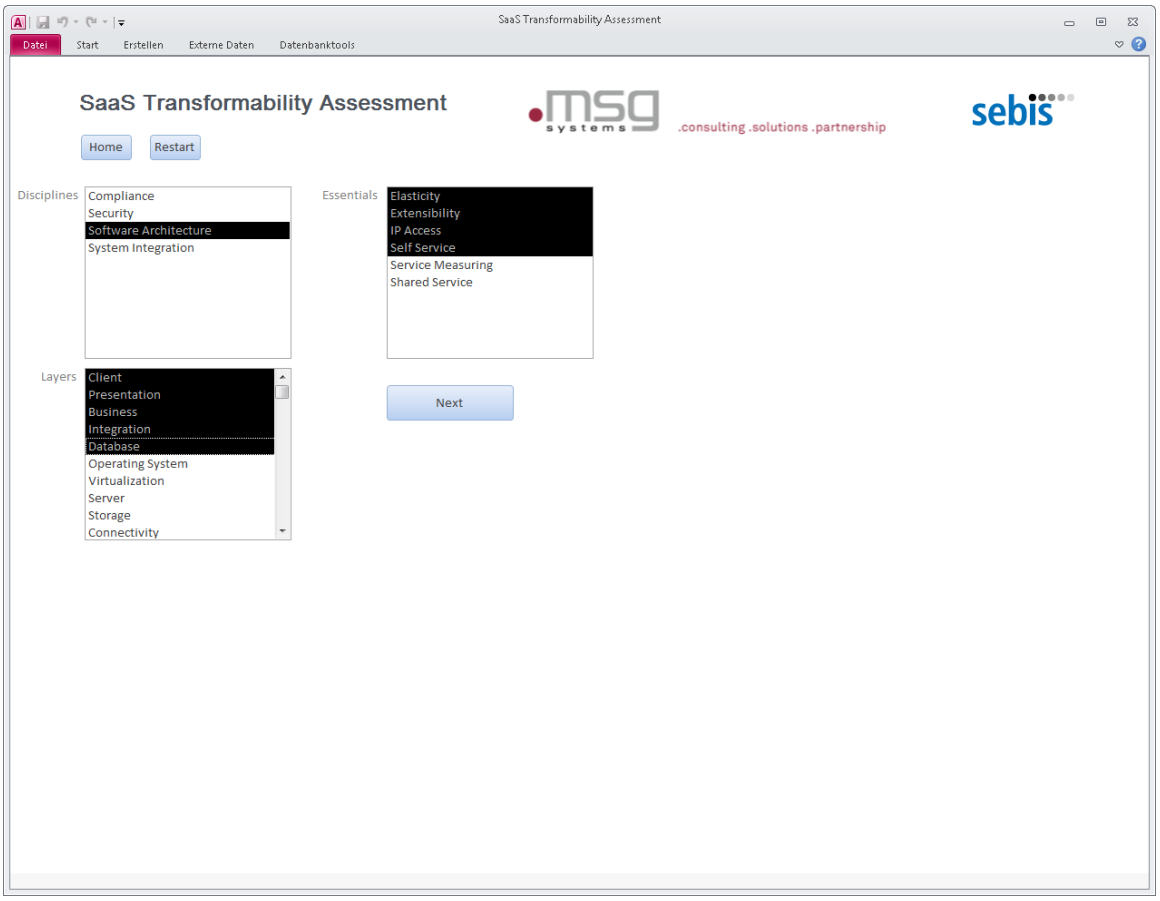


Abbildung 5.6.: Vorauswahl der Aspekte zur Bewertung

Startet der Anwender den Bewertungsprozess so muss er zu Beginn Einstiegspunkte definieren. Diese Vorauswahl dient dazu die Anforderungsspezifikation zu verkürzen. Auf diese Weise kann die Bewertung zielgerichtet, zum Beispiel für eine einzelne Disziplin oder nur für bestimmte Container (z.B. der Datenbank) erfolgen. Sicherlich wäre diese Selektion auch am Ende der Spezifikation möglich, allerdings müsste der Anwender dann auch irrelevante Teile der Baumstrukturen durchlaufen. Neben den Disziplinen und Containerschichten wird an dieser Stelle auch spezifiziert, welche der *SaaS Essentials* zu erfüllen sind.

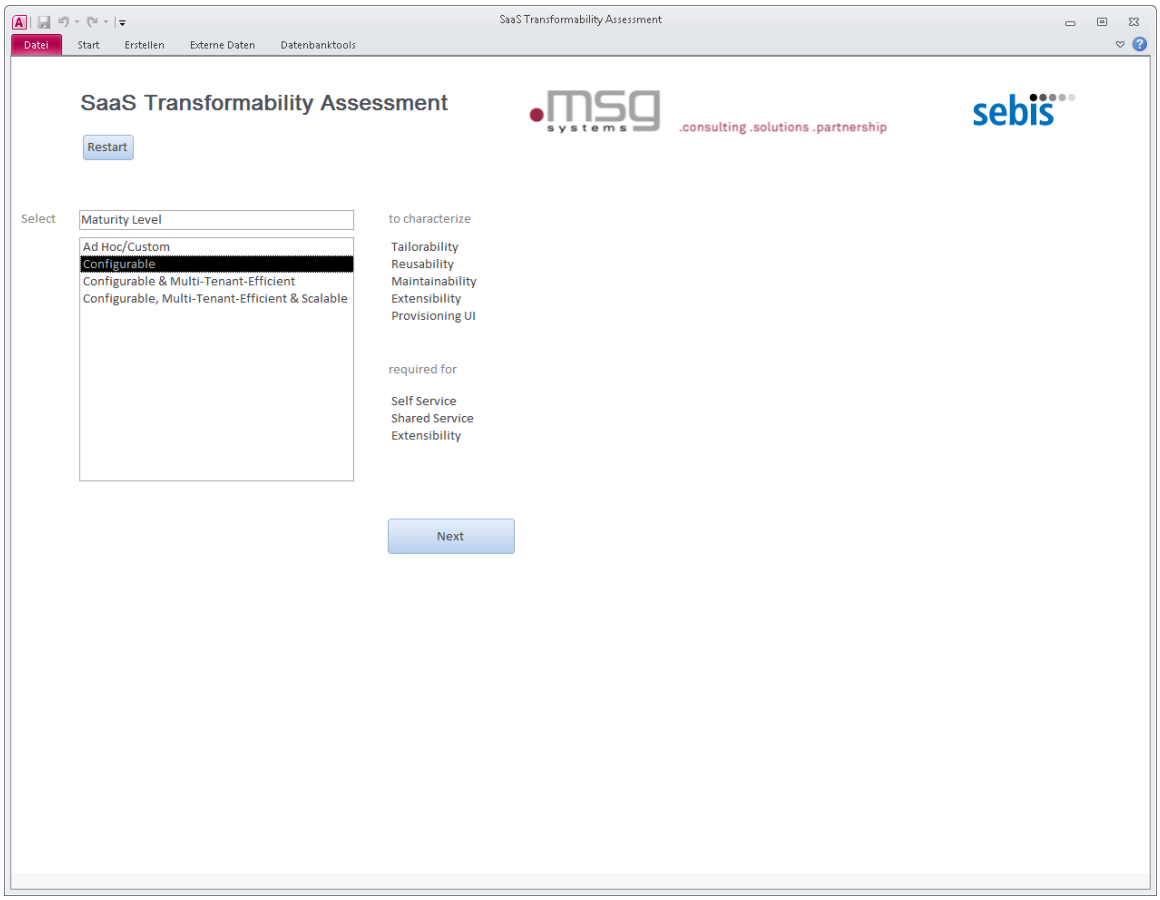


Abbildung 5.7.: Spezifikation von Charakteristiken

Die Vorauswahl, vor allem die der *Essentials*, impliziert eine Reihe von zu erfüllenden speziellen Anforderungen. In diesem Formular müssen deren Merkmale bestimmt werden. So gilt es unter anderem Lastprofile, Metriken zur Nutzungsmessung oder den gewünschten Reifegrad der *SaaS*-Lösung festzulegen. Dabei iteriert das in Abbildung 5.7 dargestellte Formular durch die Filterstruktur bis alle benötigten Charakteristiken selektiert wurden. Neben den Auswahlmöglichkeiten werden dem Anwender auch die *Essentials* sowie die davon abgeleiteten konkreten Anforderungen angezeigt.

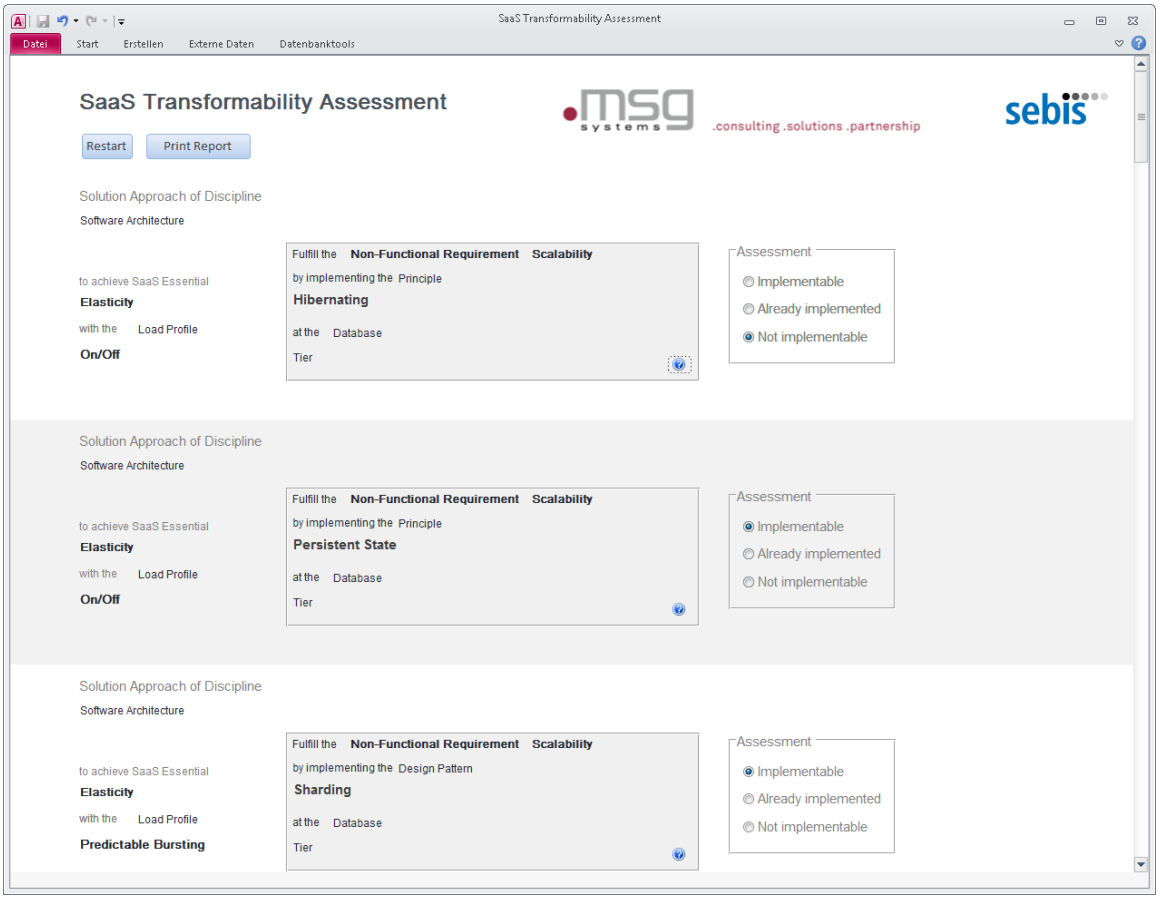
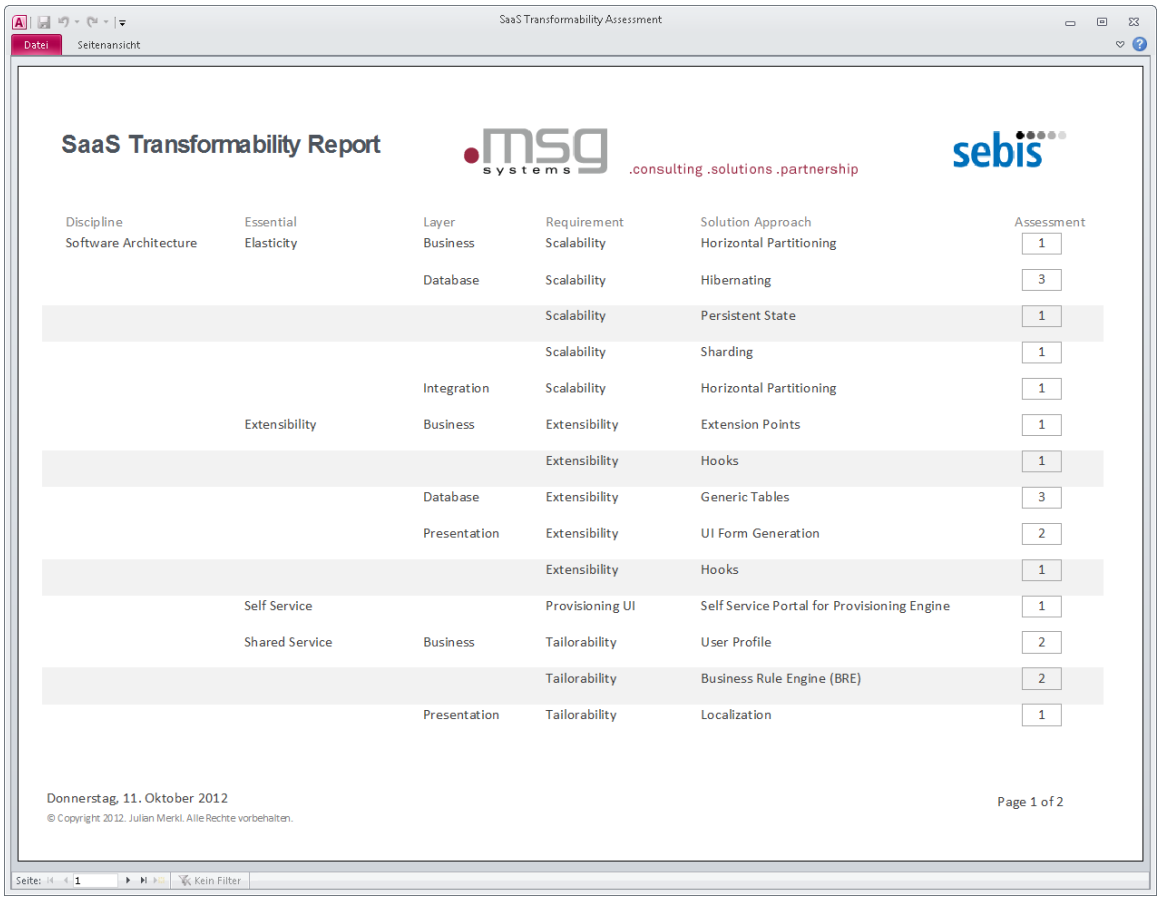


Abbildung 5.8.: Darstellung der Checkliste zur Bewertung

Am Ende der Iteration generiert das *Tool* aus den Informationen im Katalog und den spezifizierten Anforderungen die Handlungsanweisungen und gibt diese in Form einer Checkliste aus. Im Prototyp wurde die Bewertung direkt in die Checkliste integriert. Jede Handlungsanweisung muss einzeln analysiert und hinsichtlich der Machbarkeit bewertet werden. Um Unklarheiten bei den Lösungsansätzen und eine daraus resultierende Fehlinterpretation der Handlungsanweisungen zu vermeiden können im Katalog Beschreibungen hinterlegt werden. In der Checkliste kann es durchaus vorkommen das Lösungsansätze mehrfach aufgenommen werden. Dies ist der Fall wenn ein Lösungsansatz mehrere Anforderung bedient oder er in mehreren Containerschichten angewendet werden kann.

Bei der Ausgabe der Checkliste ist darauf zu achten, dass im Katalog mehrere Ansätze zur Erfüllung eines Kriterium enthalten sein können, das heißt die Methode liefert auch Alternativen zur Implementierung.



Discipline	Essential	Layer	Requirement	Solution Approach	Assessment
Software Architecture	Elasticity	Business	Scalability	Horizontal Partitioning	1
		Database	Scalability	Hibernating	3
			Scalability	Persistent State	1
			Scalability	Sharding	1
		Integration	Scalability	Horizontal Partitioning	1
	Extensibility	Business	Extensibility	Extension Points	1
			Extensibility	Hooks	1
		Database	Extensibility	Generic Tables	3
		Presentation	Extensibility	UI Form Generation	2
			Extensibility	Hooks	1
	Self Service		Provisioning UI	Self Service Portal for Provisioning Engine	1
	Shared Service	Business	Tailorability	User Profile	2
			Tailorability	Business Rule Engine (BRE)	2
		Presentation	Tailorability	Localization	1

Abbildung 5.9.: Druckansicht des Berichts zur Bewertung

Nach der Bewertung hat der Anwender die Möglichkeit einen Bericht zu generieren, der eine vereinfachte Form der Checkliste sowie die Bewertung enthält. Dabei werden die Lösungsansätze sortiert und gruppiert. Zur weiteren Entscheidungsfindung kann so beispielsweise relativ einfach nachvollzogen werden an welcher Stelle wie viele Anforderungen zu erfüllen sind, oder auch welche Alternativen existieren. Der Bericht ist für den Druck optimiert und eignet sich somit auch zur Dokumentation des Bewertungsprozesses.

6. Kritische Reflexion

Im vorletzten Kapitel folgt die kritische Betrachtung der Ergebnisse. Dabei werden die Schwierigkeiten und Problemfelder der Konzeptionsphase wie auch der Implementierung beim Kooperationspartner dargelegt. Zudem werden mögliche Schwachstellen der Methode aufgezeigt.

6.1. Probleme bei Methodenentwicklung und Implementierung

Die Hauptschwierigkeit bei der Methodenentwicklung war es einen gemeinsamen Konsens zur Definition von Cloud Computing und den Kriterien für *SaaS* zu finden. Wie die Vorstudie aus Abschnitt 2.2 gezeigt hat sind in der Theorie die fundamentalen Kriterien zwar bekannt jedoch besteht ausreichend Interpretationsspielraum zur Umsetzung in der Praxis.

Setzt *SaaS* eine Mandantenfähigkeit voraus oder kann die gemeinsame Nutzung der Ressourcen auf andere Ebene implementiert werden? Muss die Anwendung auf einer *Cloud*-Infrastruktur (also *PaaS*) betrieben werden, die selbst alle *Cloud*-Kriterien erfüllt oder kann der Betrieb auch auf einer individuellen Laufzeitumgebung erfolgen?

Ein gemeinsames Verständnis aller Akteure stellt eine Grundsatzvoraussetzung zur erfolgreichen Anwendung der Methode dar. Schließlich sind es die Experten aus Entwicklung und Implementierung und nicht Kundenberater, die den Katalog mit Wissen füllen. Die Qualität der Ergebnisse ist maßgeblich von der Präzision der Lösungsansätze hinsichtlich ihrer Zweckdienlichkeit und der Einordnung in der Matrix abhängig. Die Lösungsansätze müssen den gemeinsam festgelegten Kriterien und Charakteristiken entsprechen und nicht zuletzt müssen auch die Berater im Kundengespräch methodenkonforme Anforderungen spezifizieren um zu einem belastbarem Ergebnis zu kommen.

Auch bei der Entwicklung der Methode und der späteren Implementierung beim Kooperationspartner *msg* musste zunächst ein Konsens gefunden werden. Vor dem Projekt galt die *NIST* Definition als Unternehmensstandard und so musste gerade bei der Durchführung der Workshops oft diskutiert werden, ob ein Lösungsansatz auch den Kriterien von *SaaS* gerecht wird.

Die Kernfrage ist letztendlich, ob es sich bei einer *SaaS*-Implementierung, die nicht alle essentiellen Kriterien erfüllt, überhaupt noch um *SaaS* handelt. Bei der Anwendung der Methode in der Kundenberatung ist nicht immer gegeben, dass die eigene Definition auch der des Kunden entspricht.

Diese sehr individuell zu erörternde Problemstellung war ausschlaggebend für die Entwicklung der Filterstrukturen auf Basis von Bäumen, da so ganze Pfade heraus gefiltert werden können. Essentielle Kriterien lassen sich dadurch auch während der Anwendung der Methode und nicht nur bei der Implementierung und dem Aufbau des Katalogs flexi-

bel interpretieren.

Bei der Durchführung der Workshops zum Aufbau des Katalogs und der Identifikation von Lösungsansätzen konnte ein weiteres Problem ausgemacht werden. Nicht immer können diese direkt benannt werden. Die Experten waren durchaus in der Lage Ansätze zu umschreiben, oder Beispiele zu nennen. Eine Aufbereitung oder Verallgemeinerung der Ergebnisse erfolgte in manchen Fällen jedoch erst im Nachgang. Die Wissensentwicklung stellt also einen weitaus langwierigeren Vorgang dar, als zunächst vermutet wurde.

6.2. Mögliche Schwachstellen der Methode

Weiterhin besteht die Gefahr, dass trotz Aufbereitung und Revision – idealerweise durch eine zentrale Stelle – nicht alle Lösungsansätze in die geforderte Struktur eingepasst werden können. Die Methode wurde anhand der Disziplin Software-Architektur entwickelt. Dennoch stellt die erfolgreiche Evaluation anhand Beispielen aus anderen Disziplinen keinen Garant für eine grenzenlose Anwendung dar.

Obwohl die Struktur der Matrix und die der Filter adaptiert werden kann müssen nach der Implementierung alle Lösungsansätze konfektioniert werden. Nur wenn eine Kategorisierung entlang der Dimensionen und Typisierung erfolgt werden diese zur Bewertung berücksichtigt. Konkret bedeutet das, dass es nicht ausreicht eine Möglichkeit zur Implementierung zu benennen sondern es muss auch mindestens eine Charakteristik gefunden werden. Dieser Umstand erhält zwar die Konsistenz im Katalog, birgt aber die Gefahr das Lösungsansätze beliebig zugeordnet werden um überhaupt berücksichtigt zu werden. Die Wissensstrukturierung ist jedoch absolutes Qualitätsmerkmal.

Neben der Vorenthaltung von relevanten Informationen stellt auch der Aufbau von redundanten Einträgen eine mögliche Schwachstelle dar. Beispielsweise können im Katalog vergleichbare Lösungsansätze mit unterschiedlicher Typisierung aufgenommen werden. Stehen die Typen in einer Generalisierungsbeziehung wäre es durchaus denkbar, dass auch konkrete *Design Patterns* einem übergeordneten Prinzip unterliegen. Demnach ist es nicht ausgeschlossen, dass der Katalog kongruente Lösungsansätze, die sich nur in ihrer Spezialisierung unterscheiden, beinhaltet. Diese Redundanz stellt vermutlich keine Gefahr für die Aussagefähigkeit der Bewertung dar sofern die Typisierung bei der Auswertung beachtet wird.

Ein weiterer kritischer Aspekt ist die Belastbarkeit der Methode unter dem Aspekt der Vollständigkeit des Katalogs. Die Methode liefert nur eine Bewertungsmöglichkeit bezüglich identifizierter Anforderungen und Lösungsansätzen. Ein produktiver Einsatz der Methode, also zum Beispiel zur Beratung in realen Kundenprojekten, ist nur empfehlenswert wenn der Katalog eine hinreichende Reife erlangt hat. Der Katalog muss daher im Vorfeld aufgebaut werden.

Folglich kann bei der Anwendung der Methode kein Anspruch auf Vollständigkeit gestellt werden. Die Methode wurde konzeptioniert um als Hilfsmittel und nicht als Automatismus herangezogen zu werden und Bedarf stets einer menschlichen Beratungsleistung, sowie eine manuelle Verifikation.

7. Zusammenfassung und Ausblick

Diese Arbeit legt den Grundstein für den Ausbau der Methode beim Kooperationspartner *msg*, oder auch in anderen Institutionen. In diesem letzten Kapitel werden die Ergebnisse zusammengefasst. Schließlich folgt noch ein Ausblick auf mögliche, zukünftige Entwicklungen der Methode.

7.1. Zusammenfassung der Ergebnisse

Möchte man die Transformierbarkeit eines *On-Premise*-Anwendungssystems in eine *SaaS*-Lösung untersuchen und bewerten müssen die Rahmenbedingungen zur Portierung der Software von einem Ausgangszustand in einen Zielzustand bekannt sein. Eine solche Machbarkeitsstudie erfordert detaillierte Kenntnis über die vorhandene Software als auch über die Möglichkeiten zur Implementierung eines *Cloud*-Service.

Eine mögliche Komplexitätsreduzierung der Bewertung erfordert das Herausfiltern von Projekt-irrelevanten Anforderungen an den Zielzustand, welcher sich anhand von Lösungsansätzen beschreiben lässt. Somit gilt es durch möglichst genaue Spezifikation irrelevante Lösungsansätze zu eliminieren.

Um eine Spezifikation von Anforderungen und damit die Filterung durchführen zu können, müssen die Mengen der Lösungsansätze sowie der Anforderungen in eine Struktur gebracht werden, die eine Zuordnung ermöglicht. Dazu konnten zwei Datenstrukturen gefunden werden. Lösungsansätze werden in einer mehrdimensionalen Matrix eingeordnet. Die Dimensionen der Matrix entsprechen den Aspekten der Anforderungsspezifikation. Anforderungen und ihre Charakteristiken werden in Bäumen strukturiert, die sich von der Wurzel abwärts spezialisieren. Lösungsansätze werden ausschließlich den Blättern der Bäume zugeordnet. Werden also generelle Anforderungen abgewählt, so fallen auch die Spezialfälle heraus.

Je nach Kontext der Anwendung der Methode sind verschiedene Bewertungsverfahren möglich. Die Bewertung erfolgt jedoch immer anhand von Handlungsanweisungen, die von den Lösungsansätzen und den spezifizierten Eigenschaften abgeleitet werden. Die Methode selbst führt keine eigenständige Bewertung durch, sondern integriert ein zweckdienliches Bewertungsverfahren und stellt ein Hilfsmittel dar.

Schließlich konnte die Methode formalisiert werden, so dass sich die Strukturen zur Filterung und auch die Matrix selbst erweitern lassen.

Nach der Konzeptionsphase erfolgte die Implementierung beim Kooperationspartner. Die Katalogstruktur wurde individuell an die Anforderungen des Unternehmens angepasst. Mit der Durchführung von Workshops konnten zahlreiche Lösungsansätze erarbeitet werden, die eine hinreichende Vollständigkeit zur Validierung der Methode und zum weiteren Ausbau, ermöglichten. Neben diesen Lösungsansätzen kristallisierten sich

auch wichtige Charakteristiken, die zur Spezifikation eines Transformationsprojekts relevant sind, heraus.

Dazu zählen beispielsweise Metriken zur Messung der Nutzung, Lastprofile zur Skalierbarkeit in einer elastischen Infrastruktur oder Reifegrade von Anforderungen wie der gemeinsamen Nutzung von Ressourcen oder der Erweiterbarkeit.

Auch grundlegende Erkenntnisse zur Implementierung einer *SaaS*-Lösung konnten gewonnen werden. *Provisioning*, also die automatisierte Verwaltung von Ressourcen, stellt eine funktionale Anforderung dar, die von mehreren essentiellen Kriterien von *SaaS* gefordert wird. Bei allen Anforderungen muss stets hinterfragt werden, auf welcher Ebene der Architektur die Kriterien erfüllt und wirtschaftlich umgesetzt werden können.

Nicht zuletzt bieten serviceorientierte Architekturen gute Voraussetzungen zur Transformation.

7.2. Ausblick

Zum Ausbau der Methode sollen bei der *msg* Innovationsprojekte durchgeführt werden, mit denen weitere Lösungsansätze gefunden und strukturiert werden sollen. Sicherlich steht auch der Katalog zur Disziplin Software-Architektur erst am Anfang seiner Möglichkeiten. Letztendlich werden es aber Projekte aus der Praxis sein, die reale Bedingungen zur Prüfung der Belastbarkeit der Methode liefern.

Auch die Erweiterung um neue Disziplinen bietet ausreichend Potential zur Weiterentwicklung der Methode. Gerade hier wird sich mit der Zeit zeigen, wie universell die Anwendung erfolgen kann oder ob weitere Möglichkeiten zur Wissensstrukturierung geschaffen werden müssen.

Wie die Methode muss auch die *Tool*-Unterstützung bedarfsgerecht konzipiert und ausgebaut werden. Der im Rahmen dieser Arbeit erstellte Prototyp bietet zwar die Basis-Funktionalitäten dennoch ist Entwicklung eines leistungsstarken Werkzeugs anzustreben. Mögliche Zusatzfunktionen, wie eine grafische Darstellung des Prozesses zur Filterung oder Anforderungsspezifikation wären im Beratungsgespräch hilfreich.

Verbesserte Auswertungsmöglichkeiten nach der absolvierten Bewertung könnten den Prozess zur Entscheidungsfindung erheblich vereinfachen und auch eine Verwaltung und Archivierung von abgeschlossenen Projekten zur Vergleichbarkeit und Dokumentation wäre von Vorteil. Schließlich ist, sobald die Methode in der Praxis angewendet wird, auch die Zentralisierung des Katalogs als *Backend* ein sinnvoller Schritt, so dass alle Anwender auf den gleichen und vor allem aktuellen Katalog zugreifen können.

Abseits der Aufgabenstellung ist auch eine Ausweitung auf andere Service-Modelle des *Cloud Computings* möglich. So könnte die Methode auch zum Aufbau von *PaaS* und *IaaS*-Angeboten herangezogen werden. Die Voraussetzungen sind dabei mit dem umfangreichen Ebenenmodell der Implementierung bei der *msg* bereits heute geschaffen. Die Sammlung der Lösungsansätze bietet zudem auch eine gute Einstiegspunkt für vollständige Neuentwicklungen. In diesem Fall wäre jedoch nicht die Transformierbarkeit sondern, wenn überhaupt, die Komplexität oder der Aufwand zu bewerten.

Anhang

A. Zielsetzung der msg

Gesprächsnotiz

Teilnehmer: Peter Huber (Experte Java und Enterprise Architekturen & Security)
Jürgen Biebl (Leiter CoC Projektinstruktur)
Julian Merkl

Datum: 09.08.2012

Zielsetzung zur Methode

Die Methode

- dient als Hilfsmittel zur Entscheidungsfindung (über Transformation).
- dient zur Dokumentation des Projekts.

Die Methode soll

- nachvollziehbar
- strukturiert
- wiederholbar (und damit verlässlich - Vertrauen)

sein.

Ergebnis der Methode

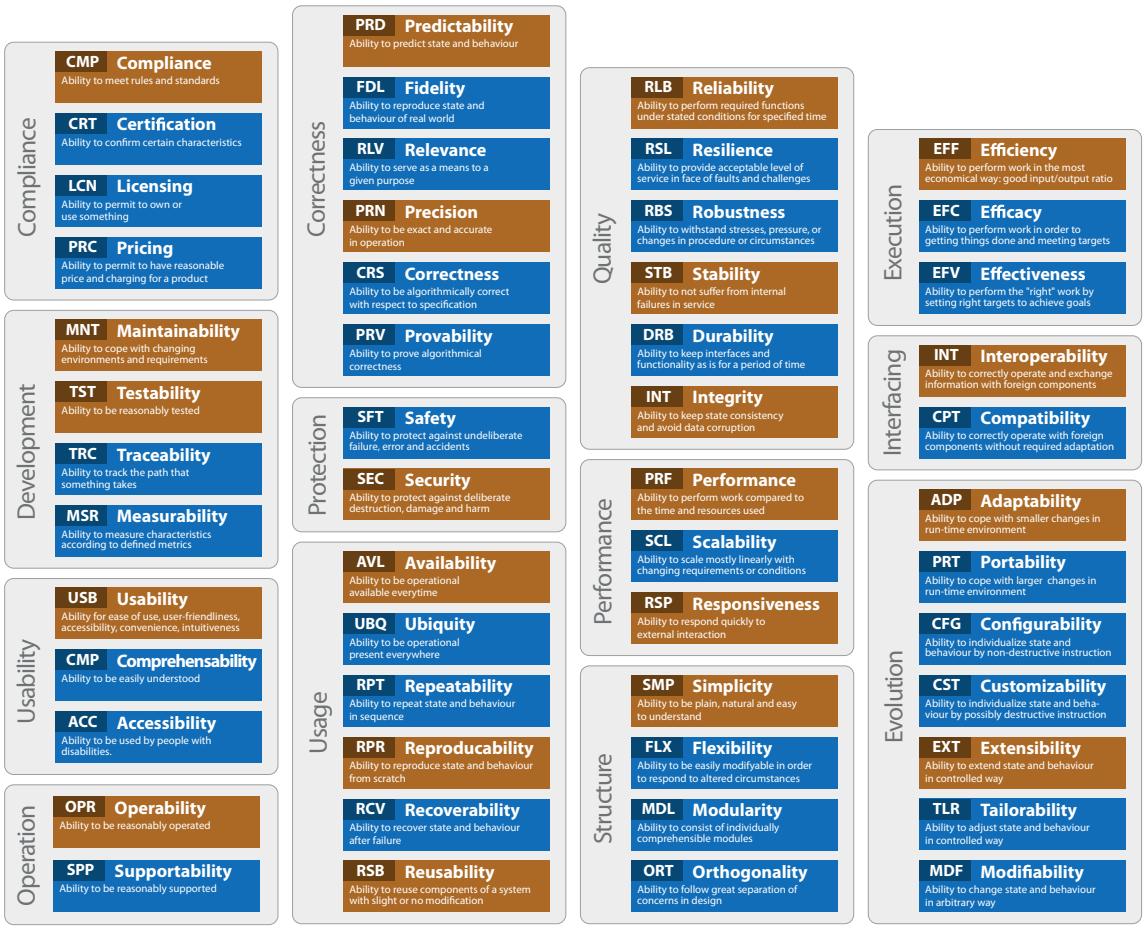
Identifikation relevanter Aspekte (Teilmenge) aus einem Katalog (Zielmenge) von Lösungsmöglichkeiten (Patterns / Prinzipien)

Tool-Unterstützung

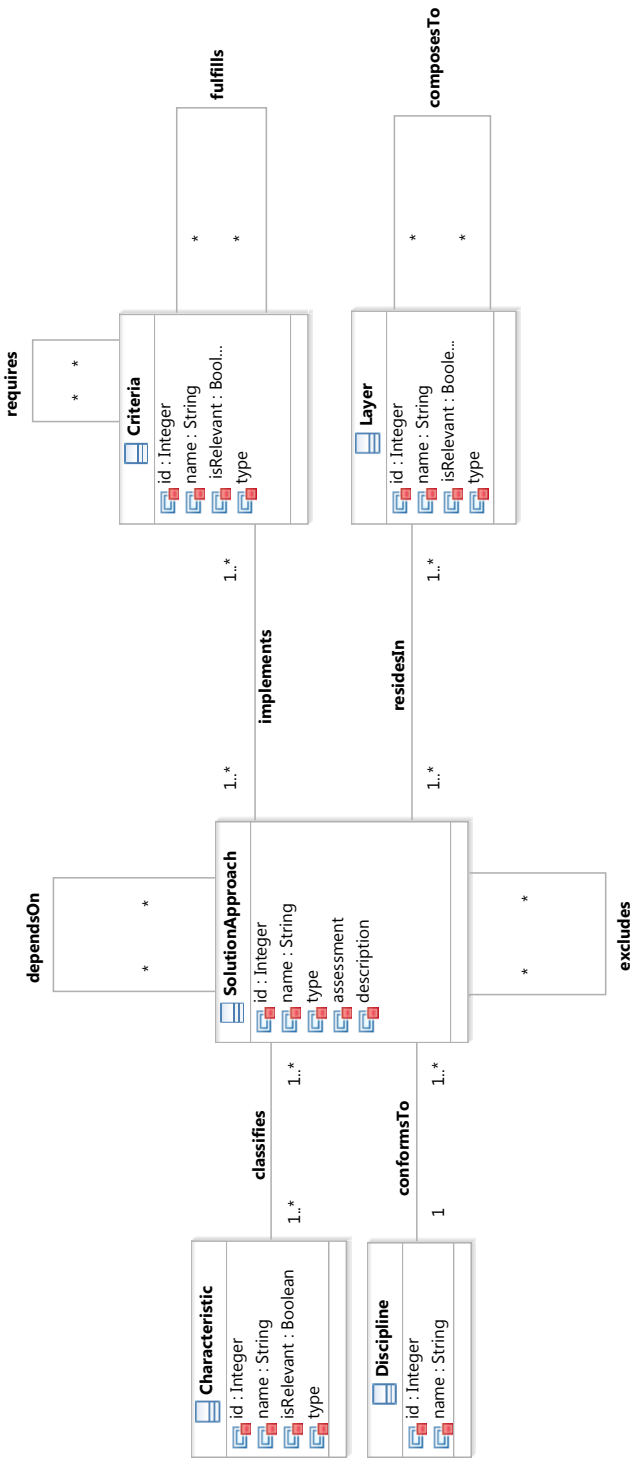
- Filterfunktionalität
- Bewertungsmöglichkeit
- Abbildung von Abhängigkeiten
- Selektion der Filter als quasi Navigationsstruktur durch den Cloud-Jungle

Der Katalog und damit das Tool sollen hinreichend vollständig sein, um die Methode exemplarisch an einem oder zwei Beispielprojekten validieren zu können. Diese hinreichende Vollständigkeit ergibt sich aus dem Experten-Wissen der msg Mitarbeiter.

B. Übersicht nicht-funktionaler Anforderungen



C. Konzeptuelles Klassendiagramm der msg Implementierung



D. Ergebnisse der Workshops

Workshop 1 von 5

Teilnehmer: Peter Huber (Experte Java und Enterprise Architekturen & Security)
Jürgen Biebl (Leiter CoC Projektinstruktur)
Julian Merkl

Datum: 10.08.2012

Ergebnisse:

- Scalability als NFR für Elasticity
  - Primäre Charakteristik zur Auswahl an Lösungsansätzen = Lastprofil
  - Lastprofile (Beispiele)
    - On/Off
    - Growing Fast
    - Unpredictable Bursting
    - Predictable Bursting
  - Parameter von Lastprofilen
    - Initial load
    - Ramp-up time
    - Minimum final load
    - Maximum final load
- Patterns für Scalability
  - On/Off
    - On-hold / Cold Standby
    - Hot Standby (schlecht bei Pay per use)
    - Fresh Deployment
    - Persistent State / Cache
  - Growing Fast
    - Clustering
    - Stateless => Scale out
    - Shared Nothing
    - BASE vs. ACID
    - Master/Slave (DB)
    - CQRS (Command Query Responsibility Segregation)
  - Unpredictable Bursting
    - Automated/Intelligent Scale-out
    - Hot Standby

        - Alle von Growing Fast
    - Predictable Bursting
        - Cold Standby
        - Scale up

Workshop 2 von 5

Teilnehmer:	Erwin Wacha (Experte für Frontend-Entwicklung) Peter Huber (Experte Java und Enterprise Architekturen & Security) Julian Merkl
Datum:	22.08.2012

Ergebnisse:

Welchen der SaaS Essentials sehen Sie in Ihrem Spezialgebiet von besonderer Bedeutung?

- IP Access
- Shared Service (Multi-Tenancy)

Welche Architektur-Prinzipien, Design Patterns oder Best Practices können herangezogen werden um diese SaaS Essentials zu erfüllen, d.h. welche Ansätze gibt es die daraus resultierenden NFRs zu erfüllen?

- leider keine direkte Benennung der Patterns möglich
- Ubiquity (im Kontext Cloud Computing)
    - erfordert oft ein Multi-Channel Frontend, d.h. mehrere Nutzergruppen (gerade in Verbindung mit Multi-Tenancy), unterschiedliche Clients (Plattformen) und mehrere UIs (API, Smart /Rich Clients, Thin Clients, Mobile, RIA)
      ➔ (Web)-Service orientation / SOA, z.B. mit REST, Jason
- Configurability als NFR für Shared Service (Multi-Tenancy)
    - Localization
    - Nationalization
    - Trennung von GUI und Daten
- Security als NFR für Shared Service (Multi-Tenancy)
    - Federation
    - Single-Sign-On
    - AAA (Authentifizierung (engl. authentication), Autorisierung (engl. authorization) und Abrechnung (engl. accounting))

Zusätzliche Antworten:

- Measurability (im Frontend)
    - Zeit: Sitzungsdauer messen, bei einer Stateless-Architecture ist schwierig, letztendlich muss es aber wohl über CSPD erfolgen

    - Ressourcen: API der Container verwenden ➔ Auswahl der Container von Bedeutung
- Scalability (im Frontend)
    - Content Delivery Networks (wobei das keine Lösung der Disziplin Software Architektur darstellt)
    - Stateless Architecture erfordert CSPD

Workshop 3 von 5

Teilnehmer: Hermann Schmidt (Experte für Software-Architektur, SOA)
Ralf Engelschall (Abteilungsleiter CoC Application Architecture)
Peter Huber (Experte Java und Enterprise Architekturen & Security)
Julian Merkl

Datum: 29.08.2012

Ergebnisse:

- Multi Tenancy:
  - Tenant Separation
    - VM oder Server per Tenant != Single RTE
    - DB per Tenant (SQL Template Engines)
    - Schema per Tenant
    - Shared Tables
  - GUI
    - Theming (& Branding)
    - Profiling
    - Localization
    - Multilingualization
  - Business Logic
    - Script-based Model and Data Maintenance (Scripting Engines)
    - Business Rule Engine (BRE)
    - (Fluid Kernel Scripting)
- Provisioning (wird in PaaS Layer realisiert, in SA nur bedingt (z.B Config Updates)
  - Instance Deployment
  - Instance Preparation (= Setup and initial Configuration)
- Extensiblity & Customization
  - Use Domain-specific languages (z.B. XAML)
  - Relay on Standards
  - Open/Closed Principle

Workshop 4 von 5

Teilnehmer: Peter Huber (Experte Java und Enterprise Architekturen & Security)
Julian Merkl

Datum: 23.08.2012

Ergebnisse:

- Measurability als NFR für Pay per use
    - Charakteristiken für Measurability
        - Zeit (z.B. Sitzungsdauer)
        - Ressourcen (wie CPU, Speicher, Traffic)
        - Fachliche Größen (Anzahl, Summe von Transaktionen, DB Record)
    - Lösungsansätze
        - Pipes and Filter (z.B. für Traffic)
        - Monitoring background processes
        - Track front-end time
            - Upstream Web Server Time
            - Request Queueing
            - Middleware Execution Time
        - Method Tracing
        - Performance Counters von Containern verwenden (APIs je nach Tier)

Workshop 5 von 5

Teilnehmer:	Carol Gutzeit (Leiter CoC Enterprise Architecture) Ralf Engelschall (Abteilungsleiter CoC Application Architecture) Julian Merkl
Datum:	06.09.2012

Ergebnisse:

- Komplexität der Architektur (bzw. Anwendung) steigt mit dem Grad der Mandantenfähigkeit und dem Grad der Indiviualisierbarkeit (Konfigurierbarkeit und Erweiterbarkeit)
- Extensibility und Configurability als NFR für Extendable Data Model
    - o Hooking / Hooks
    - o Extension Points / Plugins / Service Provider Interfaces
    - o Generic Tables i.V. mit Form Generations / Business Rule Engines, …
- Measurability als NFR für Pay per Use:
    - o Business Intelligence Funktionalität zur Messung fachlicher Größen
    - o Querschnittskomponente für Plan Management, z.B. für Accounting und Authorization (z.B. aspect-oriented programming, AOP)
- Provisioning
    - o Ressource Management ➔ kann Routing erfordern
- Multi-Tenancy i.V. mit Realtime Self-Service
    - o Zero-Downtime erfordert Online Configuration Management / Updates
        - ⇨ Konfigurations- und Versionsmanagement sehr wichtig
        - ⇨ Update einzelner Komponenten und Orchestration von Konfiguration bzw. Versionen

E. Initialer Katalog

Discipline	Type	Essential	Requirement	Type	Characteristic	Parameter	Solution Approach
System Integration	Essential	Elasticity	Scalability	Non-Functional	Load Profile	Growing Fast	Clustering
System Integration	Essential	Elasticity	Scalability	Non-Functional	Load Profile	Growing Fast	Master-Slave
System Integration	Essential	Elasticity	Scalability	Non-Functional	Load Profile	On/Off	Cold Standby
System Integration	Essential	Elasticity	Scalability	Non-Functional	Load Profile	On/Off	Hibernating
System Integration	Essential	Elasticity	Scalability	Non-Functional	Load Profile	Predictable Bursting	Cold Standby
System Integration	Essential	Elasticity	Scalability	Non-Functional	Load Profile	Predictable Bursting	Scale up
System Integration	Essential	Elasticity	Scalability	Non-Functional	Load Profile	Unpredictable Bursting	Clustering
System Integration	Essential	Elasticity	Scalability	Non-Functional	Load Profile	Unpredictable Bursting	Hot Standby
System Integration	Essential	Elasticity	Scalability	Non-Functional	Load Profile	Unpredictable Bursting	Master-Slave
System Integration	Essential	Elasticity	Scalability	Non-Functional	Load Profile	Growing Fast	Content Delivery Network
System Integration	Essential	Elasticity	Scalability	Non-Functional	Load Profile	On/Off	Content Delivery Network
System Integration	Essential	Elasticity	Scalability	Non-Functional	Load Profile	Predictable Bursting	Content Delivery Network
System Integration	Essential	Elasticity	Scalability	Non-Functional	Load Profile	Unpredictable Bursting	Content Delivery Network
Software Architecture	Essential	Elasticity	Scalability	Non-Functional	Load Profile	Growing Fast	BASE
Software Architecture	Essential	Elasticity	Scalability	Non-Functional	Load Profile	Growing Fast	CQRS
Software Architecture	Essential	Elasticity	Scalability	Non-Functional	Load Profile	Growing Fast	Shared Nothing Architecture
Software Architecture	Essential	Elasticity	Scalability	Non-Functional	Load Profile	Growing Fast	Stateless Clustering
Software Architecture	Essential	Elasticity	Scalability	Non-Functional	Load Profile	On/Off	Persistent State
Software Architecture	Essential	Elasticity	Scalability	Non-Functional	Load Profile	Unpredictable Bursting	BASE
Software Architecture	Essential	Elasticity	Scalability	Non-Functional	Load Profile	Unpredictable Bursting	CQRS
Software Architecture	Essential	Elasticity	Scalability	Non-Functional	Load Profile	Unpredictable Bursting	Shared Nothing Architecture
Software Architecture	Essential	Elasticity	Scalability	Non-Functional	Load Profile	Unpredictable Bursting	Stateless Clustering
Software Architecture	Essential	Extensibility	Extensibility	Non-Functional	Maturity Level	Ad Hoc/Custom	Open/Closed
Software Architecture	Essential	Extensibility	Extensibility	Non-Functional	Maturity Level	Configurable	Hooks
Software Architecture	Essential	Extensibility	Extensibility	Non-Functional	Maturity Level	Configurable	Extension Points (Plugins, SPIs)
Software Architecture	Essential	Extensibility	Extensibility	Non-Functional	Maturity Level	Configurable	Generic Tables
Software Architecture	Essential	Extensibility	Extensibility	Non-Functional	Maturity Level	Configurable	UI Form Generation
Compliance	Essential	Industry Standards	Security	Guideline	Country	Germany	BSI Grundschutz
Compliance	Essential	Industry Standards	Security	Guideline	Industry	USA	PCI DSS
Compliance	Essential	Industry Standards	Security	Guideline	Industry	Health	HIPAA
System Integration	Essential	IP Access	Ubiquity	Non-Functional	Audience	Worldwide	Content Delivery Network
Software Architecture	Essential	IP Access	Multi-Channel Frontend	Functional	Interface	API	REST
Software Architecture	Essential	IP Access	Multi-Channel Frontend	Functional	Interface	BUI	RIA
Software Architecture	Essential	IP Access	Multi-Channel Frontend	Functional	Interface	GUI	
Software Architecture	Essential	IP Access	Multi-Channel Frontend	Functional	Device	Mobile	

Discipline	Type	Essential	Requirement	Type	Characteristic	Parameter	Solution Approach
Software Architecture	Essential	IP Access	Multi-Channel Frontend	Functional	Device	Smart Client	
Software Architecture	Essential	IP Access	Multi-Channel Frontend	Functional	Device	Thin Client	
Software Architecture	Essential	IP Access	Ubiquity	Non-Functional	Connection	Offline	Occasionally Connected Computing (OCC)
Compliance	Essential	Legal Requirements	Privacy	Law	Industry		
Compliance	Essential	Legal Requirements	Privacy	Law	Country	Germany	Bundesdatenschutzgesetz
System Integration	Essential	Provisioning	Deployment	Functional	Artefact	Instance	Cloning
System Integration	Essential	Provisioning	Deployment	Functional	Artefact	Instance	Fresh Install
Software Architecture	Essential	Provisioning	Deployment	Functional	Artefact	Service	Enterprise Service Bus
Software Architecture	Essential	Provisioning	Deployment	Functional	Artefact	Tenant	SQL Template Engine
Software Architecture	Essential	Provisioning	Plan Management	Functional	Artefact	User	Accounting
Software Architecture	Essential	Provisioning	Ressource Management	Functional	Artefact	User	Accounting
Software Architecture	Essential	Service Measuring	Measurability	Non-Functional	Metric	Data Elements	Directory Service Query
Software Architecture	Essential	Service Measuring	Measurability	Non-Functional	Metric	Data Elements	Method Tracing
Software Architecture	Essential	Service Measuring	Measurability	Non-Functional	Metric	Data Elements	Database Query
Software Architecture	Essential	Service Measuring	Measurability	Non-Functional	Metric	Data Elements	Business Analytics
Software Architecture	Essential	Service Measuring	Measurability	Non-Functional	Metric	Ressource	Monitoring Background Processes
Software Architecture	Essential	Service Measuring	Measurability	Non-Functional	Metric	Ressource	Performance Counter Container API
Software Architecture	Essential	Service Measuring	Measurability	Non-Functional	Metric	Ressource	Pipes and Filter
Software Architecture	Essential	Service Measuring	Measurability	Non-Functional	Metric	Time	Middleware Execution Time
Software Architecture	Essential	Service Measuring	Measurability	Non-Functional	Metric	Time	Request Queueing
Software Architecture	Essential	Service Measuring	Measurability	Non-Functional	Metric	Time	Upstream Web Server Time
Software Architecture	Essential	Shared Service	Interoperability	Non-Functional	Artefact	User	Federation
Software Architecture	Essential	Shared Service	Security	Non-Functional	Artefact	User	Authentication
Software Architecture	Essential	Shared Service	Security	Non-Functional	Artefact	User	Authorization
Software Architecture	Essential	Shared Service	Reusability	Non-Functional	Maturity Level	Ad Hoc/Custom	Server per Tenant
Software Architecture	Essential	Shared Service	Reusability	Non-Functional	Maturity Level	Ad Hoc/Custom	VM per Tenant
Software Architecture	Essential	Shared Service	Reusability	Non-Functional	Maturity Level	Ad Hoc/Custom	DB per Tenant
Software Architecture	Essential	Shared Service	Reusability	Non-Functional	Maturity Level	Configurable	SOA
Software Architecture	Essential	Shared Service	Reusability	Non-Functional	Maturity Level	Multi-Tenant-Efficient	Schemas per Tenant
Software Architecture	Essential	Shared Service	Reusability	Non-Functional	Maturity Level	Multi-Tenant-Efficient	Shared Tables
Software Architecture	Essential	Shared Service	Reusability	Non-Functional	Maturity Level	Scalable	Tenant Load Balancer
Software Architecture	Essential	Shared Service	Tailorability	Non-Functional	Maturity Level	Configurable	Localization
Software Architecture	Essential	Shared Service	Tailorability	Non-Functional	Maturity Level	Configurable	Branding
Software Architecture	Essential	Shared Service	Tailorability	Non-Functional	Maturity Level	Configurable	Profiling
Software Architecture	Essential	Shared Service	Tailorability	Non-Functional	Maturity Level	Configurable	Theming

Discipline	Type	Essential	Requirement	Type	Characteristic	Parameter	Solution Approach
Software Architecture	Essential	Shared Service	Tailorability	Non-Functional	Maturity Level	Multi-Tenant-Efficient	Business Rule Engine (BRE)
Software Architecture	Essential	Shared Service	Tailorability	Non-Functional	Maturity Level	Multi-Tenant-Efficient	Scripting Engine
Software Architecture	Essential	Shared Service	Tailorability	Non-Functional	Maturity Level	Multi-Tenant-Efficient	Fluid Kernel Scripting
Software Architecture	Essential	Shared Service	Tailorability	Non-Functional	Maturity Level	Scalable	Tenant Load Balancer
Software Architecture	Essential	Shared Service	Maintainability	Non-Functional	Maturity Level	Multi-Tenant-Efficient	Configuration Management
Software Architecture	Essential	Shared Service	Maintainability	Non-Functional	Maturity Level	Multi-Tenant-Efficient	Version Management
Software Architecture							Client-Side Persistent Data (CSPD)

Literaturverzeichnis

[AFG+09] ARMBRUST, Michael ; FOX, Armando ; GRIFFITH, Rean ; JOSEPH, Anthony D. ; KATZ, Randy H. ; KONWINSKI, Andrew ; LEE, Gunho ; PATTERSON, David A. ; RABKIN, Ariel ; STOICA, Ion ; ZAHARIA, Matei: Above the Clouds: A Berkeley View of Cloud Computing / EECS Department, University of California, Berkeley. Version: Feb 2009. http://www.eecs.berkeley.edu/Pubs/TechRpts/2009/EECS-2009-28.html. 2009 (UCB/EECS-2009-28). – Forschungsbericht

[AJ99] ALLWEYER, T. ; JOST, W.: Geschäftsprozessmanagement und Knowledge Management – Ein integrierter Lösungsansatz. In: *E-Business und Knowledge Management – Neue Dimensionen für den Unternehmenserfolg*. Heidelberg : Scheer, A.-W., 1999

[ama12] *History & Timeline*. Website, 2012. – http://phx.corporate-ir.net/phoenix.zhtml?c=176060&p=irol-corporateTimeline; zugegriffen am 09.10.2012.

[Amb99] AMBERG, Michael: *Prozeßorientierte betriebliche Informationssysteme - Methoden, Vorgehen und Werkzeuge zu ihrer effizienten Entwicklung*. Berlin, u.a. : Springer, 1999

[Ang] ANGERMEIER, Georg: *Machbarkeit*. Website, . – http://www.projektmagazin.de/glossarterm/machbarkeit; zugegriffen am 12.10.2012.

[Bab11] BABCOCK, Charles: *Oracle Takes Plunge Into Public Cloud Computing*. Website, 2011. – http://www.informationweek.com/cloud-computing/infrastructure/oracle-takes-plunge-into-public-cloud-co/231900110; zugegriffen am 09.10.2012.

[Ban11] BANERJEE, Udayan: *Cloud Computing – Important events till 2010*. Website, 2011. – http://setandbma.wordpress.com/2011/03/08/cloud-computing-important-events-till-2010/; zugegriffen am 09.10.2012.

[bit] *Leitfaden für SaaS-Anbieter*. Leitfaden, . – http://www.bitkom.org/files/documents/Leitfaden_SaaS_20090310_web_neu(1).pdf; zugegriffen am 09.10.2012.

[BLB+00] BENNETT, K. ; LAYZELL, P. ; BUDGEN, D. ; BRERETON, P. ; MACAULAY, L. ; MUNRO, M.: Service-based software: the future for flexible software. In:

Software Engineering Conference, 2000. APSEC 2000. Proceedings. Seventh Asia-Pacific, 2000, S. 214 –221

[BLRK10] BÖHM, M. ; LEIMEISTER, S. ; RIEDL, C. ; KRCMAR, H.: Cloud Computing and Computing Evolution. In: MURUGESAN, S. (Hrsg.): *Cloud Computing Technologies Business Models Opportunities and Challenges*, CRC Press, 2010

[bsia] *Cloud Computing*. Website, . – https://www.bsi.bund.de/DE/Themen/CloudComputing/CloudComputing_node.html; zugegriffen am 09.10.2012.

[bsib] *Cloud Computing Grundlagen*. Website, . – https://www.bsi.bund.de/DE/Themen/CloudComputing/Grundlagen/Grundlagen_node.html; zugegriffen am 09.10.2012.

[CC06] CHONG, Frederick ; CARRARO, Gianpaolo: *Architecture Strategies for Catching the Long Tail*. Website, 2006. – http://msdn.microsoft.com/en-us/library/aa479069.aspx; zugegriffen am 09.10.2012.

[com01] *Giga: Der ASP-Markt löst sich in Luft auf*. Website, 2001. – http://www.computerwoche.de/nachrichtenarchiv/522867/; zugegriffen am 09.10.2012.

[DM09] DI MAIO, Andrea: *U.S. Federal Definition of Cloud Computing: Handle With Care*. Report, 2009

[Eil] EILERS, Markus: *Architekturpatterns für Cloud-Anwendungen*. Website, . – http://www.andrena.de/Entwicklertag/2010/Downloads/Conference-Day/ArchitectForTheCloud.pdf; zugegriffen am 13.10.2012.

[eni11] *Governmental Cloud in the EU - New Agency Report*. Website, 2011. – http://www.enisa.europa.eu/media/press-releases/governmental-cloud-in-the-eu-new-agency-report; zugegriffen am 09.10.2012.

[Erl05] ERL, Thomas: *Service-Oriented Architecture - Concepts, Technology, and Design*. Upper Saddle River, NJ : Prentice Hall PTR, 2005

[Eva10] EVANS, Bob: *Global CIO: Oracle's Larry Ellison Embraces Cloud Computing's 'Idiocy'*. Website, 2010. – http://www.informationweek.com/global-cio/interviews/global-cio-oracles-larry-ellison-embrace/225200363; zugegriffen am 09.10.2012.

[Fal11] FALKNER, Jürgen: *Cloud Standards und Zertifizierungen - Bedarf und Wirklichkeit*. Website, 2011. – http://wiki.iao.fraunhofer.de/index.php/Folien:_Cloud_Standards_und_Zertifizierungen; zugegriffen am 09.10.2012.

[for08] *SaaS Maturity Model according to Forrester*. Website, 2008. – http://blogs.msdn.com/b/architectsrule/archive/2008/08/18/saas-maturity-model-according-to-forrester.aspx; zugegriffen am 09.10.2012.

[for11] *How Cloud Computing is Fueling the Next Startup Boom*. Website, 2011. – http://www.forbes.com/sites/joemckendrick/2011/11/01/cloud-computing-is-fuel-for-the-next-entrepreneurial-boom/; zugegriffen am 10.10.2012.

[FS95] FERSTL, Otto ; SINZ, Elmar J.: Der Ansatz des Semantischen Objektmodells (SOM) zur Modellierung von Geschäftsprozessen. In: *Wirtschaftsinformatik 37* (1995), Nr. 3, S. 209–220

[gara] *Software As A Service (SaaS)*. Website, . – http://www.gartner.com/it-glossary/software-as-a-service-saas/; zugegriffen am 09.10.2012.

[garb] *Studie: IT-Outsourcing in Deutschland – Dynamisches Wachstum trotz anhaltender Kostenkontrolle*. Website, . – http://www.gartner.de/fokus/061042_ou.html; zugegriffen am 09.10.2012.

[gar09] *Gartner Highlights Five Attributes of Cloud Computing*. Website, 2009. – http://www.gartner.com/it/page.jsp?id=1035013; zugegriffen am 09.10.2012.

[Hü10] HÜLSBÖMER, Simon: *Gartner stellt neuen Hype Cycle vor*. Website, 2010. – http://www.computerwoche.de/software/bi-ecm/2520636/; zugegriffen am 09.10.2012.

[Hau11] HAUPTER, Ralph: *Cloud Computing ist in den Unternehmen angekommen – das Potenzial ist damit aber noch lange nicht ausgeschöpft*. Website, 2011. – http://www.computerwoche.de/cloud-expertenrat/2011/05/03/cloud-computing-ist-in-den-unternehmen-angekommen; zugegriffen am 09.10.2012.

[hei03] *Forrester Research übernimmt Giga Information Group*. Website, 2003. – http://www.heise.de/newsticker/meldung/Forrester-Research-uebernimmt-Giga-Information-Group-73337.html; zugegriffen am 09.10.2012.

[HN05] HANSEN, Hans R. ; NEUMANN, Gustaf: *Wirtschaftsinformatik. 1. Bd., Grundlagen betrieblicher Informationsverarbeitung*. 9., völlig neubearb. und erw. Aufl. Stuttgart : Lucius & Lucius, 2005

[ibm] *Defining a framework for cloud adoption*. Whitepaper, . – http://public.dhe.ibm.com/common/ssi/ecm/en/ciw03067usen/CIW03067USEN.PDF; zugegriffen am 10.10.2012.

[idca] *Defining Cloud Services – an IDC update*. Website, . – http://blogs.idc.com/ie/?p=422; zugegriffen am 09.10.2012.

[idcb] *Software as a Service (SaaS) Definition*. Website, . – http://www.idc.com/2010st/saas.html; zugegriffen am 09.10.2012.

[kpm12] *Cloud-Monitor 2012*. Studie, 2012. – http://www.kpmg.de/docs/cloud-monitor-20120529.pdf; zugegriffen am 09.10.2012.

[Kra12] KRAUS, Matthias: *Cloud Computing und Consumerization of IT in Deutschland 2012*. Whitepaper, 2012. – http://download.microsoft.com/download/2/E/0/2E0E6536-BD57-4975-9E41-53DE4645A824/IDC_Whitepaper_CloudComputing_CoIT.pdf; zugegriffen am 09.10.2012.

[KS93] KRCMAR, H. ; STRASBURGER, H.: *Informationsmanagement und Informationssystem-Architekturen - Vorteile und Risiken von Client-Server-Architekturen aus der Sicht des Informationsmanagement,*. Halbergmoos : Angewandte-Informations-Technik-Verl., 1993

[Lev07] LEVINSON, Meridith: *Software as a Service (SaaS) Definition and Solutions*. Website, 2007. – http://www.cio.com/article/109704/Software_as_a_Service_SaaS_Definition_and_Solutions; zugegriffen am 09.10.2012.

[LMHL06] LLORENTE, I.M. ; MONTERO, R.S. ; HUEDO, E. ; LEAL, K.: A Grid Infrastructure for Utility Computing. In: *Enabling Technologies: Infrastructure for Collaborative Enterprises, 2006. WETICE '06. 15th IEEE International Workshops on*, 2006. – ISSN 1524–4547, S. 163 –168

[LMR+] LIEBING, Andreas ; MEINEN, Peter ; RIED, Stefan ; SCHRÖDER, Stefand ; VOLLMAR, Friedrich ; ZIEGLER, Stephan: *SaaS – Checkliste für ISVs*. Website, . – http://www.cloud-practice.de/know-how/saas-%E2%80%93-checkliste-fuer-isvs; zugegriffen am 11.10.2012.

[MG] MELL, Peter ; GRANCE, Tomothy: *The NIST Definition of Cloud Computing*. Report, . – http://csrc.nist.gov/publications/nistpubs/800-145/SP800-145.pdf; zugegriffen am 09.10.2012.

[MPR+] MÜNZL, Gerhard ; PRZYWARA, Bernhard ; RETI, Martin ; SCHÄFER, Jörg ; SONDERMANN, Karin ; WEBER, Mathias ; WILKER, Andreas: *Cloud Computing - Evolution in der Technik, Revolution im Business*. Leitfaden, . – http://www.bitkom.org/files/documents/BITKOM-Leitfaden-CloudComputing_Web.pdf; zugegriffen am 10.10.2012.

[msd] *Microsoft Provisioning System Component Overview*. Website, . – http://technet.microsoft.com/en-us/library/cc526764.aspx; zugegriffen am 09.10.2012.

[msg] *Technologiekompetenz für passgenaue Lösungen*. Website, . – http://www.msg-systems.com/unt_technologiekomp.0.html; zugegriffen am 10.10.2012.

[Pau10] PAULUS, Sachar: *Kennzeichen S(icherheit): Cloud-readiness - Auswirkungen für Softwareentwickler*. Website, 2010. – http://heise.de/-926000; zugegriffen am 09.10.2012.

[Rep12] REPPNER, Markus: *Business By Design 4.0: SAP fängt noch einmal von vorne an*. Website, 2012. – http://www.zdnet.de/88116206/noch-einmal-von-vorne-business-by-design-version-4-0/; zugegriffen am 09.10.2012.

[RN] RASMUSSEN, Neil ; NILES, Suzanne: *Rechenzentrumsprojekte: Systemplanung*. Whitepaper, . – http://www.apcmedia.com/salestools/SNIS-6QJGS2_R1_DE.pdf; zugegriffen am 09.10.2012.

[SH04] STAHLKNECHT, Peter ; HASENKAMP, Ulrich: *Einführung in die Wirtschaftsinformatik*. 11., überarb. und aktualisierte Aufl. Berlin, u.a. : Springer, 2004

[SS06] SCHMIDT, Eric ; SULLIVAN, Danny: Conversation with Eric Schmidt hosted by Danny Sullivan, 2006. – http://www.google.com/press/podium/ses2006.html; zugegriffen am 09.10.2012.

[Tap09] TAPSCOTT, Don (Hrsg.): *Grown up digital: how the net generation is changing your world*. New York : McGraw-Hill, 2009

[vmw] *Cloud Readiness Self-Assessment*. Website, . – http://getcloudready.vmware.com/crsa/; zugegriffen am 10.10.2012.

[VRMCL09] VAQUERO, Luis ; RODERO-MERINO, Luis ; CACERES, Juan ; LINDNER, Maik: A Break in the Clouds: Towards a Cloud Definition. In: *ACM SIGCOMM Computer Communication Review* Bd. 39, 2009

[Wil08] WILLIS, John: *Who Coined The Phrase Cloud Computing?* Website, 2008. – http://www.johnmwillis.com/cloud-computing/who-coined-the-phrase-cloud-computing/; zugegriffen am 09.10.2012.

[ZCZH10] ZHANG, Shuai ; CHEN, Xuebin ; ZHANG, Shufen ; HUO, Xiuzhen: The comparison between cloud computing and grid computing. In: *Computer Application and System Modeling (ICCASM), 2010 International Conference on* Bd. 11, 2010, S. V11–72 –V11–75

[Zei12] ZEITLER, Nicolas: *IBM: Outsourcing-Trends bis 2017*. Website, 2012. – http://www.cio.de/knowledgecenter/outsourcing/2877932/index3.html; zugegriffen am 09.10.2012.